

빠르고 쉽게 휘뽕휘뽕 만드는 웹! Razor 와 WebMatrix 강좌 eBook

2010. 10. 1

(WebMatrix Beta 기준이며 정식 버전 발표시 업데이트 예정입니다. 이 eBook의 전체 내용은 <http://www.sqler.com> 에서 무료로, 제한 없이 보실 수 있습니다.)

SQLER 의 Konan 김대우(<http://www.sqler.com>)
김태형 역음(<http://tenisland.tistory.com>)

코난이와 함께하는
Razor & WebMatrix

Contents

(1) WebMatrix 와 Razor! 이제 시작합니다.....	3
(2) WebMatrix 설치부터 Hello World 까지	7
(3) Razor 강좌 - 기본 구문 및 주석 처리.....	16
(4) Razor 강좌 - 코드 블록과 POST 처리	21
(5) Razor 강좌 - 재사용 가능한 코드 생성	28
(6) Razor 강좌 - 레이아웃 페이지 구조 처리	32
(7) Razor 강좌 - 파일처리, 파일 업로드.....	44
(8) Razor 강좌 - 데이터베이스 처리	55
(9) Razor 강좌 - Helper 소개(이미지, 비디오).....	75
(10) Razor 강좌 - 디버깅.....	82
(11) Razor 강좌 - 캐시 처리.....	89
(12) Razor 강좌 - SMTP 메일전송(Live 메일과 Gmail 지원. SSL 지원).....	92
(13) Razor 강좌 - 웹사이트 전체, 또는 폴더 내 파일 요청 시 항상 실행 되는 모듈	95
(14) Razor 강좌 - URL 라우팅(Routing) 으로 SEO 최적화 구현	101

카테고리가 새로 생겨 약간 당황스러우시죠? 10년 넘게 운영 & 놀아터였던 저는 더욱 더 당황스럽습니다. ^_~;; 어쩌면 웹과 DB는 튀김과 떡볶이 같을지도 모르겠습니다. 둘을 따로 따로 즐겨도 되지만, 둘이 합쳐 시너지(?)를 내면 엄청난 결과(뭐지?)가 이루어지니까요. 이런 시너지나 저의 생각은 나중에 조금씩 더 풀어 보도록 하고, 오늘은 WebMatrix가 뭐길래? 이런 큰 결정 – 새로운 카테고리 와 섹션 제작 – 을 하게 되었는지 알아 보도록 하겠습니다.

(1) WebMatrix 와 Razor! 이제 시작합니다.

WebMatrix가 뭘니까?



WebMatrix는 웹 개발자가 웹사이트를 쉽고 빠르게 제작, 커스터마이징이 가능하도록 돕는 새로운 개발 도구 & 플랫폼입니다. 특히, 기존의 복잡한 웹 개발 방식을 심플하고, 직관적으로 개발 할 수 있도록 돕는, 개발 도구입니다.

참고 : WebMatrix 공식 웹사이트(<http://www.microsoft.com/web/webmatrix/>)

개발 도구만 제공되나요? 웹 개발에 필요한 웹 서버, 데이터베이스, 개발언어는요?

WebMatrix는 “개발 도구”이지만, 가장 최신의 웹 개발 도구이기에 기존 웹 개발자들의 요구를 수렴해 쉽고 빠른 웹사이트 제작에 특화되도록 개발 되었습니다. 특히, WebMatrix는 아래의 플랫폼 기술들을 포함하고 있습니다.

1) 웹 서버

IIS Developer Express입니다. 개발자가 웹사이트를 실행해 개발과 테스트를 진행하도록 돕는 가벼운 웹 서버를 자체 내장하고 있습니다. Visual Studio 2010 과 Visual Web Developer 2010 Express 버전과 연계해 동작 가능합니다.

2) 데이터베이스

SQL Server Compact Edition 4 가 WebMatrix 에 포함되어 있습니다. 무료 데이터베이스 엔진으로 .NET 기반의 API 를 제공해 WebMatrix 를 통한 손쉬운 웹 개발이 가능하며, SQL Server 로 쉬운 데이터 마이그레이션이 가능합니다.(WebMatrix 에서 자체 마이그레이션 도구를 제공합니다.)

3) 개발 프레임워크 - "Razor"

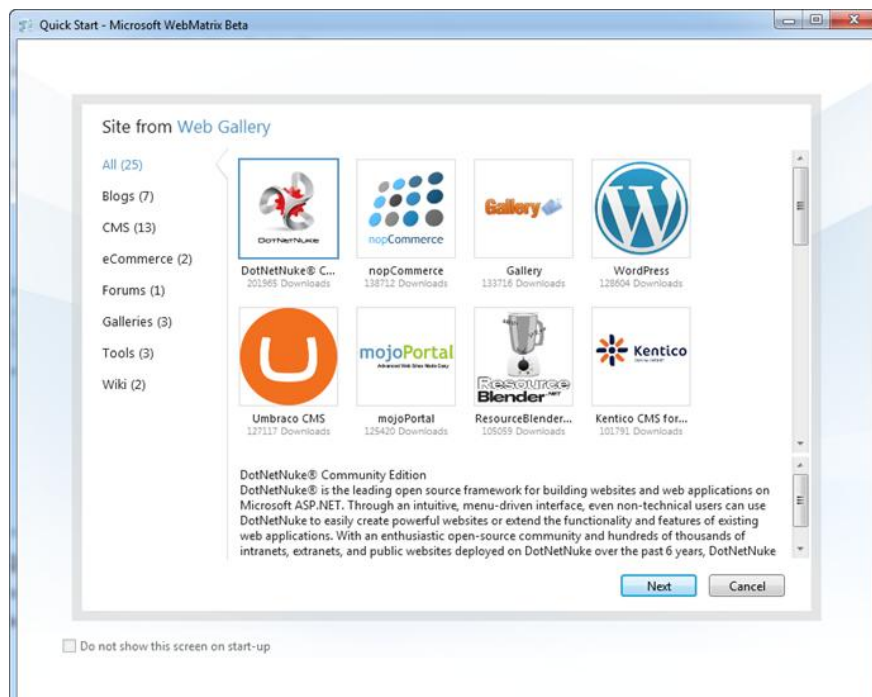
WebMatrix 는 "Razor"라는 ASP.NET 기반의 엔진을 지원합니다. 기존 웹 개발의 복잡성을 획기적으로 개선하고, ASP.NET MVC 의 성능과 안정성을 그대로 유지하면서 빠른 웹 개발에 최적화된 구문과 모듈화된 개발이 가능하도록 돕는 Helper 를 지원합니다.(Help 는 곧...)

코난이의 개인적인 느낌으로, 특히 더 쾌적한 데이터베이스 개발 관련 기능들을 제공하는게 좋더라구요~ DB 와 Razor 의 궁합도 대단히 좋습니다.

ASP.NET 개발자를 위한 Razor 추가 정보

"Razor"는 정확히 ASP.NET MVC 의 새로운 View 엔진 입니다. (와~~ MVC 만쉐이~) ASP.NET 에서 사용하던 언어인 C#이나 VB.NET 을 그대로 이용 가능하며(와~ 만만쉐이~) , 현재 국내 기업, 오픈마켓, 웹사이트 등에도 많이 적용된 ASP.NET 의 안정성과 확장성을 그대로 사용 가능하지요. **하나, 코난이와 주변분들의 피드백으로, Razor 는 기존의 웹폼이나 MVC 개발 방식과는 달리, Razor 는 MVC 가 도데체 어디 있다는거지? 할 정도로 쉽고 직관적인 개발이 가능합니다.** MVC 에 대해서는 나중에 한번 더 풀어 보도록 할게요. ^_~

아, WebMatrix 에서 개발과 커리어가 끝나는데 아닙니다. WebMatrix 의 웹 서버와 데이터베이스, 개발 프레임워크는 모두 자연스럽게 IIS 웹서버, SQL 서버 데이터베이스, ASP.NET 개발 프레임워크, Visual Studio 개발 도구로 통합 가능합니다.



새로운 웹사이트를 개발할 경우에도, 템플릿 기반으로 어플리케이션을 개발하거나, “웹 플랫폼 설치 관리자-Web Platform Installer”를 이용해 국내외 오픈 소스 어플리케이션을 설치한 후 우리에게 맞게 커스터마이징 하는 형태로 이용도 가능합니다. – 맨땅에 헤딩하면서 웹사이트 개발할 필요 없어욧(템플릿을 기반으로한 개발도 우리에게 너무너무 잘 맞아요~ 쿨~)!!!

Publishing Settings

First time publishing? Add existing server information below or [find web hosting.](#)

Remote Server

Protocol: **Web Deploy** (FTP, FTP/SSL, Web Deploy)

Server:

Username:

Password:

Site Name:

Destination URL:

☐ Save password

또한, 개발과정과 테스트 과정을 진행한 후 스테이징이나 프리덕션을 진행하실 경우에도 출판(Publishing) 과정을 모두 자체적으로 지원하기 때문에 쉽고 빠른 웹사이트 제작이 가능해요.

FirstSite - Microsoft WebMatrix Beta

Home Table

New Table Definition Data New Column Delete Column Indexes Relationships Refresh Delete Row

FirstSite

FirstSite.sdf

Tables Other Connections

(FirstSite.sdf).NewTable_1

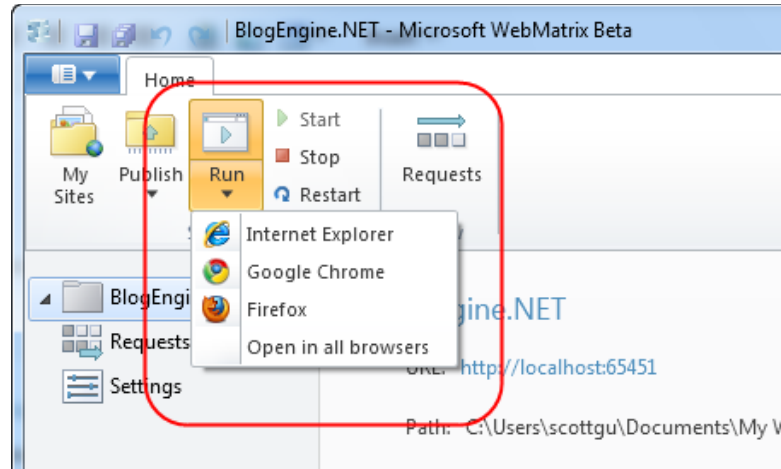
Column Name	Data Type	Allow Nulls
id	int	False
Name	nvarchar	True
Year	int	True

Column Properties

(Name)	id
Allow Nulls	False
Data Type	int
Default Value	
Is Identity?	True
Is Primary Key?	True

Site Files Databases Reports

특히, 앞에서도 언급 드린 것처럼 데이터베이스 테이블, 관계, 제약, 인덱스 등의 구조를 WebMatrix 자체에서 개발하고, SQL 서버로 쉽게 마이그레이션 가능하기 때문에 마이그레이션이나 배포 관련 고민이 줄어듭니다. – SSMS(SQL Server Management Studio)를 열 필요가 없습니다.



IIS Developer Express 도 자체 내장이라, 개발 과정에서 웹 서버 관련 작업을 진행하고 시작과 종지를 WebMatrix 자체에서 진행이 가능해요. 저기 위에 Start, Stop, Restart 아이콘 보이시죠?

개발 하면서 브라우저에서 테스트를 해 보고 싶으실 경우에도 버튼 하나로, 또는 F12 번키만 뽐 눌러주면 바로 결과를 웹에서 확인 가능합니다. – 보이시죠? 크로스 브라우저 기반 테스트는 기본이예요. 브라우저 열고 URL 넣어서 테스트하고, 이런 과정이 필요가 없습니다.

요약해 드리자면!!!

쉽고 빠른 웹사이트 개발에 최적화된 개발 도구 & 플랫폼! 바로 WebMatrix 입니다. 다음 강좌에서는 실제로 이용하는 과정을 보여 드리도록 할게요.

참고링크 :

Introducing WebMatrix (<http://www.sqler.com/scottgu/archive/2010/07/06/introducing-webmatrix.aspx>)

Introducing "Razor" – a new view engine for ASP.NET(<http://www.sqler.com/scottgu/archive/2010/07/02/introducing-razor.aspx>)

New Embedded Database Support with ASP.NET (<http://www.sqler.com/scottgu/archive/2010/06/30/new-embedded-database-support-with-asp-net.aspx>)

Introducing IIS Express(<http://www.sqler.com/scottgu/archive/2010/06/28/introducing-iis-express.aspx>)

(2) WebMatrix 설치부터 Hello World 까지

어제의 강좌에 이어서, 이번에는 “(2) WebMatrix 설치부터 Hello World 까지”라는 내용으로 진행해 보도록 하겠습니다.

사실 “설치”라고 말씀 드리기 어색할 정도로 그 과정이 쉬워서 찝끔~ 민망하기도 하네요(그만큼, 설치 자체는 쉽다는 이야기에요~)

그럼 시작해 보겠습니다.

WebMatrix 를 설치하기 전에 꼭 알아두시면 좋습니다.

최근 마이크로소프트가 웹에서 작지만 여러 개발자와 관리자에게 도움이 되는 일들을 많이 진행하고 있었는데요. 그 중 하나가 바로 Web Platform 입니다. 제가 예전에 작성해둔 강좌도 옆에 있는데요.

마이크로소프트 웹 플랫폼 - (1) 다시 쓰는 웹 플랫폼 (<http://www.sqler.com/193028>)

마이크로소프트 웹 플랫폼 - (2) 마이크로소프트 웹 플랫폼은 무엇인가? (<http://www.sqler.com/193035>)

마이크로소프트 웹 플랫폼에서 제공하는 다양한 개발환경, 데이터베이스, 프레임워크, 심지어는 오픈 소스 소프트웨어들을 하나의 어플리케이션으로 모두 설치/업데이트 하실 수 있습니다. 그게 바로 **“웹 플랫폼 설치 관리자 – Web Platform Installer(이하 WPI)”** 입니다.

이 “웹 플랫폼 설치 관리자”(이하 WPI)를 이용하시면 여러 장점들이 있는데요.

- 복잡한 웹서버 설치, 유지 보수 환경 구성을 클릭 딱 “한번”에 끝낼 수 있도록 쉬운 설치를 제공합니다.
- 개발환경과 배포환경, 웹서버 에서 일관적으로 어플리케이션에 필요한 구성요소들과 프레임워크, 데이터베이스, 관리 도구 등을 유지 가능합니다.
- 웹사이트 및 웹 어플리케이션 제작에 필요한 웹서버 기능, 프레임워크, 데이터베이스, 개발환경을 이 WPI 안에서 모두 제어가 가능합니다.
- 국내외 최고 수준의 오픈 소스 소프트웨어(OSS)를 클릭 한번으로 바로 설치 – 사용이 가능하며, 국내에서 가장 많이 사용되는 XpressEngine(구, 제로보드), Textile, KimsQ 와 같은 토종 오픈 소스 소프트웨어도 이 WPI 에서 한번 클릭으로 설치가 가능합니다. (아~ 지금 보고 계신 이 www.sqler.com 웹사이트도 이 WPI 의 XpressEngine 클릭 한번으로 제작된 커뮤니티입니다.)

오늘 우리의 목표인 WebMatrix 설치 도 마찬가지로 입니다. 이 WPI 를 통해서만 설치 가능합니다. – WPI 에 대한 이야기는 나중에 천천히 더 풀어 보도록 하겠습니다. – 현재 Beta 버전이지만, 이런 설치에 대한 큰 그림은 바뀌지 않을 겁니다. ^_^

WPI 관련 참고 링크 : <http://www.microsoft.com/web/downloads/platform.aspx>

설치 패키지 크기

IIS Developer Express 웹서버, WebMatrix 개발도구, SQL CE 데이터베이스, Razor 개발 엔진이 포함된 WebMatrix 의 설치 패키지 크기는 단 15M 입니다. 현재 Beta1 이지만, 공식 버전도 이 크기를 유지할 예정이라고 하며, 설치 과정은 1 분 이내 다운로드 및 설치가 가능하도록 목표로 하고 있습니다. 만약, .NET Framework4 가 설치되어 있지 않다면 설치가 필요하며 포함한 전체 패키지는 50M 입니다.

설치 가능한 시스템 환경

현재 WebMatrix 는 2010 년 7 월 8 일 현재 Beta1 환경입니다. 지원하는 시스템 환경은 아래와 같으니 참고 하시길 바랍니다.(공식 버전에서는 지원하는 개발 환경이 추가될 수도 있다고 하네요.)

Windows XP Pro SP3 이상, Windows Vista SP1 이상, Windows 7, Windows Server 2003 SP2 이상, Windows Server 2008, Windows 2008 R2 에서 설치가 가능합니다. 사실상 XP 부터 모든 환경을 지원한다고 보시면 좋을 것 같아요.

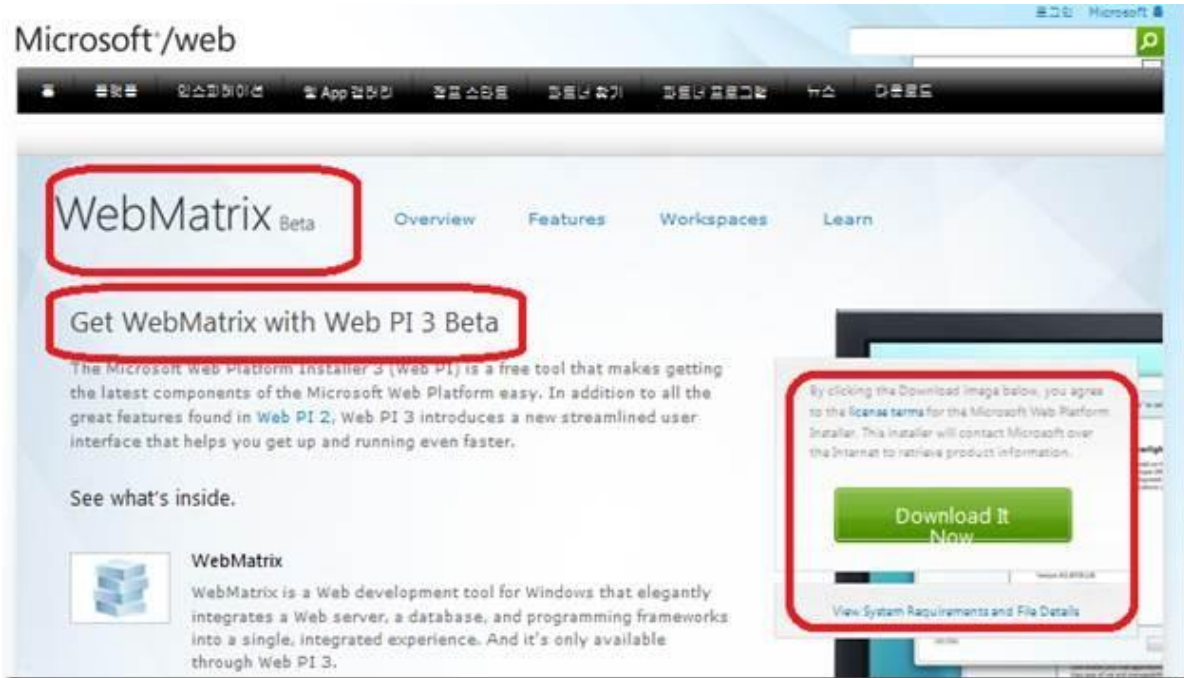
아, 저 코난이는 당연히 새술은 새 부대에! 제가 진행하게 될 WebMatrix 와 Razor 강좌는 모두 Windows7 환경으로 진행하게 될 겁니다.

설치 진행!!! – 살짝 민망하지만, 클릭질 한번에 끝납니다. 저와 설치를 진행해 보시죠~

우선 WebMatrix 공식 사이트에 접속합니다.(WebMatrix 공식 버전 발표 전후로 한글 지원 예정이 있다고 합니다.)

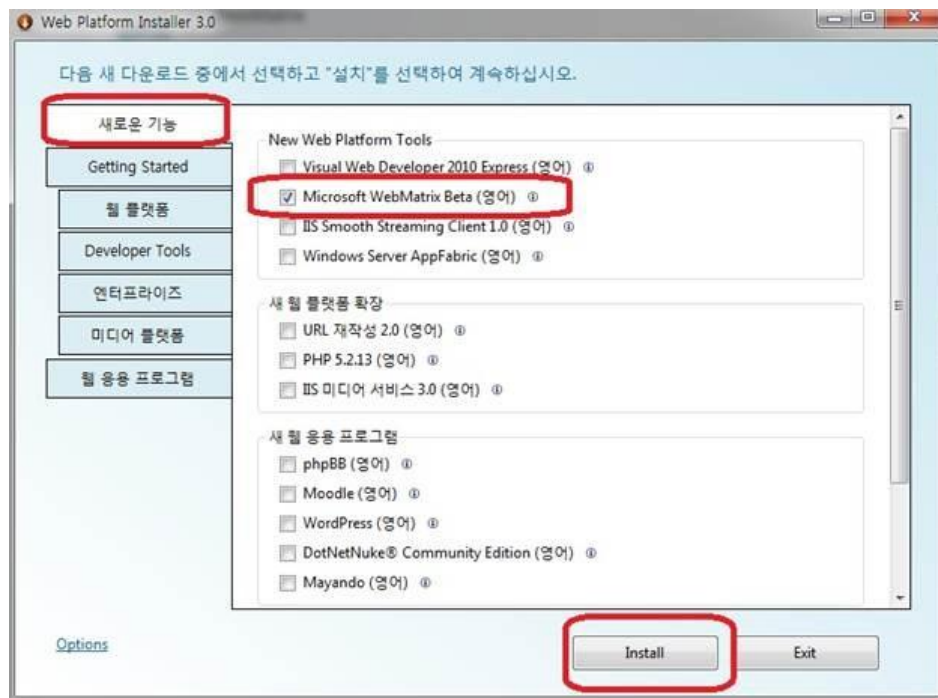


공식 사이트 링크 : <http://www.microsoft.com/web/webmatrix/>



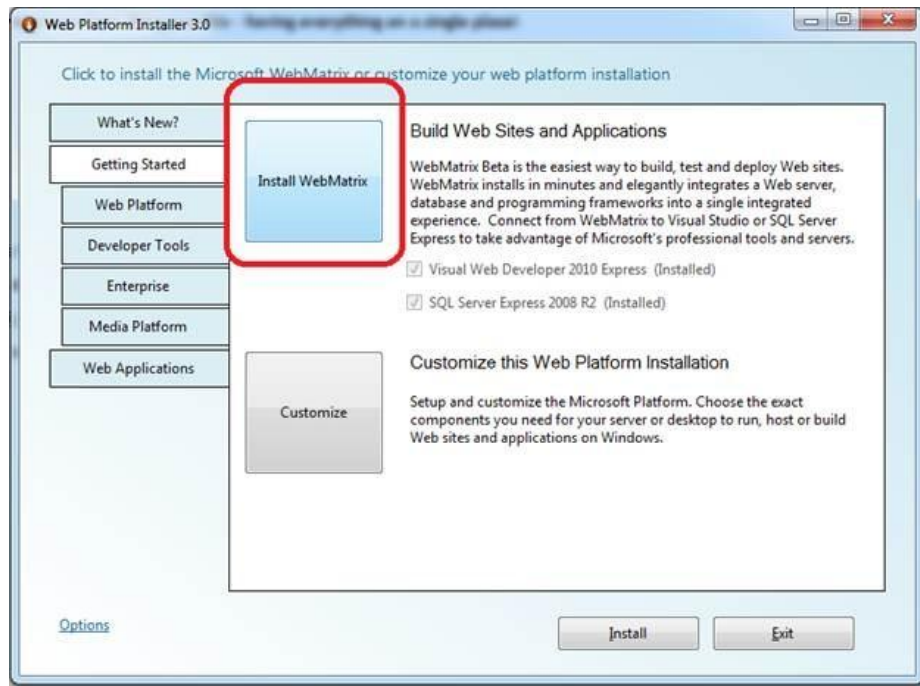
이어서 우측 맨 위에 초록색 Download 클릭하시면 앞에서 소개해 드린 “웹 플랫폼 설치 관리자 – Web Platform Installer – WPI” 설치 화면이 보입니다. 오른쪽의 초록색 다운로드를 클릭해 주세요.

바로 다운로드 : <http://www.microsoft.com/web/webmatrix/download/>

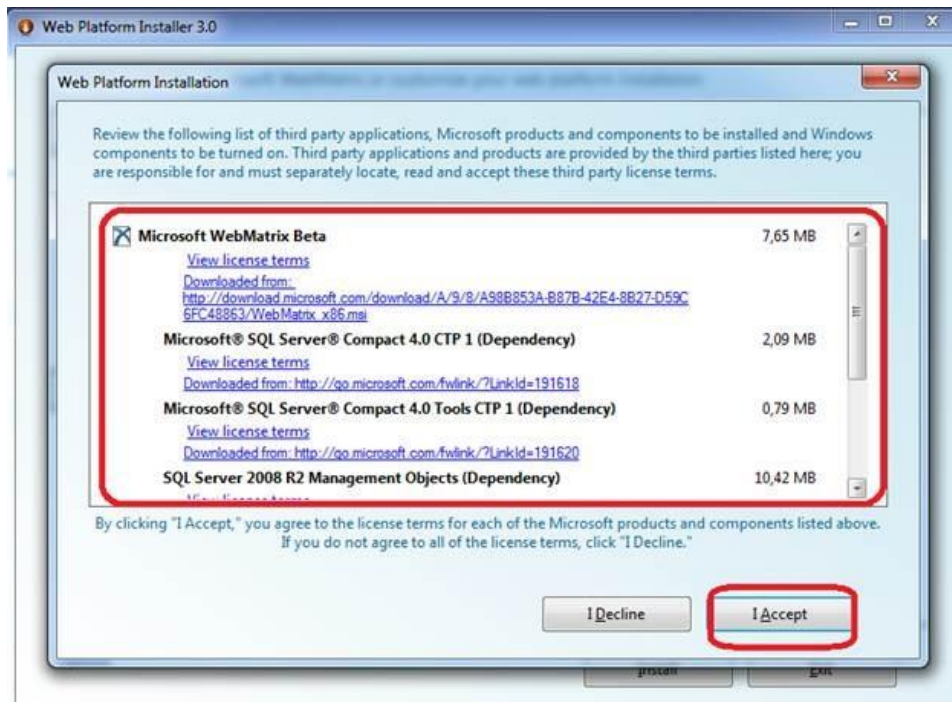


자~ 설치가 끝나시면 앞에서 소개해 드린 “웹 플랫폼 설치 관리자-WPI”가 실행됩니다.

참고로, 이 WPI 는 3.0 Beta 버전이에요. 여기 에서 “Microsoft WebMatrix Beta”를 선택하고 install 을 클릭!
(현재 WPI 는 2.0 정식 버전이 한글로 모두 서비스 되고 있습니다. WPI 3.0 역시 정식 버전 발표시 한글을 지원할 예정이라고 합니다.)

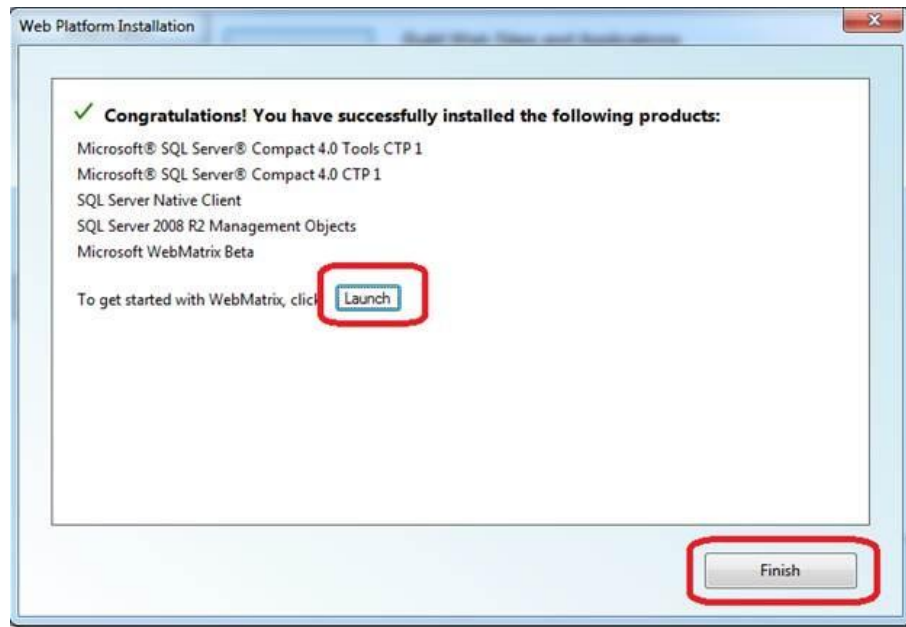


영문 환경이실 경우 요렇게 나올 수 도 있으니 참고 하시길 바랍니다. 마찬가지로, "Install WebMatrix"를 클릭하시면 됩니다.



WebMatrix 를 선택하시면 WebMatrix 실행에 필요한 종속된 프로그램들이 주르륵 자동으로 올라옵니다.(WPI 썬유!!!)

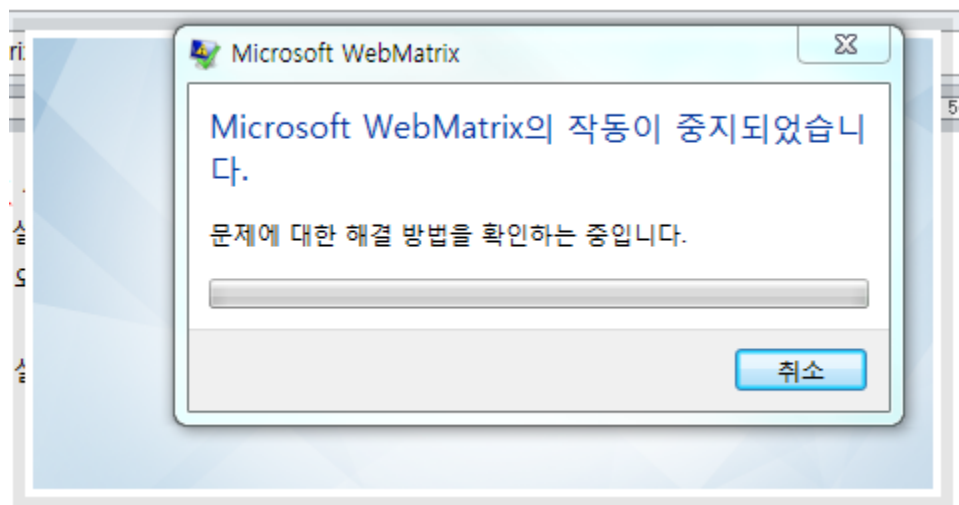
찬찬히 어떤 녀석들이 설치되는지 살펴 보세요. (WPI의 장점!!! 기존에는 개발자와 관리자가 개별적으로 구성 했을 겁니다. 개별 구성했다가는 설치만 3박 4일 걸릴지도. 쿨럭... 보시는 것처럼, WPI가 이런 복잡한 설치/개발환경 구성을 깔끔하게 자동화/단순화/유지보수 해 줍니다.) 이제 설치가 자동으로 진행 되실 거예요~



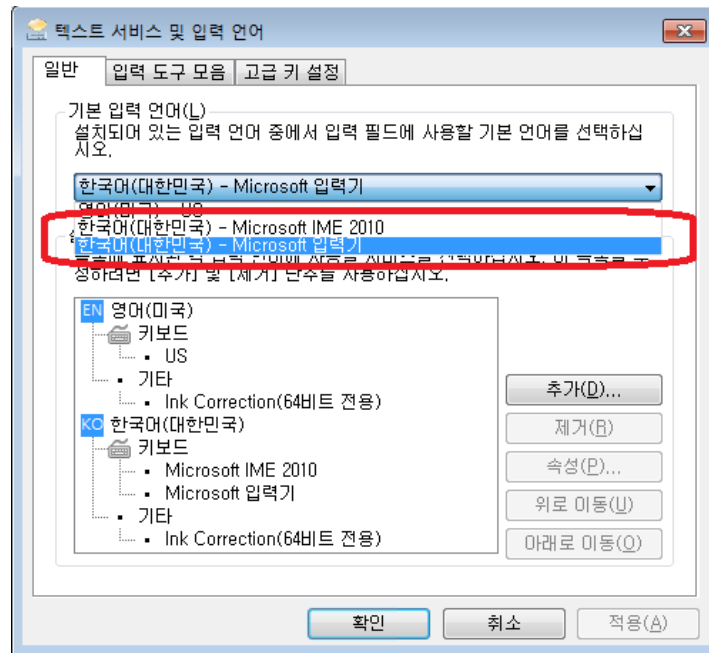
이렇게 설치가 끝나면 Launch를 누르셔서 바로 WebMatrix를 실행하시거나, Finish를 눌러 종료하시고, Windows7의 시작을 눌러 WebMatrix를 실행 하셔도 됩니다.

참고 : 처음 실행 하실때 WebMatrix가 실행되지 않고, 아래 화면처럼 비정상 종료하실 경우가 있습니다.

"Microsoft WebMatrix의 작동이 중지되었습니다." - 후덜덜~



이 문제는 Beta 버전에서만 발생하는 "한글 입력기"와 관련된 문제임을 확인했으며, 아래와 같은 방법으로 조치 가능합니다.



WebMatrix 가 실행되지 않고 비정상 종료 되실 경우에만, 위에서 보시는 것처럼 제어판 - 국가 및 언어 - “키보드 변경” 버튼 클릭 - “기본 입력 언어”를 “한국어(대한민국) - Microsoft 입력기”로 변경 - 확인 하시면 됩니다. 이 이슈는 오직 현재 "베타" 버전의 WebMatrix 에서만 발생하며, 이후 버전에서는 이 프로세스 비정상 종료 문제는 해결될 예정입니다.

바로 클릭해서 실행하시면 이렇게~ WebMatrix 시작 화면을 보실 수 있습니다!!!



WebMatrix 설치 과정 요약!

- (1) “웹 플랫폼 설치 관리자”를 실행하고 - <http://www.microsoft.com/web/webmatrix/download/>
- (2) WebMatrix 를 클릭한다.
- (3) 끝~ 참 쉽죠잉~

참고로, Simple, Small, Seamless 가 WebMatrix 의 설계 철학이라고 하네요. 설치 과정에서 보신 것처럼, Small 과 Simple 은 확실히 철학 맞는 것 같습니다.(Razor 를 이제 저랑 공부하시면 더욱 더 확실히 Simple 의 철학을 느끼실 것 같아요!!!)

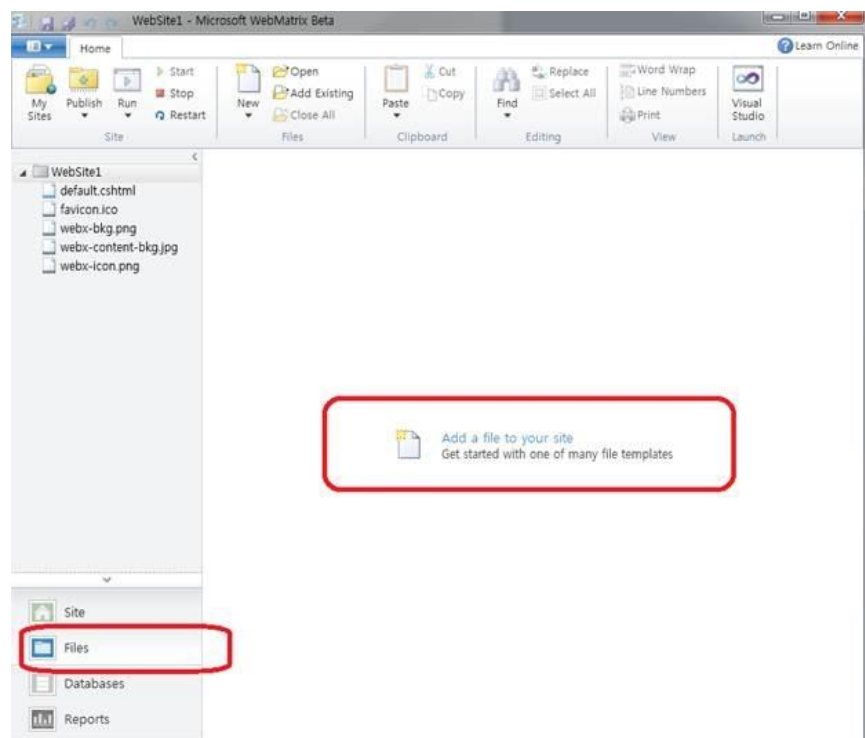
자 이제 Hello World 를 찍어 볼까요~

자 이제 WebMatrix 처음 시작 화면에서 My Sites 를 선택하고, 기본으로 되어있는 WebSite1 을 선택합니다.

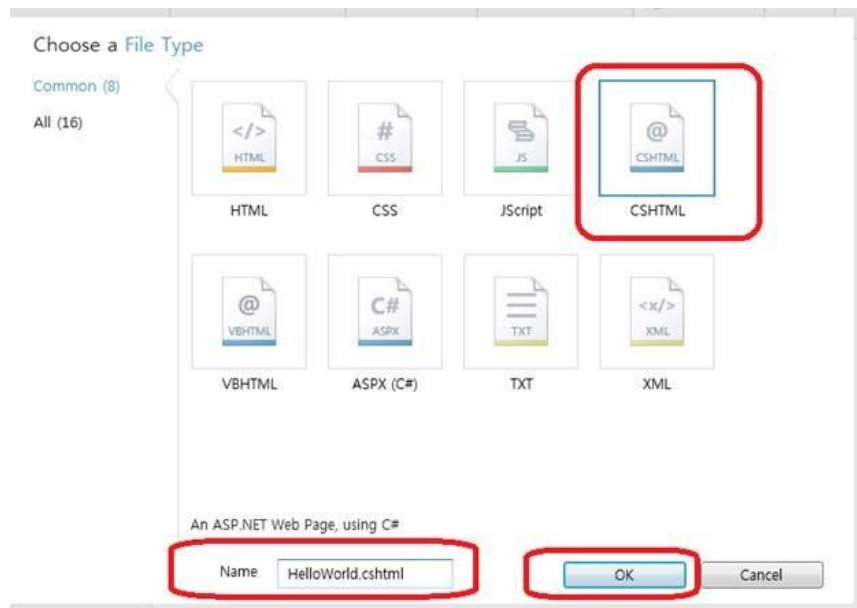


드디어 잘 모르겠지만, WebMatrix 가 시작된 것 같습니다.

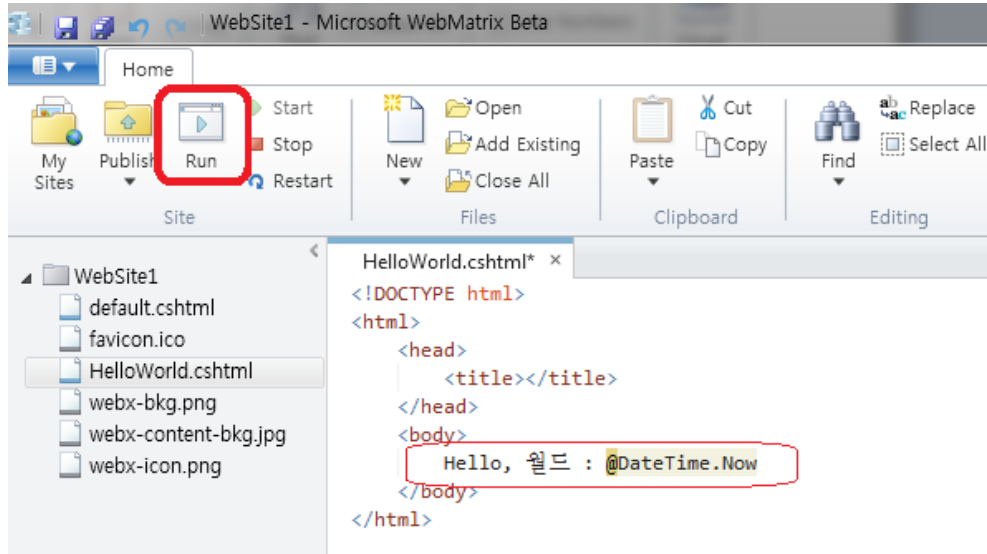
넵~ 바로 첫 번째 WebMatrix 를 이용한 Razor 개발!!! 시작합니다.



왼쪽 하단의 “Files”를 클릭하고, 파일을 사이트에 추가 합니다. – ”Add a file to your site” 클릭합니다.



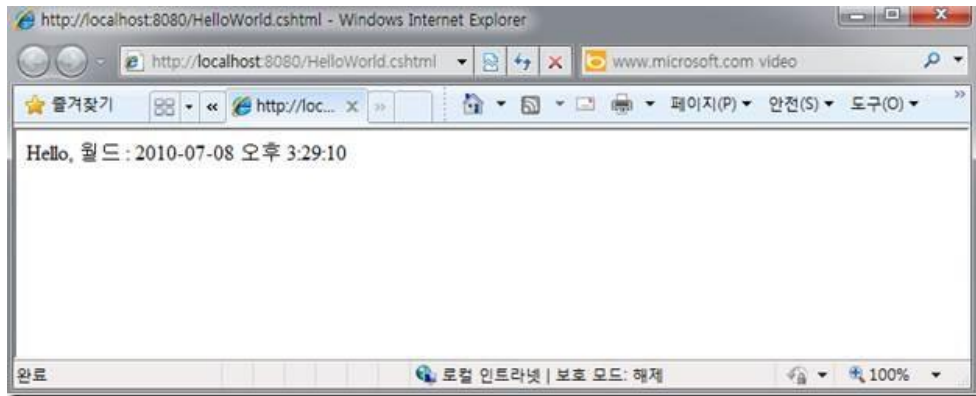
“Razor”를 실행하기 위해~ CSHTML 을 선택하고, 이름을 HelloWorld.cshtml 로 지정한 다음 OK 를 클릭!



기본 HTML 코드가 완성되었을 겁니다. 중간 <Body> 영역에 아래 한 줄을 추가합니다.

Hello, 월드 : @DateTime.Now

한 줄을 추가하고, F12 번 키를 누르거나 위의 “Run” 버튼을 누르시면 끝! (@ 선언자를 이용하지요.)



출력 완료~ (한글처리, 결과의 한글 낱자 출력 문제 없음을 확인해 보세요.)

자. 이렇게 WebMatrix 설치부터 Hello World 까지 하나의 포스트로 진행해 보았습니다. 대충 WebMatrix 설치에 대해서 감이 잡히시나요?

오늘 진행한 내용을 요약해 보면

- “웹 플랫폼 설치 관리자-WPI”는 웹사이트 제작에 필요한 다양한 프레임워크, 개발환경, 데이터베이스 등을 관리하는 최고의 설치 관리자입니다. 이 녀석 하나만 있으면 개발, 테스트, 실제 프러덕션 환경 모두에 꼭 맞는 웹서버 운영 환경 구축이 가능해요. 나름 깔끔한 녀석인 것 같네요. ^_^
- WebMatrix 는 이 WPI 로 클릭질 한번이면 설치가 완료됩니다. 설치 크기는 15M 내외(.NET Framework 제외). 설치 시간 1 분.
- 그리고, HelloWorld 까지 진행해 보았습니다.

자~ 그럼 다음 시간에는”(3) WebMatrix 다양한 기능 활용”이라는 내용으로 WebMatrix 에 대한 기본적인 내용을 마치고, 다음 챕터 부터는 “Razor”를 이용한 개발 과정을 상세히 짚어 보도록 할게요~ 감사합니다.

참고링크 :

[Introducing WebMatrix](#)

[Introducing “Razor” – a new view engine for ASP.NET](#)

[New Embedded Database Support with ASP.NET](#)

[Introducing IIS Express](#)

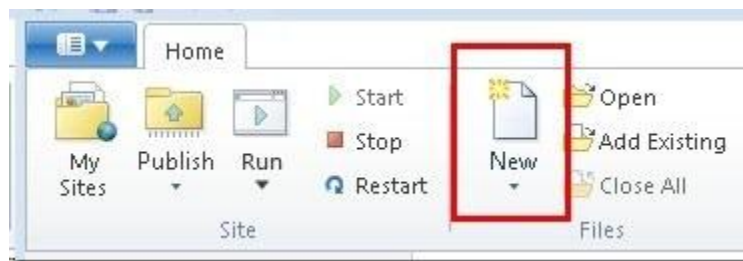
(3) Razor 강좌 - 기본 구문 및 주석 처리

안녕하세요. 코난 김대우 입니다. 지난 시간에는 WebMatrix 로 Hello World 를 찍어 보셨고, WebMatrix 여러 편리한 기능들에 대해서도 소개를 해 드리는 시간을 가졌습니다. 이번 시간에는 따라 하다 보면 끝나는 Razor 이야기를 진행해 보려고 합니다.(꼬옥 따라해 보세요~)

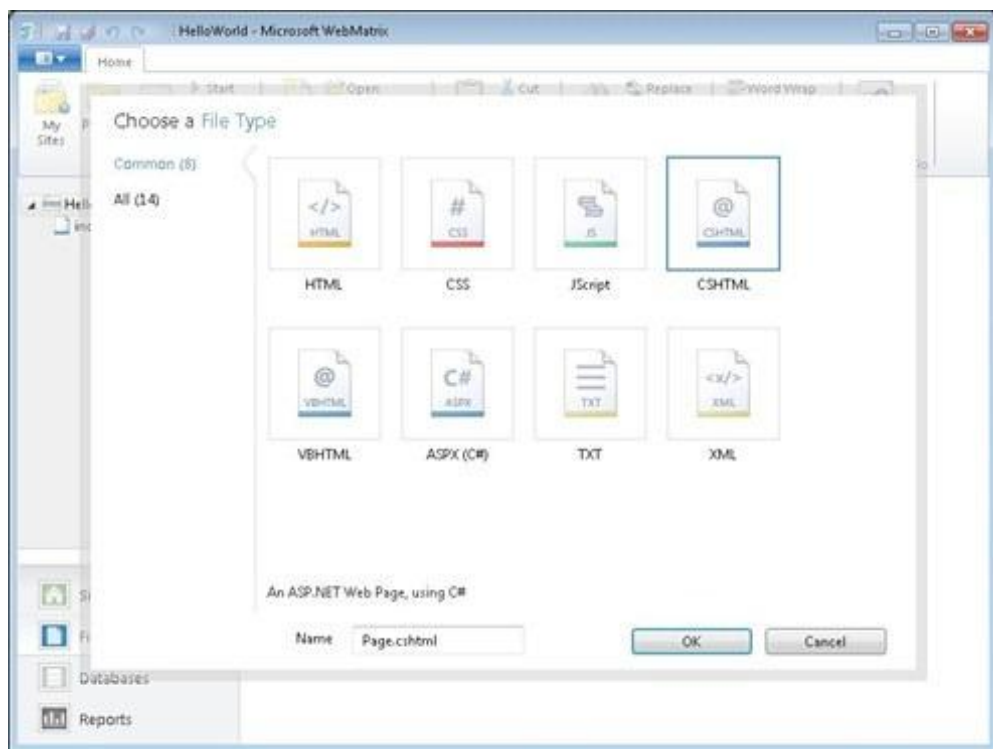
단순히, 박스 안에 들어가 있는 코드를 복사해 WebMatrix 에서 실행하시면 Razor 를 공부할 준비는 끝날 것 같습니다.

그럼 시작해 보도록 할게요.

WebMatrix 에서 Razor 실행을 위해 CSHTML 파일을 생성하고 코드를 수행합니다.



WebMatrix 에서 기본 설치되어 있는 Website1 를 선택 하시고, 왼쪽 아래 섹션에서 Files 를 선택합니다. 위의 화면처럼 New 버튼을 클릭 하시고 CSHTML – 바로 Razor 파일이지요! – 를 선택하고 파일 이름을 적어 주세요.



자. 기본 HTML 구문이 들어가 있을 텐데요. 과감히 모두 삭제 하시고 바로 이어서 Razor 코딩을 시작합니다!

심플한 Razor 의 기본적인 구문(Syntax)를 알아 봅니다.

1) “@” 문자를 이용해 코드를 시작합니다. - 인라인 표현식이에요.

```
<h2>인라인 표현식</h2>
```

```
@// @인라인 표현식(Inline Expression) 사용
```

```
<p>오늘은 @DateTime.Now 입니다.</p>
```

- “@” 문자로 Razor 코드 블록을 선언할 수 있습니다.

- 코드 블록 안에서 한 줄 주석 처리는 // 구문으로 가능합니다.

(알고 있으시죠? 위 박스 안의 코드를 복사해 WebMatrix 에 넣고 F12 번 키를 눌러 바로 실행 해 보실 수 있습니다.)

2) 함수 호출과 코드 블록

```
<h2>함수 호출</h2>
```

```
오늘은 @DateTime.DaysInMonth(2010, 5) 일 입니다.
```

```
<h2>Razor 가 실행되는 웹서버의 정보 표시</h2>
```

```
@ServerInfo.GetHtml()
```

```
<h2>C# 스타일의 전체 괄호 처리 - ";"으로 종료</h2>
```

```
@{
```

```
/*
```

```
코드 블록 안에서 여러줄의 주석은 이렇게 처리합니다.
```

```
*/
```

```
var showToday = true;
```

```
if(showToday)
```

```
{
```

```
    @DateTime.UtcNow;
```

```
}
```

```

else
{
    @DateTime.Now;
}
}

```

- 함수 처리 방식입니다. (기존에 아무 언어나 개별 경험이 있는 분들은 익숙하실 거예요.)
- 여러 줄의 코드 블록을 이용하실 경우에는 이렇게 괄호 - {} - 을 이용합니다. 코드 블록 안에서 구문 완료 후에는 “;”으로 코드의 끝을 지정합니다. (C# 하신 분들은 익숙하실 거예요.)
- ServerInfo.GetHtml() 함수로 Razor 가 실행되는 웹 서버의 설정 정보 등을 보실 수 있습니다.(디버깅에도 유용합니다.)

3) 반복문 사용

```

<h2>HTML 조합 반복문</h2>
@{
    <ul>
        @foreach (string item in Request.ServerVariables)
        {
            <li>@item</li>
        }
    </ul>
}

```

- 코드 블록 안에서 HTML 구문을 조합 가능합니다.
- 어떤 Server Variable 들이 있는지 결과를 참조하세요.

전체 코드입니다.

```

<h2>인라인 표현식</h2>
@// @인라인 표현식(Inline Expression) 사용
<p>오늘은 @DateTime.Now 입니다.</p>
<h2>함수 호출</h2>

```

오늘은 @DateTime.DaysInMonth(2010, 5) 일 입니다.

<h2>Razor 가 실행되는 웹서버의 정보 표시</h2>

```
@ServerInfo.GetHtml()
```

<h2>C# 스타일의 전체 괄호 처리 - ";"으로 종료</h2>

```
@{  
  
    /*  
        코드 블록 안에서 여러줄의 주석은  
        이렇게 처리합니다.  
    */  
  
    var showToday = true;  
    if(showToday)  
    {  
        @DateTime.UtcNow;  
    }  
    else  
    {  
        @DateTime.Now;  
    }  
}
```

<h2>HTML 조합 반복문</h2>

```
@{  
    <ul>  
        @foreach (string item in Request.ServerVariables)  
        {  
            <li>@item</li>  
        }  
    </ul>  
}
```

너무 간단한가요? 이렇게 간단히 Razor 의 기본 구문을 알아 보았습니다. - 간단한거 몇줄 안한거 같은데 해본거 참 많죠?

- 인라인 표현식
 - 주석처리
 - 함수(메서드) 호출
 - ServerInfo 표시
 - 코드블록 처리
 - 반복문 처리
 - Server Variable 정보
- 등을 살펴 보셨습니다.

아마, 기존에 웹개발 경험이 있으시다면, 이번에 처음 손맛을 보신 Razor 구문이 새롭게 디자인된 웹 개발 구문인 만큼, 직관적이고 깔끔한 코드를 제공한다고 느껴지실 거예요.

뿐만 아니라 WebMatrix 를 통해 직접 코딩을 하시면서 깔끔한 코딩 화면이나 코드 블록 자동 하이라이팅 처리 등을 보시면 심플한 개발을 손으로 조금은 느껴 보실 듯 합니다.

Razor 가 제공하는 좀더 상세한 개발 관련 구문 등을 보기 원하시면 아래 링크를 참고 하시길 바랍니다.

[2 - Introduction to ASP.NET Web Programming Using the Razor Syntax](http://www.asp.net/webmatrix/tutorials/2-introduction-to-asp-net-web-programming-using-the-razor-syntax)(<http://www.asp.net/webmatrix/tutorials/2-introduction-to-asp-net-web-programming-using-the-razor-syntax>)

이렇게 해서 **(3) Razor 강좌 - 기본 구문 및 주석 처리** 강좌를 마무리 했습니다. 다음 시간에는 **(4) Razor 강좌 - 코드 블록과 POST 처리** 내용을 살펴 보도록 하겠습니다. 감사합니다.

참고자료 :

[1 - Getting Started with WebMatrix Beta and ASP.NET Web Pages](#)

[2 - Introduction to ASP.NET Web Programming Using the Razor Syntax](#)

(4) Razor 강좌 - 코드 블록과 POST 처리

지난 시간에는 “(3) Razor 강좌 - 기본 구문 및 주석 처리” 강좌를 진행 했습니다.

이번 시간에는 바로 **이어서 네번째 강좌**, “(4) Razor 강좌 - 코드 블록과 POST 처리” 강좌를 진행 하도록 하겠습니다.

코드블록 정리

지난 시간에 간단히, 코드블록에 대해서 설명해 드렸습니다. 넵, 간단히 Razor 구문을 사용해 프로그램 로직을 만드는 영역 정도로 봐 주시면 될 것 같아요. 혹시 웹 프로그래밍을 개발하신 경험이 있으시다면, 다시 한번 코드 블록을 봐 보세요. 기존의 웹 개발 언어, ASP.NET 이나 PHP 와는 조금 다르지 않으신지요? 예를들어, Razor 구문 코드 블록 안에 자유롭게 HTML 을 끼워 넣을 수 있는 부분이에요.

```
<h2>HTML 조합 반복문</h2>
@{
    <ul>
        @foreach (string item in Request.ServerVariables)
        {
            <li>@item</li>
        }
    </ul>
}
```

두 번째 줄의 HTML 태그 앞이나 뒤에 Razor 구문 끝과 시작을 알리는 식별문자(@)가 포함되지 않고도, 코드 블록으로 처리 됩니다. ASP 나 PHP 개발자라면, 이런 인라인 코딩 방식이 조금은 “**땡큐하게**” 다가 오실지 모르겠습니다. ^_^

Razor 는 웹 개발자의 요구로 개발된 가장 최신의 웹 개발 구문입니다. 작고 심플한 웹 개발을 그 목표로 개발 되었다고 하네요(안그런게 녀석이 있겠습니까마는... ^_~;;)

Razor 는 웹 개발을 단순하고 빠르게 수행하기 위한 수 많은 개발자들의 요구를 수렴해 개발 되어서 이런 부분도 신경쓰게 조금 보이는거 같아요. 아직 감이 잘 안 잡히신다고요? 조금 있으면 Razor 로 데이터베이스를 처리하는 챕터가 진행되는데요. ASP 나 PHP 개발을 해오신 분이라면, 눈물나게 땡큐한(?) Razor 의 이 심플 코드블록을 느끼실 수 있을 겁니다. 그 복잡한 데이터베이스 결과를 디자이너가 건네 준 퍼즐같은 복잡한 UI 에 맞춰서 HTML 사이사이에 곡예 하듯 끼워 맞추던(그냥 들어내고 그리드 컨트롤 박아 넣었던... -_-;;;) 그 복잡함을 많이 덜어 버리실 수 있을 거예요.

현재 Razor beta 버전인데요. **Razor 의 강력한 기능을 살린 다양한 중고급 개발자를 위한 고급 개발 패턴도 함께** 준비되고 있는 것 같아요. 또한, **마이크로소프트의 클라우드 컴퓨팅 기술인 Azure 에 Razor 로 개발한 웹 어플리케이션이 올라가면** 어떻게 될까요? 더 빠르고 쉽게 웹 애플리케이션 개발이 가능해 질겁니다.

넵~ 기회 있을 때마다 이런 소식들도 모아모아서~ 강좌 게시판을 통해 소개해 드리도록 하겠습니다. 그럼 바로 이어서, POST 처리-HTML Form 값 처리에 대한 내용을 진행해 보도록 하겠습니다.

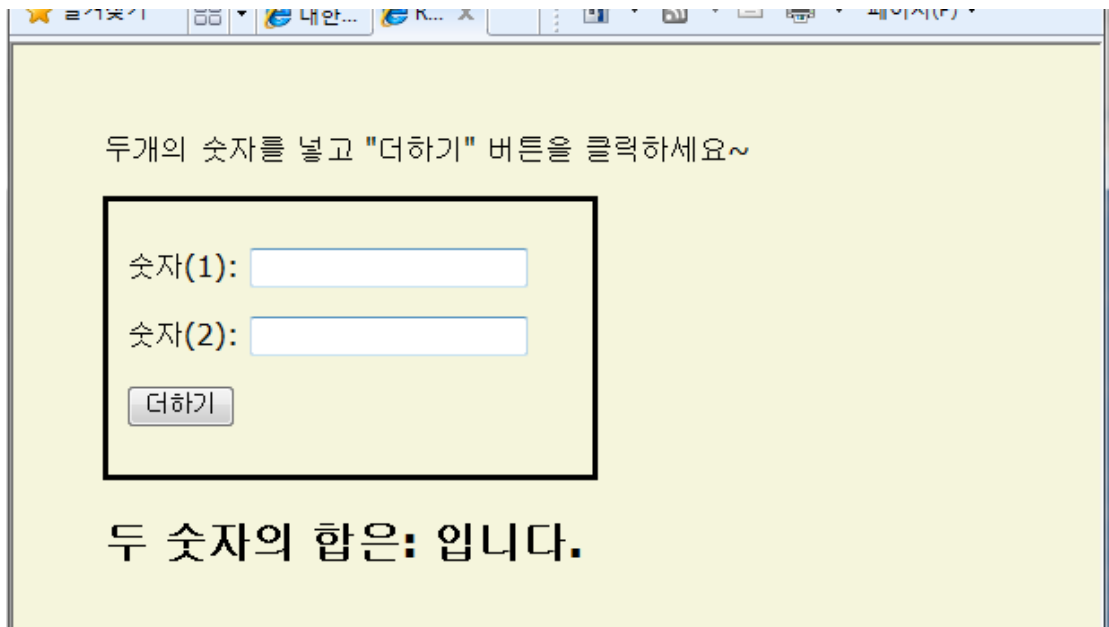
POST 처리 - HTML Form 값 처리

먼저 간단한 값을 텍스트박스로 입력 받아 값을 더하는 HTML 페이지 예제를 생각해 보도록 하세요.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-  
transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">  
  <head>  
    <title>Razor 로 HTML Form 값을 처리</title>  
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />  
    <style type="text/css">  
      body {  
        background-color: beige; font-family: Verdana, Arial; margin: 50px;  
      }  
      form {  
        padding: 10px; border-style: solid; width: 250px;  
      }  
    </style>  
  </head>  
  <body>  
    <p>두개의 숫자를 넣고 "더하기" 버튼을 클릭하세요~</p>  
    <p></p>  
    <form action="" method="post">  
      <p>  
        <label for="text1">숫자\(1\):</label>  
        <input type="text" name="text1" />  
      </p>  
      <p>  
        <label for="text2">숫자\(2\):</label>  
        <input type="text" name="text2" />  
      </p>  
      <p>  
        <input type="submit" value="더하기" />  
      </p>  
    </form>
```

```
<h2>두 숫자의 합은: 입니다.</h2>
</body>
</html>
```

이런 HTML 페이지를 생각해 보시면 될 것 같습니다. - WebMatrix 에서 Razor 를 실행하기 위해 CSHTML 파일을 하나 생성하고, 위의 HTML 코드를 마찬가지로 붙여 넣으신 후 실행해 보세요.



대충~ 이런 화면이 나오실 겁니다. 여기 Form 의 입력 값을 Post 로 받아 처리한다는거 감이 오시죠? 그럼 실제로 코드를 입력 보도록 하지요.

```
@{
var total = 0;

//HTML 의 Form 에서 POST 될 경우에만 실행
if(IsPost) {
    // 사용자의 입력값을 받아서 저장
    var num1 = Request["text1"];
    var num2 = Request["text2"];
    // 넘겨받은 값을 Integer 형으로 변환 후 더하기 수행
    total = num1.AsInt() + num2.AsInt();
}
```

```

}

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

  <head>
    <title>Razor 로 HTML Form 값을 처리</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
    <style type="text/css">
      body {
        background-color: beige; font-family: Verdana, Ariel; margin: 50px;
      }
      form {
        padding: 10px; border-style: solid; width: 250px;
      }
    </style>
  </head>
  <body>
    <p>두개의 숫자를 넣고 "더하기" 버튼을 클릭하세요~</p>
    <p></p>
    <form action="" method="post">
      <p>
        <label for="text1">숫자(1):</label>
        <input type="text" name="text1" />
      </p>
      <p>
        <label for="text2">숫자(2):</label>
        <input type="text" name="text2" />
      </p>
      <p>
        <input type="submit" value="더하기" />
      </p>
    </form>

    @// 더한 값이 아래 표시 됩니다.
    <h2>두 숫자의 합은: @total 입니다.</h2>
  </body>

```

```
</html>
```

먼저, HTML 구문을 보시면, Input 값을 Form 에서 Post 메서드로 처리하게 됩니다. Action 은 비어있으니, 이 페이지로 Post 되겠지요.

코드블록을 보시면 사용자의 입력 값을 Request 로 받은 후에 값을 더하는 처리가 이루어지고, 아래의 @total 에서 값을 처리하게 되는 구조 입니다. 이렇게 HTML 의 Form 으로 넘어온 POST 된 값을 Request 처리를 이용해 받아 처리할 수 있습니다. 아울러, 넘어온 값을 숫자형으로 변환하는 처리도 봐 두시면 좋을 듯 하네요.

Form 으로 넘어온 Post 값도 처리가 가능하며, URL 에 붙어 넘어오는 쿼리 스트링(URL query-string)을 받아 처리 역시 가능합니다. 이때도 마찬가지로, Request 로 값을 받아 처리 가능합니다.

예를 들어, 요청한 URL 이 이런 형식이라면

<http://localhost:8080/post.cshtml?param=hello> (테스트 URL 예제로 클릭해도 동작 안합니다)

```
var RequestQueryString = Request["param"];
```

그리고, 아래에서 넘겨 받은 값을 받아 처리 가능합니다.

```
<h2>Request 로 넘어온 문자열은 : @RequestQueryString 입니다.</h2>
```

이렇게 되겠네요.

```
@{
var total = 0;
var RequestQueryString = Request["param"];

//HTML 의 Form 에서 POST 될 경우에만 실행
if(IsPost) {
    // 사용자의 입력값을 받아서 저장
    var num1 = Request["text1"];
    var num2 = Request["text2"];
    // 넘겨받은 값을 Integer 형으로 변환 후 더하기 수행
    total = num1.AsInt() + num2.AsInt();
}
}
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
```

[transitional.dtd](#)">

<html xmlns="<http://www.w3.org/1999/xhtml>" xml:lang="en" lang="en">

<head>

<title>Razor 로 HTML Form 값을 처리</title>

<meta http-equiv="content-type" content="text/html; charset=utf-8" />

<style type="text/css">

body {

background-color: beige; font-family: Verdana, Arial; margin: 50px;

}

form {

padding: 10px; border-style: solid; width: 250px;

}

</style>

</head>

<body>

<p>두개의 숫자를 넣고 "더하기" 버튼을 클릭하세요~</p>

<p></p>

<form action="" method="post">

<p>

<label for="text1">숫자(1):</label>

<input type="text" name="text1" />

</p>

<p>

<label for="text2">숫자(2):</label>

<input type="text" name="text2" />

</p>

<p>

<input type="submit" value="더하기" />

</p>

</form>

@// 더한 값이 아래 표시 됩니다.

<h2>두 숫자의 합은: @total 입니다.</h2>

<h2>Request 로 넘어온 문자열은 : @RequestQueryString 입니다.</h2>

</body>

</html>

이렇게 간략히 Razor 가 제공하는 코드블록에 대한 정리와 함께 HTML 페이지의 Form 에서 넘어온 Post 값과 URL 쿼리 문자열로 넘어온 값을 처리하는 예제에 대해서 살펴 보았습니다. 어떠세요? Form 값에 대한 처리 쉽지요?

그럼, 다음 강좌에서는 "(5) Razor 강좌 - 재사용 가능한 코드 생성" 내용을 진행하도록 하겠습니다.

감사합니다.

참고자료 :

[WebMatrix Tutorial : 4 - Working with Forms](#)

(5) Razor 강좌 - 재사용 가능한 코드 생성

지난 시간에는 “(4) Razor 강좌 - 코드블록과 POST 처리” 강좌를 진행 했습니다.

이번 시간에는 바로 **이어서 다섯번째 강좌**, “(4) Razor 강좌 - 재사용 가능한 코드 생성” 강좌를 진행 하도록 하겠습니다.

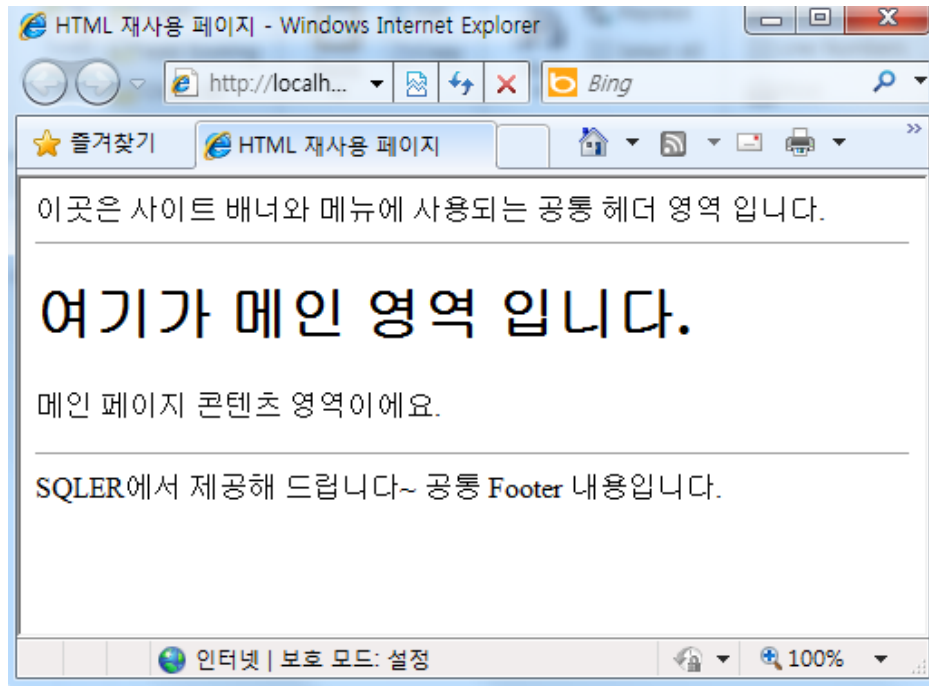
재사용 가능한 코드는 어쩌면 기존 웹 개발자 분들에겐 아주 짧은 내용이 될 것 같은데요.

바로, Include 구문과 같다고 보시면 됩니다.

이런 HTML 페이지가 있다고 생각해 보시지요.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-  
transitional.dtd">  
  
<html xmlns="http://www.w3.org/1999/xhtml">  
  <head>  
    <title>HTML 재사용 페이지</title>  
  </head>  
  <body>  
  
    <div class="header">  
      이곳은 사이트 배너와 메뉴에 사용되는 공통 헤더 영역 입니다.  
      <hr />  
    </div>  
  
    <h1>여기가 메인 영역 입니다.</h1>  
    <p>메인 페이지 콘텐츠 영역이에요.</p>  
  
    <div class="footer">  
      <hr />  
      SQLER 에서 제공해 드립니다~ 공통 Footer 내용입니다.  
    </div>  
  
  </body>  
</html>
```

아시죠? 걡 복사하시고, WebMatrix 에서 실행해 보세요~ 결과는 아래 처럼 보이실 겁니다.



넵. 우리네 웹사이트에서 일반적으로 사용되는 구조라고 가정해 보도록 하겠습니다.

그렇다면, 저 헤더 영역과 바닥 영역이 거의 모든 페이지에서 공통으로 사용되는데요. 재사용 가능한 저런 영역을 분리해 사용 하면 웹사이트 개발이 편리하지 않을까요? 넵, 웹개발 경험이 있는 분들은 바로 감이 오실 겁니다. include 처리죠.

Razor 에서 이런 공통 모듈 처리 - 재사용 가능한 웹페이지 영역 제작을 해 보도록 하겠습니다.

먼저 WebMatrix 에서 간단히 폴더를 만들어 볼게요. 간단히, 최상위의 WebSite1 폴더에서 마우스 우버튼 - "New Folder" 하시거나, 메뉴바의 New 버튼 아래를 클릭해 "New Folder"를 하셔도 됩니다. - 저는 Resue 라는 폴더를 하나 만들고 3 개의 파일을 만들었습니다.

index.cshtml : 메인영역의 콘텐츠 영역으로 사용

_header.cshtml : 공통 헤더 영역으로 사용

_footer.cshtml : 공통 바닥 영역으로 사용

할 예정입니다.

그리고, index.cshtml 파일을 아래처럼 구성합니다.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-  
transitional.dtd">
```

```

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>HTML 재사용 페이지</title>
  </head>
  <body>

    @RenderPage("_header.cshtml")

    <h1>여기가 메인 영역 입니다.</h1>
    <p>메인 페이지 콘텐츠 영역이에요.</p>

    @RenderPage("_footer.cshtml")

  </body>
</html>

```

보시면, 중간에 보이시지요? @RenderPage("_header.cshtml")

이렇게 @RenderPage 를 이용해 파일을 해당 영역으로 불러와 합칠 수 있습니다. - include 구문 처리라고 보시면 됩니다.

```

<div class="header">
  이곳은 사이트 배너와 메뉴에 사용되는 공통 헤더 영역 입니다.
  <hr />
</div>

```

_header.cshtml 페이지의 내용

```

<div class="footer">
  <hr />
  SQLER 에서 제공해 드립니다~ 공통 Footer 내용입니다.
</div>

```

_footer.cshtml 페이지의 내용

참고하세요 :

파일 이름 앞에 "_"를 넣는 이유는 다른 파일과의 손쉬운 식별을 위해 개발 보조 차원에서 구별해 넣은 것입니다. "_"를 반드시 사용할 필요는 없습니다.

자. 이렇게 3 개의 파일을 모두 저처럼 채우셨다면, 이제 실행을 해 보도록 하세요.

아~ 당연하겠지만, WebMatrix 에서 index.cshtml 페이지를 열고 실행(F12) 하셔야 합니다.

어떠세요? 분리된 파일이 하나의 페이지로 잘 보이시는지요?

넵~ 재사용 가능한 코드 생성! 많은 도움 되시길 바랍니다.

다음은 조금 더 향상된 코드 재사용 방안 - "레이아웃 페이지" 처리에 대해서 살펴 보도록 하겠습니다.

참고자료 :

[ASP.NET 사이트의 Razor Tutorial 내용 3 - Creating a Consistent Look](#)

(6) Razor 강좌 - 레이아웃 페이지 구조 처리

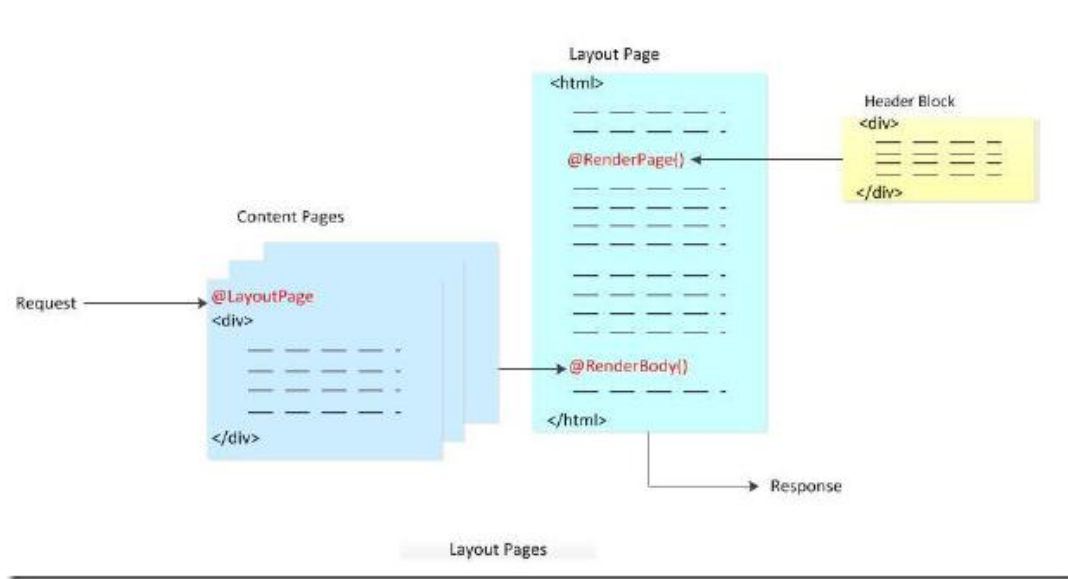
지난 시간에는 “(5) Razor 강좌 - 재사용 가능한 코드 생성” 강좌를 진행 했습니다.

이번 시간에는 바로 이어서 여섯번째 강좌, “Razor 강좌 - 레이아웃 페이지 구조 처리” 강좌를 진행 하도록 하겠습니다.

이 녀석도 실제 구조만 이해하시면 어렵지 않은 내용이에요. 꼬옥 아래 처음으로 보여 드리는 처리 구조를 잘 살펴 보시고 이해하시면 아래의 내용은 모두 껌처럼 보이실 겁니다.

왜 Layout 을 이용하는가?

이전 강좌에서는 RenderPage 를 이용해 페이지의 특정 위치에 원하는 내용을 삽입하는 처리를 진행해 보았습니다. 하지만, 개발자의 입장에서 볼 때, 재사용성을 극대화하고 좀더 구조적으로 페이지의 디자인과 레이아웃을 살릴 수 있는 과정이 필요하게 됩니다. 이 경우에 이용 가능한 처리가 바로 레이아웃 페이지(Layout page) 방식입니다. 먼저 아래 그림을 통해 처리 방식을 반드시 먼저 이해 하시는 게 아주 중요합니다.



처리 방식은 이렇습니다. 꼭 이해하셔야 아래 나머지 강좌 내용이 수월 하실 거예요.

- 1) 사용자가 Content Page 를 요청합니다.
- 2) Content Page 는 레이아웃 페이지의 링크를 담고 있습니다.
- 3) 레이아웃 페이지가 호출되며 레이아웃 페이지 내부에서 Content Page 가 렌더링 됩니다.
- 4) 레이아웃 페이지에서 다른 페이지 역시 호출해(include) 재사용 가능합니다.
- 5) 사용자에게 모든 렌더링된 레이아웃 페이지가 전달됩니다.

어떠세요? 조금 감이 잡히시는지요?

레이아웃 페이지는 마치 HTML 파일처럼 콘텐츠 정보를 제외한 모든 처리를 넣을 수 있습니다.

HTML 페이지와 다른점은 오직 콘텐츠 페이지의 정보를 담기 위한 `RenderBody` 나 `RenderPage` 가 호출된다는 점 뿐이지요.

몇 가지 더 경험적인 Tip 을 말씀 드리자면, 앞의 강좌의 재사용을 위한 `RenderPage` 방식보다 이런 레이아웃 방식을 대규모의, 디자인과 협업이 더 많이 필요할 경우에 주로 사용하게 됩니다. 정보와 디자인을 쉽게 분리해 개발자와 디자이너간 협업을 증대시키며 구조적으로 정보를 개발자의 조건에 따라 레이아웃 페이지에 삽입 여부 등도 제어가 가능하기 때문에 더욱 동적이고 개발자와 디자이너 협업이 가능한 처리를 이용 가능하기 때문입니다.

자, 이제 처리 방식이 조금 이해 되셨다면, 저와 같이 코드를 직접 실행해 보시지요.

앞의 강좌에서 “Reuse”라는 폴더를 만들어 두었습니다. 아래처럼 파일들을 이 폴더에 생성해 볼게요.

```
<!DOCTYPE html>

<head>
  <title> 레이아웃 콘텐츠를 만들어요 </title>
  <link href="@Href("Styles/Site.css")" rel="stylesheet" type="text/css" />
</head>
<body>
  @RenderPage("_Header2.cshtml")
  <div id="main">
    @RenderBody()
  </div>
  <div id="footer">
    &copy; 2010 SQLER 의 Razor 와 WebMatrix 강좌.
  </div>
</body>
</html>
```

“_layout1.cshtml” 파일로 저장합니다.

```
@{
  LayoutPage = "_Layout1.cshtml";
}

<h1> 구조화된 콘텐츠 </h1>
<p>

택시기사가 나를 보통 녀석이 아닌데?라고 생각하게 만드는 방법 (1/3)

1 왼쪽 문을 열고 타자마자 오른쪽 문을 열고 내린다.
```

4 요금이 오를 때마다 비명을 지른다.

7 어느새 내가 운전한다.

18 U턴 해서 반대 차선에 내려 주세요.

21 택시에 타자마자 페브리즈를 마구 뿌린다.

28 재개발 지구나 새 건물이 들어 선 거리를 지날 때 한숨을 쉬며 「하나도 안 변했네... 여기」라고 중얼거린다.

번역 : 행복한 마조히스트(sweetpjy.tistory.com)

<p>

“Content1.cshtml” 파일로 저장합니다.

<div id="header">

구조화된 콘텐츠를 만들어 보세요~

</div>

“_header2.cshtml” 파일로 저장합니다.

Reuse 폴더 하위에 “Styles” 폴더를 만들어 주세요.

```
h1 {
    border-bottom: 3px solid #cc9900;
    font: 2.75em/1.75em Georgia, serif;
    color: #996600;
}

ul {
    list-style-type: none;
}

body {
    margin: 0;
    padding: 1em;
    background-color: #ffffff;
    font: 75%/1.75em "Trebuchet MS", Verdana, sans-serif;
    color: #006600;
```

```

}

#list {
    margin: 1em 0 7em -3em;
    padding: 1em 0 0 0;
    background-color: #ffffff;
    color: #996600;
    width: 25%;
    float: left;
}

#header, #footer {
    margin: 0;
    padding: 0;
    color: #996600;
}

```

이 파일은 Styles 폴더 하위에 “Site.css” 파일로 만들어 주세요.

이제 실행을 해 볼까요? WebMatrix 에서 Content1.cshtml 을 선택하시고 F12 를 눌러 실행해 주세요.

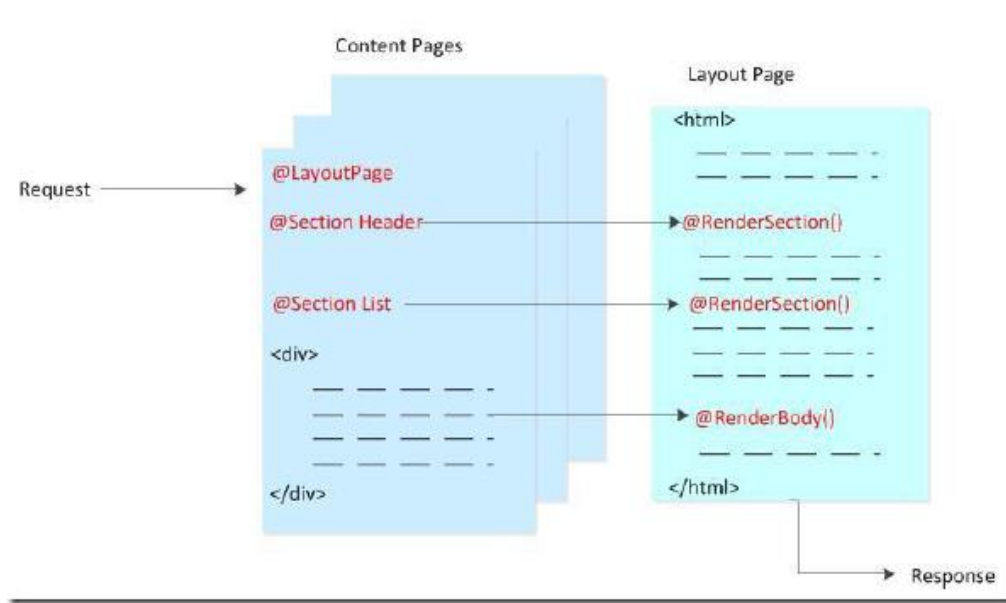
조금 파일들이 많으시지요? 걱정 마세요. 이해만 하시면 아주 쉽습니다.

이 강좌 맨 위의 처리 흐름을 생각 하시면서 설명 드릴게요.

- 1) 사용자의 요청으로 Content1.cshtml 파일이 실행되면 Layout1.cshtml 페이지를 호출합니다.
- 2) Layout 페이지에서 Header 영역을 불러와 include 시키고 바로 아래에서 Content 페이지의 내용을 RenderBody 로 호출합니다.
- 3) 만들어진 콘텐츠가 담긴 레이아웃 페이지가 사용자에게 CSS 디자인 파일과 함께 전달됩니다.

콘텐츠 페이지의 특정 영역을 레이아웃 페이지 특정 영역에 표시

쿨럭. 쓰고 나니 제목이 더 복잡하네요. 의도한 바는 아닙니다만, 아래 도표를 보시면 바로 감이 오실 겁니다. 먼저, 아래 도표를 보시지요.



- 1) 사용자의 요청이 콘텐츠 페이지에 도착하면 레이아웃 페이지가 호출됩니다.
- 2) 레이아웃 페이지의 RenderSection 은 콘텐츠 페이지의 지정된 Section 을 썩썩 가져와 표시합니다.
- 3) RenderBody 와 혼용도 가능합니다.
- 4) 사용자에게 콘텐츠가 임혀진 레이아웃 페이지가 전달됩니다.

자 그럼 마찬가지로, 예제를 통해 이해해 보도록 하겠습니다.

```
<!DOCTYPE html>
<html>
  <head>
    <title>섹션이 존재하는 콘텐츠 파일</title>
    <link href="@Href("Styles/Site.css")" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <div id="header">
      @RenderSection("header")
    </div>
    <div id="list">
      @RenderSection("list")
    </div>
    <div id="main">
      @RenderBody()
    </div>
  </body>
</html>
```

```
<div id="footer">
    &copy; 2010 SQLER 의 Razor 와 WebMatrix 강좌
</div>
</body>
</html>
```

”_Layout2.cshtml” 파일로 저장합니다.

```
@{
    LayoutPage = "_Layout2.cshtml";
}

@section list {
    <ul>
        <li>김대우</li>
        <li>박종석</li>
        <li>황리건</li>
        <li>조성우</li>
        <li>김영욱</li>
        <li>김재우</li>
        <li>서진호</li>
    </ul>
}

@section header {
    <div id="header">
        레이아웃 페이지 제작
    </div>
}

<h1>여러개의 섹션이 있는 콘텐츠</h1>
<p>
```

택시기사가 나를 보통 녀석이 아닌데?라고 생각하게 만드는 방법 (2/3)

39 내릴 때 레드 카펫을 깔다.

45 비 내리는 밤에 택시를 타고 「저기, 요즘에 이 근처에서 뭐가 나온다는 소문이 돌잖아요~」

49 「오늘은 여기에 한 번 뒤 볼까……」 라고 말하며 계기판 위에 물이 든 종이컵을 내려놓는다.

59 행선지를 물어보면 「몰라요… ! !」 하고 운다.

```
70 들고 있던 비상등을 차 위에 올리고 「빨리 갑시다, 신호는 무시하세요!」
95 차멀미가 심하니까 잘 부탁드립니다. 이러면 운전을 정말 조심스럽게 해준다.
99 타자마자 「교통 불편 신고 엮서」를 집어 든다.
104 땀을 뻘뻘 흘리면서 향문을 손으로 틀어막고 조심조심 탄다.
번역 : 행복한 마조히스트(sweetpjy.tistory.com)
</p>
```

”Content2.cshtml” 파일로 저장합니다.

보시는 것처럼, 레이아웃 페이지에서 RenderSection 을 통해 영역을 구별해 두게 됩니다.

이어서 콘텐츠 페이지에서 @section 을 이용해 콘텐츠를 분리해 처리하게 되지요.

그럼, WebMatrix 에서 Content2.cshtml 파일을 선택하시고 F12 를 눌러 실행해 보세요.

레이아웃 페이지는 구조와 영역에 대한 처리가 사실상 전부입니다. 잊지 마시고, 꼭 페이지가 어떻게 렌더링 되어 사용자에게 표시되는지, 그 흐름을 고민하면서 진행해 보시길 바랍니다.

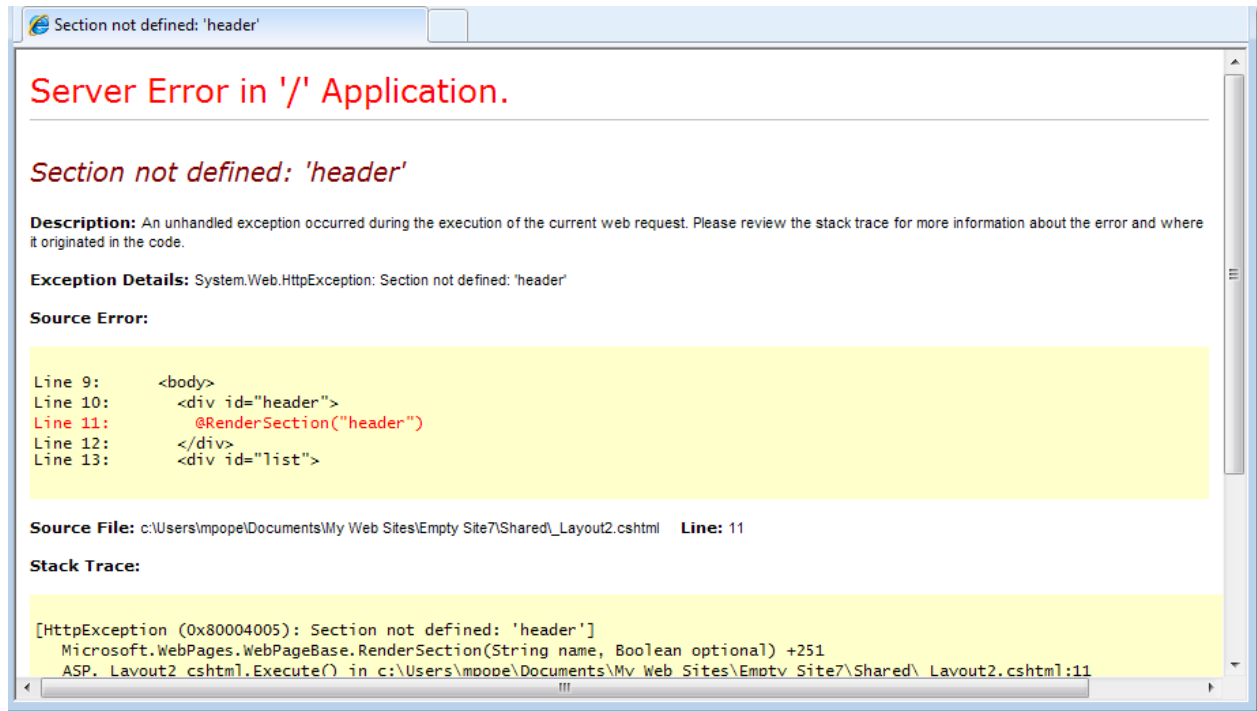
Section 이 존재하지 않을 경우 처리

위의 콘텐츠 페이지에 각 Section 정보를 담고 있습니다. 레이아웃 페이지에는 이 Section 을 표시하라고 명시되어 있는데, 몇몇 콘텐츠 페이지만 이런 Section 이 빠져 있다면 어떻게 될까요? – Section 여부에 따른 분기 처리가 가능합니다. 아래 내용을 참고 하세요.

Content2.cshtml 페이지에서 아래 섹션을 제거해 보세요.

```
@section header {
    <div id="header">
        레이아웃 페이지 제작
    </div>
}
```

네, 바로 header 섹션을 없애 버리는 겁니다. 저장 하시고, Content2.cshtml 파일을 WebMatrix 에서 실행해 보시면. 똥시궁~ 에러가 떨어집니다.



이런 경우를 위해 간단히 Layout 페이지를 살짝 손보면 됩니다.

_Layout2.cshtml 파일을 열고

```
@RenderSection("header")
```

항목을 아래로 변경합니다.

```
@RenderSection("header", optional: true)
```

또는, 아래처럼 if 분기로도 처리가 가능합니다.

```
@if (IsSectionDefined("header")) {
    @RenderSection("header")
}
```

Section 이 없더라도 걱정하실 필요 없지요? ^_^

끝으로, 데이터를 레이아웃 페이지로 넘기는 방법에 대해서 말씀 드리도록 하겠습니다.

콘텐츠 페이지의 데이터를 레이아웃 페이지로 전달

콘텐츠 페이지에서 사용자의 로그인 상태를 레이아웃 페이지에 전달해 그에 맞게 페이지를 렌더링 하고자 할 경우에 반드시 콘텐츠 페이지에서 데이터를 레이아웃 페이지에 전달해 적절한 처리가 이루어지도록 구성할 필요가 있습니다. 이런 경우를

생각해 보시면 좋을 거예요. 어렵지 않습니다. 이럴 경우 값을 전달할 경우에는 Request 처리와 비슷하게 “PageData” 속성을 이용하시면 됩니다.

예제를 돌려 보도록 할게요.

흐름은 간단히, PageData 조건을 검사해 메뉴 아이템들을 표시할지 말지 여부를 결정하는 간단한 예제라고 보시면 됩니다.

```
@{
    LayoutPage = "_Layout3.cshtml";
    PageData["Title"] = "데이터 전달";
    PageData["ShowList"] = true;
    if (IsPost) {
        if (Request["list"] == "off") {
            PageData["ShowList"] = false;
        }
    }
}
```

```
@section header {
    <div id="header">
        레이아웃 페이지 만들기
    </div>
}
<h1>@PageData["Title"]</h1>
<p>
```

택시기사가 나를 보통 녀석이 아닌데?라고 생각하게 만드는 방법 (3/3)

106 룸미러 너머로 운전기사의 얼굴을 심각하게 응시.

135 혼자 탄 다음에 아무도 없는 조수석 쪽에 계속 말을 건다.

144 조수석 앞에 붙은 기사증을 소리 내서 읽는다.

157 경찰차가 지나가면 몸을 숙인다.

161 준비한 애니메이션 CD 를 카오디오에 넣고 튼다.

172 뒷자석으로 타서 조수석으로 옮겨 앉는다.

223 타자마자 「트렁크가 열려있는데요? 」 한다. 운전기사가 확인하러 간 사이에 운전석에 앉는다. 그대로 출발.

번역 : 행복한 마조히스트(sweetpjy.tistory.com)

</p>

```
@if (PageData["ShowList"] == true) {
```

```

<form method="post" action="">
    <input type="hidden" name="list" value="off" />
    <input type="submit" value="Hide List" />
</form>
}
else {
    <form method="post" action="">
        <input type="hidden" name="list" value="on" />
        <input type="submit" value="Show List" />
    </form>
}
<br />

```

Content3.cshtml 파일로 저장합니다.

```

<!DOCTYPE html>
<html>
    <head>
        <title>@PageData["Title"]</title>
        <link href="@Href("Styles/Site.css")" rel="stylesheet" type="text/css" />
    </head>
    <body>
        <div id="header">
            @RenderSection("header")
        </div>
        @if (PageData["ShowList"] == true) {
            <div id="list">
                @RenderPage("/Reuse/_List.cshtml")
            </div>
        }
        <div id="main">
            @RenderBody()
        </div>
        <div id="footer">
            &copy; 2010 SQLER 의 Razor 와 WebMatrix 강좌
        </div>
    </body>
</html>

```

```
</div>
</body>
</html>
```

”_Layout3.cshtml” 파일로 저장합니다.

```
<ul>
    <li>김대우</li>
    <li>박중석</li>
    <li>황리건</li>
    <li>조성우</li>
    <li>김영욱</li>
    <li>김재우</li>
    <li>서진호</li>
</ul>
```

“_List.cshtml” 파일로 저장합니다.

흐름은 간단히, PageData 조건을 검사해 메뉴 아이템들을 표시할지 말지 여부를 결정하는 예제라고 보시면 됩니다.

완료 하셨다면, Content3.cshtml 파일을 실행해 보세요. 아래의 “Hide List” 버튼을 클릭하시면 Post 여부에 따라 넘겨받은 PageData 로 레이아웃이 변화합니다.

어떠세요? 어렵지 않으시지요?

자~ 이렇게 해서 지난 시간의 RenderPage 에 이어 레이아웃을 이용한 처리 역시 알아 보았습니다.

사실, 맨 처음에 말씀 드린 것처럼, 콘텐츠 페이지와 레이아웃 페이지가 어떻게 서로 지지고 묶으면서 영역들을 렌더링 하는지 여부만 이해하시면 조금은 쉬운 내용셨을 거라고 생각되네요.

레이아웃 처리를 이용할 경우는...

- 1) 개발자의 입장에서 레이아웃의 재사용성을 극대화하고
- 2) 디자이너와 협업을 통해 구조적으로 페이지의 디자인과 레이아웃을 살릴 수 있는 과정이며
- 3) 레이아웃과 콘텐츠 페이지간 데이터 전달 및 상호작용이 가능한 처리 방식입니다.

자~ 그럼 레이아웃 강좌를 마무리 하도록 하겠습니다.

다음 강좌인 “(7) Razor 강좌 - 파일처리”에서 뵙도록 하겠습니다. ^_^

감사합니다.~

PS.

cshmtl 파일 앞에 “_”가 붙은 녀석은 사용자의 요청에 의해 실행 불가한 파일입니다. 다른 콘텐츠 페이지 같은 “_”가 없는 녀석들에 의해 참조되어 호출만 가능한 녀석이지요. 레이아웃 페이지나 항상 참조 되는 header, footer 페이지에 쓰면 유용하겠지요? 참고 하시길 바랍니다.

참고자료 :

[ASP.NET 사이트의 Razor Tutorial 내용 3 – Creating a Consistent Look](#)

(7) Razor 강좌 - 파일처리, 파일 업로드

지난 시간에는 “(6) Razor 강좌 - 레이아웃 페이지 구조 처리”강좌를 진행 했습니다.

이번 시간에는 바로 **이어서 일곱번째 강좌, “(7) Razor 강좌 - 파일처리”**강좌를 진행 하도록 하겠습니다.

이번 내용도 쉬워요. 아마 현업에서 개발하시는 분들은 느끼시겠지만, 소규모의 사이트를 제작할 경우에도 파일 작업 보다는 데이터베이스 작업을 선호하는 분들이 많으실 거예요.

하지만, 파일 업로드와 같은 처리는 자주 필요하고, 간단한 업로드된 파일들에 대한 처리 작업은 파일 처리 과정에서 종종 쓰이실 경우가 있으니 모쪼록 많은 도움 되시길 바랍니다.

이 강좌는 파일처리 작업인 “파일 쓰기”, “파일 읽기”, “파일 수정”, “파일 삭제”, “파일 업로드(여러 파일 업로드)” 과정으로 진행됩니다.

파일쓰기 작업

Razor에서는 이러한 파일 처리 작업이 아주 쉬워요. 바로 코드를 통해 설명 드리도록 하겠습니다.

```
@{
    string result = "";
    if (IsPost)
    {
        // 사용자 입력 값을 변수에 저장
        string firstName = Server.HtmlEncode(Request["FirstName"]);
        string lastName = Server.HtmlEncode(Request["LastName"]);
        string email = Server.HtmlEncode(Request["Email"]);
        // 문자열 연결
        string userData = firstName + "," + lastName +
            "," + email + Environment.NewLine;
        // 경로와 파일 이름 설정
        string dataFile = Request.PhysicalApplicationPath + "data.txt";
        // 파일에 쓰기
        try
        {
            File.WriteAllText (@dataFile, userData);
            result = "저장되었습니다.";
        }
    }
}
```

```

        catch (Exception e)
        {
            result = "쓰기 도중에 오류가 발생 했습니다.";
        }
    }
}

```

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

```

```

<html xmlns="http://www.w3.org/1999/xhtml">

```

```

    <head>

```

```

        <title>파일에 쓰기 작업</title>

```

```

    </head>

```

```

    <body>

```

```

        <form id="form 1" method="post">

```

```

            <div>

```

```

                <table>

```

```

                    <tr>

```

```

                        <td>이름 :</td>

```

```

                        <td><input id="FirstName" name="FirstName"

```

```

type="text"/></td>

```

```

                    </tr>

```

```

                    <tr>

```

```

                        <td>성 :</td>

```

```

                        <td><input id="LastName" name="LastName" type="text" /></td>

```

```

                    </tr>

```

```

                    <tr>

```

```

                        <td>Email:</td>

```

```

                        <td><input id="Email" name="Email" type="text" /></td>

```

```

                    </tr>

```

```

                    <tr>

```

```

                        <td></td>

```

```

                        <td><input type="submit" value="Submit"/></td>

```

```

                    </tr>

```

```

                </table>

```

```

            </div>

```

```

        </div>

```

```

        @if(result != "")
        {
            <p>결과 : @result</p>
        }
    </div>
</form>
</body>
</html>

```

File 이라는 폴더를 만들고 Write.cshtml 파일로 저장한 후에 실행합니다.

단순히, 사용자에게 입력 값을 받고, POST 처리가 발생하면 값을 변수에 저장해 파일에 쓰는 예제 입니다. File 개체를 이용한다는 것 정도만 참고 하시면 될 것 같습니다.

파일이 만들어지는 위치는

```
Request.PhysicalApplicationPath + "data.txt";
```

처리에 의해서 결정되는데요, 보통 문서 폴더 하위의 “My Web Sites\Website1” 폴더가 기본 경로이실 거예요. 여기 하위에 data.txt 파일이 위치하고 있을 겁니다.

다음으로 파일 이어쓰기 – Append 작업을 해 보도록 할게요.

파일 이어쓰기 – Append 작업

```

@{
    string result = "";
    if (IsPost)
    {
        // 사용자 입력 값을 변수에 저장
        string firstName = Server.HtmlEncode(Request["FirstName"]);
        string lastName = Server.HtmlEncode(Request["LastName"]);
        string email = Server.HtmlEncode(Request["Email"]);

        // 문자열 연결
        string userData = firstName + "," + lastName +
            "," + email + Environment.NewLine;
    }
}

```

```

// 경로와 파일 이름 설정
string dataFile = Request.PhysicalApplicationPath + "data.txt";

// 파일에 Append
try
{
    File.AppendAllText (@dataFile, userData);
    result = "저장되었습니다.";
}
catch (Exception e)
{
    result = "오류가 발생 했습니다.";
}
}
}

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>이어쓰기 - Append 처리</title>
    </head>
    <body>
        <form id="form1" method="post">
            <div>
                <table>
                    <tr>
                        <td>이름:</td>
                        <td><input id="FirstName" name="FirstName" type="text"
/></td>

                    </tr>
                    <tr>
                        <td>성:</td>
                        <td><input id="LastName" name="LastName" type="text" /></td>

                    </tr>
                </table>
            </div>
        </form>
    </body>
</html>

```

```

        <td>Email:</td>
        <td><input id="Email" name="Email" type="text" /></td>
    </tr>
    <tr>
        <td></td>
        <td><input type="submit" value="Submit"/></td>
    </tr>
</table>
</div>
<div>
    @if(result != "")
    {
        <p>결과: @result</p>
    }
</div>
</form>
</body>
</html>

```

Append.cshtml 파일로 저장하고 실행합니다.

마찬가지로 Write 와 같지요? AppendAllText 를 이용해 기존 파일의 뒤에 이어쓰기를 진행합니다.

다음으로 파일 읽기를 진행해 보도록 할게요.

파일 읽기 작업

```

@{
    string result = "";
    Array userData = null;
    char[] delimiterChar = {'\n'};
    string dataFile = Request.PhysicalApplicationPath + "data.txt";
    try
    {
        if (File.Exists(dataFile))
        {

```

```

        userData = File.ReadAllLines(dataFile);
        if (userData == null)
        {
            // 빈 파일 오류.
            result = "파일이 비어 있습니다.";
        }
    }
    else
    {
        // 파일 존재 안 함
        result = "파일이 존재하지 않습니다.";
    }
}
}
catch (Exception e)
{
    result = e.Message.ToString();
}
}

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "<http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd>">

<html xmlns="<http://www.w3.org/1999/xhtml>">

<head>

<title>파일 읽기</title>

</head>

<body>

<div>

<h1>파일 읽기 처리 </h1>

@dataFile

@result

@if (result == "")

{

@foreach (string dataLine in userData)

{

User


```

                                @foreach (string dataltem in dataLine.Split(delimiterChar))
                                {
                                    <li>@dataltem</li>
                                }
                            </ul>
                        </li>
                    }
                </ol>
            }
        </div>
    </body>
</html>

```

Read.cshtml 파일로 저장합니다.

조금씩 난이도가 올라가나요? 간단히, 저희가 앞에서 저장한 파일을 읽어 옵니다. userData 라는 배열을 만들고 여기에 값을 넣은 후에 dataLine 문자열에서 지정한 구분자(delimiter “,” – 콤마)로 분리해 값을 출력하는 예제 입니다.

말이 길었나요? ^_;;; **주의 깊게 보실 부분은 File.ReadAllLines** 부분입니다.

다음 과정으로 파일 삭제 과정을 진행해 보도록 하겠습니다.

파일 삭제 처리

```

@{
    bool deleteSuccess = false;
    var photoName = "";
    if (IsPost)
    {
        photoName = Request["photoFileName"] + ".jpg";
        var fullPath = Path.Combine(Request.PhysicalApplicationPath + @"images\W", photoName);
        if (File.Exists(fullPath))
        {
            File.Delete(fullPath);
            deleteSuccess = true;
        }
    }
}

```

```

    }
}
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>사진 파일 삭제</title>
  </head>
  <body>
    <h1>사이트에서 파일을 삭제합니다.</h1>
    <form name="deletePhoto" action="" method="post">
      <p>지울 사진 파일의 이름을 .jpg 확장자를 빼고 적으세요. :</p>
      <input name="photoFileName" type="text" value="" />
      <p>
        <input type="submit" value="Submit">
      </p>
    </form>
    @if(deleteSuccess)
    {
      <p> @photoName 파일이 삭제 되었습니다.! </p>
    }
  </body>
</html>

```

Del.cshtml 파일로 저장합니다.

웹사이트 루트 폴더 (아마도 문서 하위에 “My Web Sites\WebSite1\” 기본 폴더일겁니다.) 하위에 images 라는 폴더를 만들어 주세요. 아무 jpg 파일을 만들거나 복사합니다.(확장자가 꼭 jpg 이어야 합니다.)

실행하시고 이 images 폴더에 확장자 .jpg 를 빼고 이름만 적으면 됩니다.

마찬가지로, “File.Delete” 을 주의해서 보시길 바랍니다. ^_^ 쉽지요~ 넵~ 쉽습니다. ^_^

다음으로 파일 업로드 과정에 대해서 알아 보도록 하겠습니다.

파일 업로드 (여러 파일 업로드)

```
@{
```

```

var fileName = "";
if (IsPost) {
    var fileSavePath = "";
    var uploadedFile = Request.Files[0];
    fileName = Path.GetFileName(uploadedFile.FileName);
    fileSavePath = Server.MapPath("~/App_Data/UploadedFiles/" +
        fileName);
    uploadedFile.SaveAs(fileSavePath);
}
}

<!DOCTYPE html>
<html>
    <head>
        <title>파일 업로드 예제</title>
    </head>
    <body>
        <h1>파일을 업로드 하는 예제 입니다.</h1>
        @FileUpload.GetHtml(
            initialNumberOfFiles:1,
            allowMoreFilesToBeAdded:false,
            includeFormTag:true,
            uploadText:"업로드")
        @if (IsPost) {
            <span>파일 업로드가 완료 되었습니다!</span><br/>
        }
    </body>
</html>

```

FileUpload.cshtml 파일로 저장하고, 웹사이트 루트에 위치하고 있는 App_Data 폴더 하위에 UploadedFiles 폴더를 만들어 두세요. 그리고 실행해 보시면 파일 업로드 처리가 되는 것을 확인 가능합니다.

기존에 웹 개발을 해 오신 분들은 아마 느끼실지 모르겠습니다. 업로드 처리가 뭔가 더 쉽고 한 줄의 코드로 처리 되지요? 바로 Upload Helper 입니다.

Helper 에 대한 간략한 소개

Helper 는 Razor 의 가장 큰 혁신 기능 중의 하나로 복잡한 코드 블록이나 기능을 모듈화 시키는 기능입니다. 보시면 Upload 기능도 속성값들만 적는 것으로 우리가 익히 아는 업로드 폼이 자동 완성되지요.(뒤의 강좌에서 아주 상세히 풀어

보겠습니다.)

Helper 를 통해 업로드를 하게 되면 Request.Files 개체가 이 요청을 받게 됩니다.

```
var uploadedFile = Request.Files[0];
```

이 부분이지요. 뒤의 오디널(Ordinal) 번호는 첫 번째 업로드 된 파일 번호이며 0 번부터 시작하게 됩니다. 여러 개의 파일을 받게 되면 후속 번호는 0 다음으로 이어서 1, 2, 3 이런 식으로 받아서 처리 가능해 지지요.(여러 개의 파일 업로드 - Multiple file upload 역시 쉽게 가능합니다.) 이어서 GetFileName 으로 파일 이름만 저장하고 MapPath 로 서버의 /App_Data/UploadedFiles 폴더 하위에 파일이 저장되는 것이지요.

끝으로 여러 개의 파일 업로드는 예제 정도만 소개해 드리도록 하겠습니다.

여러 파일 업로드(Multiple file upload)

```
@{
    var message = "";
    if (IsPost) {
        var fileName = "";
        var fileSavePath = "";
        int numFiles = Request.Files.Count;
        message = "파일 업로드 완료. 전체 업로드 파일 수 : " +
            numFiles.ToString();
        for(int i =0; i < numFiles; i++) {
            var uploadedFile = Request.Files[i];
            fileName = Path.GetFileName(uploadedFile.FileName);
            fileSavePath = Server.MapPath("~/App_Data/UploadedFiles/" +
                fileName);
            uploadedFile.SaveAs(fileSavePath);
        }
    }
}

<!DOCTYPE html>
<html>
    <head><title>여러 파일 업로드</title></head>
<body>
    <form id="myForm" method="post"
        enctype="multipart/form-data"
```

```

        action="">
<div>
<h1>여러 파일 업로드 예제</h1>
@if (!IsPost) {
    @FileUpload.GetHtml(
        initialNumberOfFiles:2,
        allowMoreFilesToBeAdded:true,
        includeFormTag:true,
        addText:"업로드 파일 추가",
        uploadText:"업로드")
}
<span>@message</span>
</div>
</form>
</body>
</html>

```

FileUploadMultiple.cshtml 파일로 저장하고 실행해 보세요. Helper 의 설정대로 2 개의 기본 업로드 파일이 가능하며, “업로드 파일 추가”를 클릭해 업로드 할 파일 수를 동적으로 늘릴 수 있습니다. 위의 Post 처리에서는 파일의 수 만큼 루프를 돌면서 처리하는 것을 확인 가능하지요.

이렇게 해서 간단히 파일 작업에 대해서 알아 보았습니다.

파일에 대한 기본 작업과 업로드 작업 모두 어렵지 않으셨을 거예요. 특히, Helper 의 경우 Razor 의 장점을 아주 잘 보여주는 녀석이고, 업로드 Helper 만 봐도 참 쉽게 잘 만들었다고 느껴지실 거예요. Razor 의 특징이 이런 쉬운 개발이 아닐까 생각합니다.

다음 강좌는 많은 분들이 기다리고 계시지요? (8) *Razor 강좌 - 데이터베이스 처리* 강좌에서 인사 드리도록 하겠습니다. 도움 되시길 바랍니다.

(8) Razor 강좌 - 데이터베이스 처리

지난 시간에는 “(7) Razor 강좌 - 파일처리, 파일 업로드”강좌를 진행 했습니다.

이번 시간에는 바로 이어서, “(8) Razor 강좌 - 데이터베이스 처리”강좌를 진행 하도록 하겠습니다.

이번 시간은 데이터베이스 관련 내용이에요. 아마 이 Razor & WebMatrix 관련 내용을 기다리시는 분들 중에 이 데이터베이스 부분이 가장 궁금한 부분일 것 같네요.

개인적으로, 가장 먼저 제가 테스트 해보고 공부한 부분이 바로 이 데이터베이스 처리 부분인데요.

기존 웹 개발 언어들의 스파게티 코드를 얼마나 분리해냈고, DB 처리가 간결하게 구성되어 있는지 비교하면서 보셔도 좋을 것 같습니다.

SQL 과 같은 데이터베이스를 처음 하시는 분들의 참고자료 :

음... 부족한 제 강좌 글을 보시는 분들 중에는 데이터베이스가 어떤 것인지 잘 감이 안 잡히는 분도 계실 것 같은데요, 제가 예전에 데이터베이스 기본 개념을 설명한 강좌 몇개를 소개해 드릴 예정이니 처음 데이터베이스를 공부하시는 분들도 참고 하시면 많은 도움 되실 거예요. - 시간이 조금 지난 강좌라도 걱정 마세요. 데이터베이스 기본 개념은 여전 하답니다. 한번이라도 =, MSSQL 이나 MySQL 이나 오라클 등을 아주 잠시라도 접해보신 분들은 아마 이하 내용을 쉽게 이해 하실것 같아요. 하지만, 전혀 DB 에 대해 경험이 없는 분이라면 아래 내용을 참고 하셔도 좋을 것 같습니다.

데이터베이스를 처음 하시는 분들이 참고하시면 좋을 내용

[데이터베이스란 무엇이고 왜 사용하는가?](#)

[SQL 프런티어가 전해 드리는 SQL2008 기본 강좌](#) - 최근에 진행된 SQLER 의 SQL 프런티어 강좌 입니다.

[SQL2000 강좌 게시판](#) - 앞 부분의 SELECT, INSERT, UPDATE, DELETE 를 보시면서 감을 잡으시는 것도 좋을 것 같습니다.

웹 개발과 데이터베이스 처리는 뗄래야 뗄 수 없는 그런 찰떡 같은 관계입니다. 이번의 요즘 좀 한다는 분들이 관심좀 가지시고 계신 Razor 가 데이터베이스 관련 개발을 공부할 좋은 기회라고 생각하시고 꼭 DB 와 SQL 처리에 대한 감을 잡으시는 것도 좋을 것 같아요.

PHP 나 ASP 를 사용 하셨고, 예전에 사용해 보신 DB 가 MSSQL 이나 MySQL 또는 이기종의 DB 라서 걱정되세요?

전혀 걱정 안하셔도 됩니다. 소견으로 감히 말씀 드리면, PHP 보다 쉽고 다양한 편의성을 제공하며, ASP.NET 의 엔터프라이즈급 성능과 안정성을 제공하는게 Razor 입니다. 당연히, WebMatrix 에 포함된 DB 도 SQL 서버처럼 ANSI 표준을 따르고, 아무 DB 나 경험이 있으시다면 쿼리 방식에는 차이가 없기 때문에 어려움 없이 개발을 진행 가능하답니다~ 이 강좌가 끝날 때 즈음이면 저 코난이처럼 “후덜~ Razor DB 처리 이거 겁나 쉽네”를 연발 하시게 될거예요~!!

Razor 와 WebMatrix 에서의 데이터베이스의 관계?

예전 강좌에서 소개해 드렸는데요. WebMatrix 에서는 간단히 SQL 서버보다 더 작고 가벼운 SQL Server Compact Edition

4 가 WebMatrix 에 포함되어 있습니다. 전문 웹 개발자 분들을 위해 조금 더 상세한 설명을 드리자면, 무료 데이터베이스 엔진으로 .NET 기반의 API 를 제공해 WebMatrix 를 통한 손쉬운 웹 개발이 가능하며, SQL Server 로 쉬운 데이터 마이그레이션이 가능합니다.(WebMatrix 에서 자체 마이그레이션 도구를 제공합니다.)

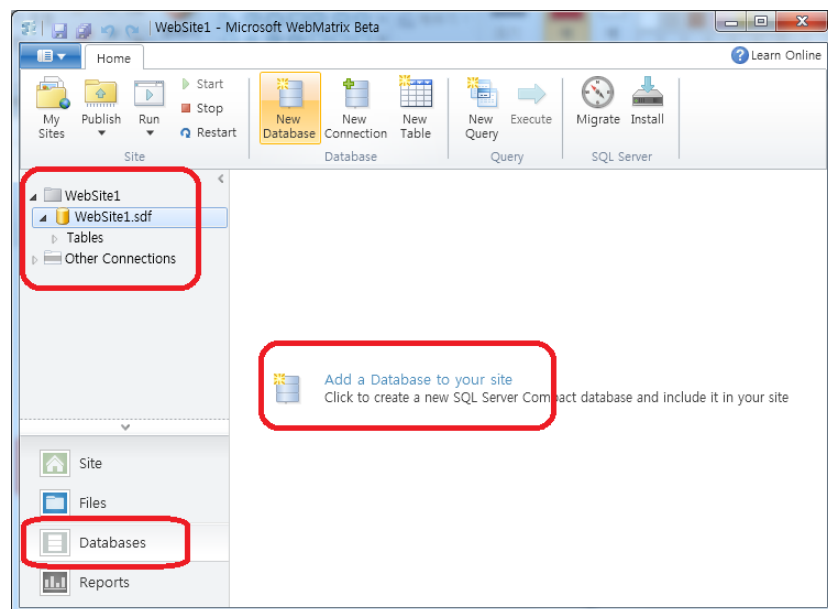
처음 데이터베이스 관련 개발을 Razor 와 WebMatrix 로 진행하셔도 아무 걱정하지 마세요. 데이터베이스 쿼리나 테이블 생성 디자인 등도 모두 WebMatrix 가 책임집니다~ 개발, DB, 배포 모든 과정을 하나의 심플하고 빠른 개발 도구, WebMatrix 에서 진행 하실 수 있어요~
우워~ 세상 점점 좋아 지는구나~ 생유 생유~

WebMatrix 의 데이터베이스 기능 참고자료 :

[\[동영상 강좌\] \(2\) WebMatrix 5 분 리뷰~](#) - 동영상으로 Database 는 물론 WebMatrix 에 대한 전반적인 기능을 보실 수 있습니다.

WebMatrix 에서 데이터베이스 작업, 테이블 작업 뿐만 아니라 쿼리도 가능하다고 계속 말씀 드렸습니다. 실제로 이번 강좌를 진행할때 다 해보도록 할게요.

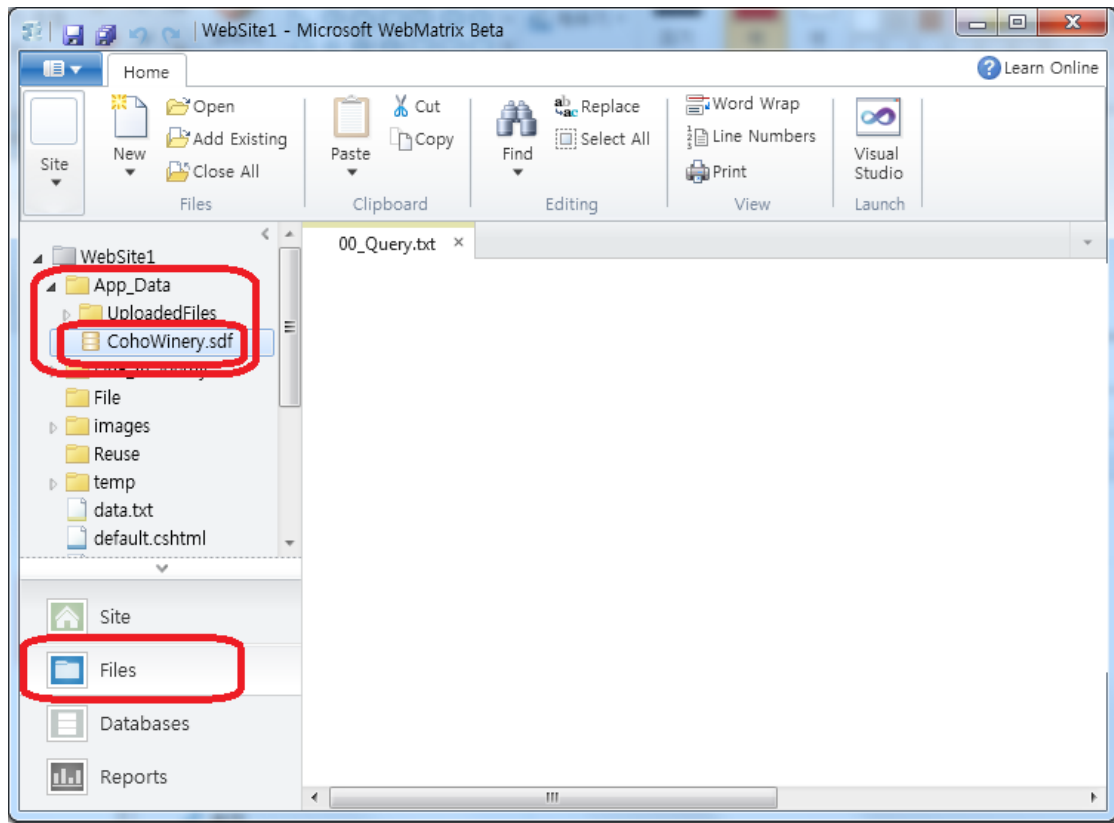
WebMatrix 에서 데이터베이스 생성



- (1) 왼쪽 아래의 Databases 를 클릭해 데이터베이스 영역으로 넘어 갑니다.
- (2) 중간에 Add a Database to your site 를 클릭해 데이터베이스를 생성하세요.
- (3) 위의 웹사이트 하위에 WebSite1.sdf 라는 DB 가 잘 만들어 진게 보이시면 완료 되신 겁니다. ^_~

이제 Database 이름을 변경해 보도록 할게요.(아쉽지만, 현재 버전-Beta1 - WebMatrix 는 Databases 섹션에서 DB 이름 변경이 불가해요.)

Database 이름 변경

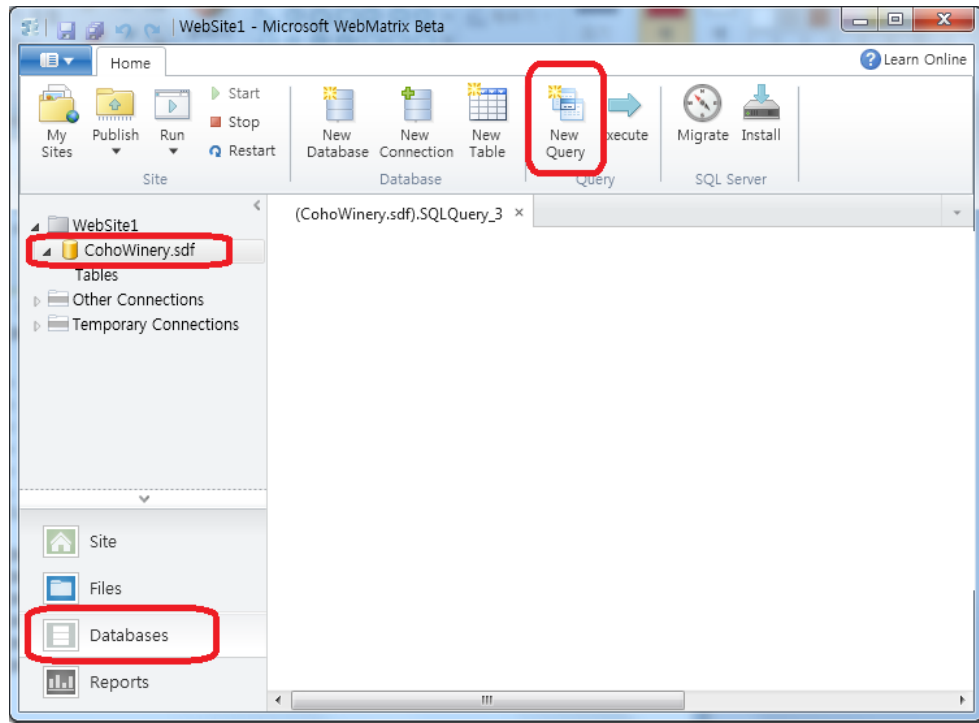


- (1) Files 를 선택하고
- (2) 해당 데이터베이스를 선택, 마우스 우버튼 후 “Rename” 이름을 변경합니다.
- (3) 이름을 저처럼 CohoWinery.sdf 로 잡아 주세요.(이후 예제에서 이 이름으로 사용합니다.)

자, 데이터베이스도 만들었으니, 이제 실제 데이터를 넣고 테스트를 하기 위한 테이블 생성, 샘플 데이터 삽입 작업을 할 차례입니다.

WebMatrix 에서 테이블 생성

WebMatrix 에서 테이블을 생성하기 위해 아래 절차를 진행 하세요~



- (1) 왼쪽 하단의 Databases 를 클릭하고,
- (2) 왼쪽 상단에서 조금 전 만든 해당 DB 를 선택 하신 후에
- (3) 중간 위의 “New Query”를 클릭해 새로운 SQL 쿼리를 수행합니다.

참고로, WebMatrix 는 쿼리가 아닌 UI 가 있는 화면으로 테이블 생성, 관계 설정 등도 가능합니다. 위의 화면 New Table 을 누르고 UI 로 제작도 가능하니 지금은 참고만 해 두세요~

테이블 생성 쿼리 수행

```
CREATE TABLE Customers (  
    CustomerID int NOT NULL  
,  
    FirstName  nvarchar(128) NOT NULL  
,  
    LastName   nvarchar(128) NOT NULL  
,  
    Address    nvarchar(256) NOT NULL  
,  
    Phone      nvarchar(15)  NOT NULL  
,  
    Email      nvarchar(128) NOT NULL  
,  
    PRIMARY KEY (CustomerID)  
);
```

위의 테이블 생성 구문을 수행하시면 테이블이 생성됩니다.

실행을 위해서는, WebMatrix 의 지금 열려 있는 쿼리 창에 붙여 넣으시고, F5 를 누르시거나, 상단 중간 정도에 위치한 “Execute”를 클릭 하셔도 됩니다.

Command(s) completed successfully 라고 나오실 거예요. 그럼 성공~

확인을 위해 왼쪽 맨 위의 우리 DB 인 “CohoWinery.sdf” 펼치고 나오는 Tables 에서 마우스 우측 버튼 – “Refresh”를 실행하면, Customers 라는 테이블이 보이실 겁니다. 또는, 왼쪽 메뉴에서 Customers 테이블을 더블클릭하거나 우클릭 후 “Data”를 클릭 하셔도 데이터 조회가 가능합니다. – 참 쉽죠? ^_^^;;

바로 이어서, 샘플 데이터를 삽입해 보도록 할게요.

샘플 데이터 삽입 – INSERT 처리

쿼리창의 CREATE TABLE 구문을 지우고 **아래 INSERT 구문을 “한 줄씩” 복사해 실행합니다.**

주의 – 아래 쿼리들을 한 Insert 구문씩 복사해 넣으세요.

```
INSERT INTO Customers VALUES (1, 'Jim', 'Corbin', '3425 23 Ave SE New York 87905', '555-0100',  
'JC@cpalnd.com');
```

```
INSERT INTO Customers VALUES (2, 'Eva', 'Coretta', '800 100 St Portland 34567', '555-0101', 'EC@cpalnd.com');
```

```
INSERT INTO Customers VALUES (3, 'Sara', 'Davis', '1123 Main St Orcas 3444', '555-0200', 'SW@contoso.com');
```

데이터 조회 구문

```
SELECT * FROM Customers;
```

(한 줄씩 복사해 개별 실행하려니 갑갑 하시죠? 현재 WebMatrix 가 MSSQL 관리 도구나 개발도구와 달리, 몇몇 부분이 조금 불편하실 수 있습니다. 정식 버전에서는 이런 불편함이 대부분 수정될 예정이라고 하니 참고 하시길 바랍니다.)

자~ 이렇게 해서 간단히 데이터베이스 작업을 수행 했습니다. 우리가 해본 작업은

- (1) 데이터베이스 생성
- (2) 테이블 생성
- (3) 쿼리 도구를 이용해 데이터 삽입, 데이터 조회

작업을 진행해 보았습니다. 그럼 이제 본편~ Razor 에서 데이터베이스를 처리하는 부분을 진행해 보도록 하지요~

Razor 에서 데이터 조회 – SELECT 처리

```
@{
    var db = Database.OpenFile("CohoWinery.sdf");
    //var db = Database.OpenFile("~/App_Data/CohoWinery.sdf");
    //var db = Database.OpenFile(@"Data Source=C:\Users\W사용자명\Documents\My Web
Sites\WebSite1\App_Data\CohoWinery.sdf");
    var selectQueryString = "SELECT * FROM Customers ORDER BY CustomerID";
}
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
    <head>
```

```
        <title>고객리스트 </title>
```

```
    </head>
```

```
    <body>
```

```
        <h2> Coho Winery 고객리스트 </h2>
```

```
        <table border="1">
```

```
            <thead>
```

```
                <tr>
```

```
                    <th>이름</th>
```

```
                    <th>성</th>
```

```
                    <th>주소</th>
```

```
                    <th>전화번호</th>
```

```
                    <th>Email</th>
```

```
                </tr>
```

```
            </thead>
```

```
            <tbody>
```

```
                @foreach (var row in db.Query(selectQueryString)) {
```

```
                <tr>
```

```
                    <td>@row.FirstName</td>
```

```
                    <td>@row.LastName</td>
```

```
                    <td>@row.Address</td>
```

```
                    <td>@row.Phone</td>
```

```
                    <td> <a href="mailto:@row.Email" style="text-decoration:
```

```
underline;">@row.Email</a></td>
```

```
                </tr>
```

```
    }  
    </tbody>  
  </table>  
</body>  
</html>
```

“DB” 폴더를 웹사이트 루트에 만들고 “01_select.cshtml” 파일을 만들고 복사해 실행합니다.

Razor 는 기존의 많은 웹 개발 방식과 달리, 더 쉽고 더 빠른 웹개발을 그 목표로 하고 있습니다.

개인적으로 저 코난이가 Razor 에 푸욱~ 빠지게 된 이유는 바로 이 데이터베이스 처리 방식. 스파게티 코드 없이 깔끔하고, DB 처리를 우리 웹 개발자에게는 눈물겹게 감사하도록, 딱 한줄로 끝내고 있습니다. DB 처리 방식이나 Helper 를 이용해 그동안 있었던 웹 개발의 어려움과 복잡함을 덜어내고, 개발자에게 더 쉽고 빠른 개발이 가능하게 돕는 녀석이지요.

지~ 코드를 봐 볼까요~

DB 연결 방식으로 Database.OpenFile 을 이용합니다. (주석들을 참고하시면 여러 DB 연결 방식 참조가 가능합니다.)

```
var db = Database.OpenFile("CohoWinery.sdf");
```

저는 이렇게 연결 했습니다. 이어서, SQL 쿼리 구문을 만들고, 아래 Foreach 구문에서 db.Query 로 한줄에 실행합니다.

```
@foreach (var row in db.Query(selectQueryString))
```

사실 위의 두 줄로 DB 연결이 끝난 겁니다. 잊지 마시고, 이 두 과정을 꼭 잘 눈에 익혀 두시면 나중에도 DB 관련 처리를 하실 때 많은 도움 되실 겁니다.

개발 경험이 있으신 분들을 위해... WebMatrix 에 기본 포함된 SQL CE 가 아니라, 타 DBMS 를 이용하실 경우(MSSQL 서버와 같은)에는 기존 방식과 마찬가지로, Connection String 을 이용해 처리가 가능합니다.(현재 WebMatrix Beta1 에서 Connection String 은 Web.config 에 저장됩니다. Connection String 구성 등은 나중에 요청이 있으시거나 하면 상세하게 풀어 볼게요.)

그럼 이어서 Razor 로 데이터 삽입 처리를 보도록 하겠습니다.

Razor 에서 데이터 삽입 – INSERT 처리

```
@{
    var db = Database.OpenFile("CohoWinery.sdf");
    var CustomerID = Request.Form["CustomerID"];
    var FirstName = Request.Form["FirstName"];
    var LastName = Request.Form["LastName"];
    var Address = Request.Form["Address"];
    var Phone = Request.Form["Phone"];
    var Email = Request.Form["Email"];

    if (IsPost) {
        // 사용자의 입력 값 저장 & 입력값을 검사
        CustomerID = Request["CustomerID"];
        if (CustomerID.IsNullOrEmpty()) {
            Validation.AddFieldError("CustomerID", "고객 ID 번호를 입력하세요.");
        }
        // 사용자의 입력 값 저장 & 입력값을 검사
        FirstName = Request["FirstName"];
        if (FirstName.IsNullOrEmpty()) {
            Validation.AddFieldError("FirstName", "이름을 입력하세요.");
        }
        // 사용자의 입력 값 저장 & 입력값을 검사
        LastName = Request["LastName"];
        if (LastName.IsNullOrEmpty()) {
            Validation.AddFieldError("LastName", "성을 입력 하세요.");
        }
        // 사용자의 입력 값 저장 & 입력값을 검사
        Address = Request["Address"];
        if (Address.IsNullOrEmpty()) {
            Validation.AddFieldError("Address", "주소를 입력하세요.");
        }
        // 사용자의 입력 값 저장 & 입력값을 검사
        Phone = Request["Phone"];
        if (Phone.IsNullOrEmpty()) {
            Validation.AddFieldError("Phone", "전화번호를 입력하세요.");
        }
    }
}
```

```

// 사용자의 입력 값 저장 & 입력값을 검사
Email = Request["Email"];
if (Email.IsNullOrEmpty()) {
    Validation.AddFieldError("Email", "이메일 주소를 입력하세요.");
}
if(Validation.Success) {
    // 입력받은 값을 파라미터화 후 insert 구문에 전달
    var insertQueryString =
        "INSERT INTO Customers (CustomerID, FirstName, LastName, Address, Phone, Email) VALUES (@0,
@1, @2, @3, @4, @5)";
    // 쿼리 실행
    db.Execute(insertQueryString, CustomerID, FirstName, LastName, Address, Phone,
Email);

    // 삽입 후 데이터 조회 페이지로 이동
    Response.Redirect("01_select.cshtml");
}
}
}

```

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

```

```

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

```

```

<html xmlns="http://www.w3.org/1999/xhtml">

```

```

    <head>

```

```

        <title>고객 정보 추가</title>

```

```

        <style type="text/css">

```

```

            label {

```

```

                float: left;

```

```

                width: 8em;

```

```

                text-align: right;

```

```

                margin-right: 0.5em;

```

```

            }

```

```

            fieldset {

```

```

                padding: 1em;

```

```

                border: 1px solid;

```

```

                width: 50em;}

```

```

            legend {

```

```

                padding: 0.2em 0.5em;

```

```
border: 1px solid;
font-weight: bold;
}
</style>
</head>
<body>
  <h2>새로운 고객 정보 추가</h2>
  <div style="font-weight:bold; color:red">
    @Html.ValidationSummary("입력값 오류가 있습니다.")
  </div>
  <form method="post" action="">
    <fieldset>
      <legend>고객 추가</legend>
      <div>
        <label>고객 ID 번호:</label>
        <input name="CustomerId" type="text" size="50" value="@CustomerId"
/>

      </div>
      <div>
        <label>이름:</label>
        <input name="FirstName" type="text" size="50" value="@FirstName" />
      </div>
      <div>
        <label>성:</label>
        <input name="LastName" type="text" size="50" value="@LastName" />
      </div>
      <div>
        <label>주소:</label>
        <input name="Address" type="text" size="50" value="@Address" />
      </div>
      <div>
        <label>전화번호:</label>
        <input name="Phone" type="text" size="50" value="@Phone" />
      </div>
      <div>
        <label>Email:</label>
        <input name="Email" type="text" size="50" value="@Email" />
      </div>
    </fieldset>
  </form>
</body>
</html>
```

```

        </div>
        <div>
            <label>&nbsp;</label>
            <input type="submit" value="Insert" class="submit" />
        </div>
    </fieldset>
</form>
</body>
</html>

```

02_Insert.cshtml 파일로 저장합니다.

참고하실 내용이 두 개 정도가 있는데요. INSERT 구문을 수행하실 경우에는 이렇게 파라미터화 된 값을 전달합니다.

// 입력 받은 값을 파라미터화 후 insert 구문에 전달

```
var insertQueryString =
```

```
"INSERT INTO Customers (CustomerID, FirstName, LastName, Address, Phone, Email) VALUES (@0, @1, @2, @3,
@4, @5)";
```

또한, 이번에 새롭게 등장한 또 하나의 Razor Helper 인 “Html.ValidationSummary” 녀석인데요.

이 Html.ValidationSummary 을 이용하면, 입력 값을 손쉽게 검사해 입력 오류 여부를 사용자에게 쉽게 알릴 수 있습니다.

간단히 빈 값을 입력하게 되면 해당 체크 영역에서 값을 처리해 @Html.ValidationSummary 영역에 표시하게 됩니다.

새로운 고객 정보 추가

입력값 오류가 있습니다.

- 주소를 입력하세요.
- 전화번호를 입력하세요.
- 이메일 주소를 입력하세요.

고객 추가

고객ID번호: 6

이름: aaa

성: bbb

주소:

전화번호:

Email:

Insert

요렇게 자동으로 도와 줍니다. 개발자의 번거로운 작업인 입력값 체크 처리를 간략화 시킬 수 있는 깔끔한 녀석이에요. 우리들의 경우 이런 validation 을 쓰는 개발자 분들과 UI 를 위해 직접 만들어 사용하시는 경우도 꽤 있는 것 같습니다.

다음은 Update 를 알아 보도록 할게요. 이제 데이터베이스 처리 좀 감이 잡히시죠? Update 도 전혀~ 어렵지 않습니다.

Razor 로 데이터베이스 수정 – Update 처리

데이터를 수정하려면, 먼저 조회를 하고, 수정 루틴을 넣은 후에 수정 화면이 있어야겠지요. 조회는 위의 SELECT 와 유사하고, Update 값 화면은 INSERT 화면과 유사합니다.

```
@{
    var db = Database.OpenFile("CohoWinery.sdf");
    var selectQueryString = "SELECT * FROM Customers ORDER BY CustomerID";
}

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>고객 정보 수정</title>
    </head>
    <body>
        <h2>        고객 정보 수정        </h2>
        <form method="post" action="" name="form">
            <table border="1">
                <thead>
                    <tr>
                        <th/>
                    </tr>
                    <th>이름</th>
                    <th>성</th>
                    <th>주소</th>
                    <th>전화번호</th>
                    <th>Email</th>
                </tr>
            </thead>
            <tbody>
                @foreach \(var row in db.Query\(selectQueryString\)\) {
                    <tr>
```

```

                <td><a href="@Href("~/db/04_update_02.cshtml",
row.CustomerID)">수정</a></td>

                <td>@row.FirstName</td>
                <td>@row.LastName</td>
                <td>@row.Address</td>
                <td>@row.Phone</td>
                <td><a href="mailto:@row.Email" style="text-decoration:
underline;">@row.Email</a></td>
            </tr>
        }
    </tbody>
</table>
</form>
</body>
</html>

```

03_update_01.cshtml 파일로 저장해 두겠습니다. 이어서 UPDATE 에 필요한 후속 파일을 만들어 주세요. 위의 “04_update_02.cshtml” 페이지로 CustomerID 를 넘겨주고 있어요. 참고 하시길 바랍니다.

```

@{
    var db = Database.OpenFile("CohoWinery.sdf");
    var selectQueryString = "SELECT * FROM Customers WHERE CustomerID=@0";
    // 업데이트를 위해 고객 ID 번호로 정보를 읽어 옵니다.
    // 고객 ID 는 앞의 update 페이지에서 넘겨 받았습니다.
    var CustomerID = UrlData[0];
    if (CustomerID.IsNullOrEmpty()) {
        Response.Redirect("03_update_01.cshtml");
    }
    // 업데이트를 위해 고객 정보를 조회
    var row = db.QuerySingle(selectQueryString, CustomerID);
    // Update 입력 상자에 넣기 위해 변수에 값을 저장
    var FirstName = row.FirstName;
    var LastName = row.LastName;
    var Address = row.Address;
    var Phone = row.Phone;

```

```

var Email      = row.Email;

if (IsPost) {
    CustomerID = Request["CustomerID"];
    if (CustomerID == null) {
        Validation.AddFieldError("CustomerID", "고객 ID 번호를 입력하세요.");
    }
    FirstName = Request["FirstName"];
    if (String.IsNullOrEmpty(FirstName)) {
        Validation.AddFieldError("FirstName", "이름을 입력하세요.");
    }
    LastName = Request["LastName"];
    if (String.IsNullOrEmpty(LastName)) {
        Validation.AddFieldError("LastName", "성을 입력하세요.");
    }
    Address = Request["Address"];
    if (String.IsNullOrEmpty(Address)) {
        Validation.AddFieldError("Address", "주소를 입력하세요.");
    }
    Phone = Request["Phone"];
    if (String.IsNullOrEmpty(Phone)) {
        Validation.AddFieldError("Phone", "전화번호를 입력하세요.");
    }
    Email = Request["Email"];
    if (String.IsNullOrEmpty(Email)) {
        Validation.AddFieldError("Email", "이메일 주소를 입력하세요.");
    }
    if(Validation.Success) {
        // 업데이트 처리를 위해 파라미터화 작업
        var updateQueryString = "UPDATE Customers SET FirstName=@0, LastName=@1, Address=@2, Phone=@3,
Email=@4 WHERE CustomerID=@5" ;
        // UPDATE 쿼리 구문 실행
        db.Execute(updateQueryString, FirstName, LastName, Address,
            Phone, Email, CustomerID);
        // 업데이트를 위한 리스트 페이지로 이동
        Response.Redirect("~/db/03_update_01.cshtml");
    }
}

```

```

}
}
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>고객 정보 수정</title>
    <style type="text/css">
      label {
        float: left;
        width: 8em;
        text-align: right;
        margin-right: 0.5em;
      }
      fieldset {
        padding: 1em;
        border: 1px solid;
        width: 50em;
      }
      legend {
        padding: 0.2em 0.5em;
        border: 1px solid;
        font-weight: bold;
      }
    </style>
  </head>
  <body>
    <h2>      고객 정보 수정 페이지      </h2>
    <div style="font-weight:bold; color:red">
      @Html.ValidationSummary\("입력값에 오류가 있습니다 :"\)
    </div>
    <form method="post" action="">
      <fieldset>
        <legend>고객정보 수정</legend>
        <div>
          <label>고객 ID 번호:</label>
          <input name="CustomerID" type="text" size="50" value="@CustomerID" />

```

```

        </div>
        <div>
            <label>이름:</label>
            <input name="FirstName" type="text" size="50" value="@FirstName" />
        </div>
        <div>
            <label>성:</label>
            <input name="LastName" type="text" size="50" value="@LastName"
/>

        </div>
        <div>
            <label>주소:</label>
            <input name="Address" type="text" size="50" value="@Address" />
        </div>
        <div>
            <label>전화번호:</label>
            <input name="Phone" type="text" size="50" value="@Phone" />
        </div>
        <div>
            <label>Email:</label>
            <input name="Email" type="text" size="50" value="@Email" />
        </div>
        <div>
            <label>&nbsp;</label>
            <input type="submit" value="Update" class="submit" />
        </div>
    </fieldset>
</form>
</body>
</html>

```

04_update_02.cshtml 파일로 저장합니다. 실행을 위해 앞에서 만들었던 03_update_01.cshtml 업데이트 리스트 페이지를 실행하세요.

위의 업데이트 화면은 Insert 처리와 마찬가지로 validation 후에 파라미터화 된 값을 처리하게 되면서 마무리 됩니다. 마지막 삭제처리 - DELETE 를 알아 보도록 할게요~

Razor 로 데이터베이스 값 삭제 – DELETE 처리

어느덧 마지막 내용이네요. 삭제 처리를 위한 DELETE 는 더 쉽습니다. 데이터 조회 리스트 화면에서 해당 값을 선택해 “삭제”를 클릭하면 처리가 완료되는 처리 입니다.

```
@{
    var db = Database.OpenFile("CohoWinery.sdf");
    var selectQueryString = "SELECT * FROM Customers ORDER BY CustomerID";
    if (IsPost) {
        var deleteQueryString = "DELETE FROM Customers WHERE CustomerID=@0";
        db.Execute(deleteQueryString, Request["rowID"]);
    }
}

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>고객 정보 삭제</title>
        <script type="text/javascript">
            function deleteRow\(rowID\) {
                var answer = confirm\("정말 삭제 하시겠습니까?"\)
                if \(answer==true\)
                {
                    document.form.rowID.value = rowID;
                    document.form.submit\(\);
                }
            }
        </script>
    </head>

    <body>
        <h2 class="SubHeader">        고객 정보 삭제        </h2>
        <form method="post" action="" name="form">
            <input type="hidden" name="rowID" value="" />
            <table border="1">
                <thead>
                    <tr>
                        <th/>
```

```

        <th>이름</th>
        <th>성</th>
        <th>주소</th>
        <th>전화번호</th>
        <th>Email</th>
    </tr>
</thead>
<tbody>
    @foreach (var row in db.Query(selectQueryString)) {
    <tr>
        <td><a href="Javascript:deleteRow('@row.CustomerID')">
삭제</a></td>
        <td>@row.FirstName</td>
        <td>@row.LastName</td>
        <td>@row.Address</td>
        <td>@row.Phone</td>
        <td><a href="mailto:@row.Email" style="text-
decoration:underline;">@row.Email</a></td>
    </tr>
    }
</tbody>
</table>
</form>
</body>
</html>

```

05_del.cshtml 파일로 저장하고 실행하세요.

마지막으로, 데이터를 손쉽게 웹에서 표시해주는 WebGrid Helper 를 소개해 드리려고 합니다. 기본 제공되는 Grid 로 그리드로 사용하기 편리하실 거예요. 자동 페이징이나, 쉬운 디자인 등등... 이번에는 맛보기 정도만 하고, 나중에 기회되면 이 WebGrid 도 잘 풀어 보도록 할게요. ^_^

WebGrid Helper 를 이용한 손쉬운 데이터 표시

```

@{
    var db = Database.OpenFile("CohoWinery.sdf");
    var selectQueryString = "SELECT * FROM Customers";
}

```

```

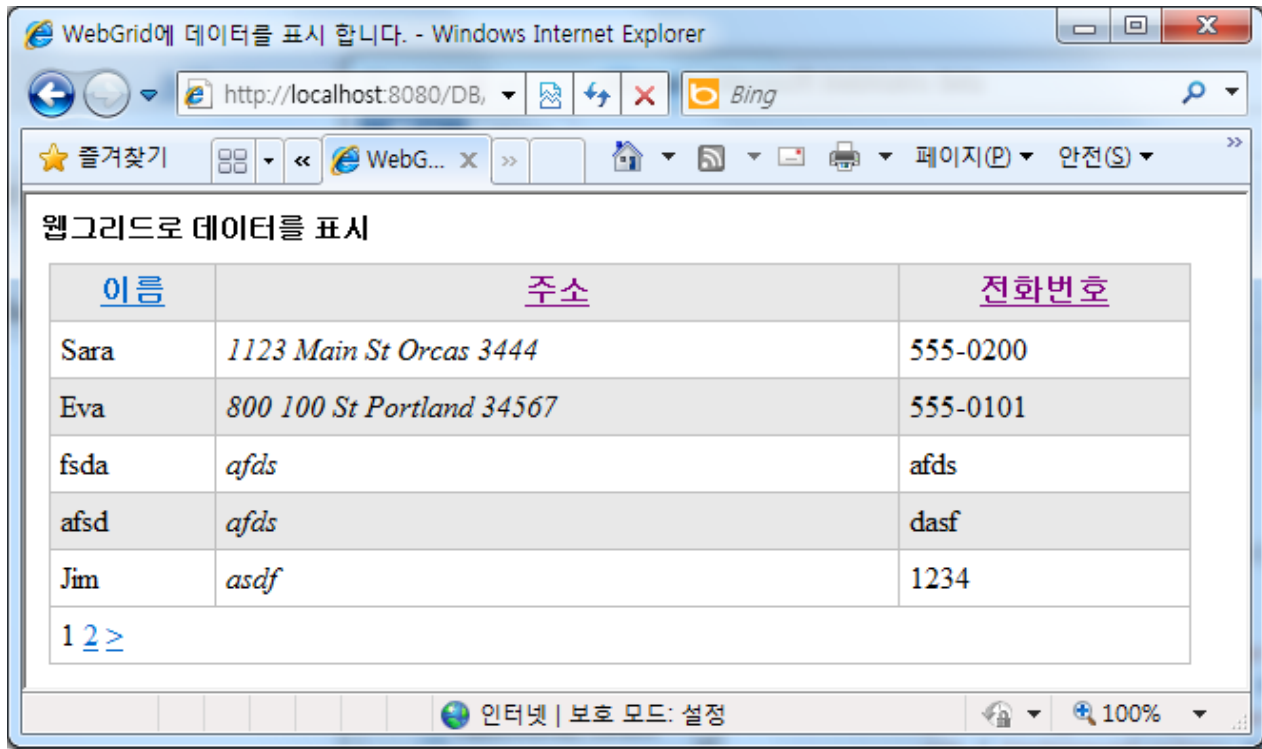
var data = db.Query(selectQueryString);
var grid = new WebGrid(data, defaultSort: "FirstName", rowsPerPage: 5);
}

<!DOCTYPE html>
<html>
  <head>
    <title> WebGrid 에 데이터를 표시 합니다. </title>
    <style type="text/css">
      h1 {font-size: 14px;}
      .grid { margin: 4px; border-collapse: collapse; width: 600px; }
      .head { background-color: #E8E8E8; font-weight: bold; color: #FFF; }
      .grid th, .grid td { border: 1px solid #C0C0C0; padding: 5px; }
      .alt { background-color: #E8E8E8; color: #000; }
      .FirstName { width: 200px; }
    </style>
  </head>
  <body>

    <h1>웹그리드로 데이터를 표시</h1>
    @grid.GetHtml(
      tableStyle: "grid",
      headerStyle: "head",
      alternatingRowStyle: "alt",
      columns: grid.Columns(
        grid.Column("FirstName", "이름", style: "product"),
        grid.Column("Address", "주소", format:@<i>@item.Address</i>),
        grid.Column("Phone", "전화번호", format:@<text>@item.Phone</text>)
      )
    )
  </html>

```

WebGrid 로 저장하고 실행해 보세요. 그럼 이런 형태로 보이실 겁니다. ^_^



짧은 코드로 그리드가 아주 깔끔하게 나오죠? 이런 방법도 Razor 와 WebMatrix 는 기본제공하고 있으니 참고 하시길 바랍니다.

Razor 와 WebMatrix 를 이용한 데이터베이스 개발의 장점은?

데이터 처리에 대한 예를 말씀 드렸어요. 어쩌면 Razor 가 기존의 웹개발 방식과 가장 차별화를 제공하는 부분이 바로 이 데이터베이스 처리 부분일 겁니다.

Razor & WebMatrix 의 데이터베이스 개발 작업의 장점은

- (1) 자체 내장된 SQL DB 를 이용하고
- (2) 무료 SQL 서버인 SQL Express 나 SQL 서버 상위 버전으로 100% 마이그레이션을 WebMatrix 내부에서 지원하며
- (3) 간결한 데이터베이스 접근 처리 방식
- (4) 개발과 쿼리, 데이터베이스 작업을 모두 하나의 도구인 WebMatrix 에서 제공

하는 것은 Razor 와 데이터베이스 개발에 대한 큰 매력이 아닐까 생각합니다.

데이터베이스에 경험이 없으신 분들은 내용이 조금 어려웠을지도 모르겠습니다. 곧, 이 강좌에 대해서도 동영상 강좌를 준비하고 있으니 많은 도움 되시길 바랍니다.

그럼 다음 강좌인 (9) Razor 강좌 - Helper 소개(이미지, 비디오) 에서 뵙도록 하겠습니다. ^_^

(9) Razor 강좌 - Helper 소개(이미지, 비디오)

지난 시간에는 “(8) Razor 강좌 - 데이터베이스 처리” 강좌를 진행 했습니다.

이번 시간에는 바로 이어서, “(9) Razor 강좌 - Helper 소개(이미지, 비디오)” 강좌를 진행 하도록 하겠습니다.

Razor 와 WebMatrix 에서 Helper 는 무엇인가요?

넵. 지난 강좌에서도 몇 번 소개해 드렸습시다만, Helper 는 웹에서 사용되는 여러 기능들을 손쉽게 이용 가능하도록 구조화시킨 모듈입니다. 즉, 다양하고, 복잡한 웹의 기능을 개발자가 쉽게 이용 가능하도록 돕는 부품이지요.

그렇다면, Helper 는 어떤 종류가 있나요?

Razor 는 이러한 Helper 를 기본적으로 포함하고 있으며, 사용자가 직접 만들어 사용할 수 있도록 역시 돕고 있습니다. 기본적으로 현재 WebMatrix Beta1 에 포함된 Helper 들의 리스트를 보여 드리자면 아래와 같아요.

참조 : [WebMatrix Helpers 종류](#) - 영문

위의 링크를 통해 대략 이런 종류의 다양한 Helper 들이 Beta1 에서 기본 제공되고 있구나~ 정도만 보셔도 좋을 것 같습니다. Beta1 에서 이 정도로 많고 앞으로 공개 예정인 녀석들 등도 많이 있으니 정식 버전에서는 엄청나겠지요? ^^ 기대됩니다.

이번 강좌에서는 이러한 Helper 들 중에서 종종 사용 하시게 될 Image 처리, Video 처리, 두개 정도의 기본 Helper 를 상세히 살펴봐서 어떻게 Helper 를 이용하는지, 또한 Helper 가 얼마나 우리의 웹 개발 닭질을 줄여 줄지 기대해보는 시간을 가지도록 하겠습니다.

참조 : [사용자가 직접 Helper 를 만들고 사용하는 방법](#) - 영문

Image 처리 Helper - WebImage

소개해 드릴 녀석은 WebImage Helper 입니다. 이 녀석으로 어떤걸 할 수 있냐고요?

- 웹 서버 경로의 이미지 로드
- 사용자에게 의해 업로드 된 이미지 로드
- 이미지 파일을 웹 서버 특정 경로에 저장
- 이미지에 워터마크(텍스트 또는 이미지) 추가 표시
- 이미지 Flip 이나 회전
- 이미지 크기조절(Resize 를 이용한 썸네일 표시)

와 같은 웹 개발에서 종종 사용하게 될 기능을 제공합니다.

그럼 요 기능들 중에서 썸네일로 사용하기 위한 크기조절(Resize) 처리와 텍스트 워터마크 기능을 살펴 보도록 하겠습니다.

아! 제 강좌는 수많은, 모든 Helper 들의 기능이나 사용 예를 보여드리는 게 아니에요. 제가 보여드릴 소개 샘플들을 봐 보시면서 Helper 란 이런 것이고, 이런 목적으로 Helper 를 사용하고, 이렇게 Helper 이용 가능하구나 부분에 초점을 맞추셔야, 앞으로도 쏟아져 나오게 될 수많은 Helper 들 중에 원하시는 Helper 를 직접 찾아 사용 가능하실 겁니다. 막상 이용 방법 설명이라고 해 봐야 코드 한 줄 정도에 파라미터 값들 넣는 게 다네요.(쿨럭) 워낙 쉬워서 사용 방법 설명이라고 하기도 좀 거시기 합니다. ^_~;;; 그럼 시작해 볼게요~

WebImage – 섬네일 이미지 표시(Resize)

예제 실행을 위해 WebMatrix 로 “WebImage” 폴더를 만들고, 그 하위에 “images”, 또 그 하위에 “tumbs” 폴더를 미리 생성합니다.

```
@{  
    WebImage photo = null;  
    var newFileName = "";  
    var imagePath = "";  
    var imageThumbPath = "";  
    if(IsPost){  
        photo = WebImage.GetImageFromRequest();  
        if(photo != null){  
            newFileName = Guid.NewGuid().ToString() + "_" +  
                Path.GetFileName(photo.FileName);  
            imagePath = @"images\W" + newFileName;  
            photo.Save(@"~\WebImage\W" + imagePath);  
            imageThumbPath = @"images\Wthumbs\W" + newFileName;  
            photo.Resize(  
                width: 60,  
                height: 60,  
                preserveAspectRatio: true,  
                preventEnlarge: true  
            );  
            photo.Save(@"~\WebImage\W" + imageThumbPath);  
        }  
    }  
}  
<!DOCTYPE html>  
<html>  
    <head>  
        <title>이미지 리사이징</title>
```

```

</head>
<body>
  <h1>섬네일 이미지 자동 추출</h1>
  <form action="" method="post" enctype="multipart/form-data">
    <fieldset>
      <legend> 섬네일 이미지 생성 </legend>
      <label for="Image">이미지</label>
      <input type="file" name="Image" />
      <br/>
      <input type="submit" value="Submit" />
    </fieldset>
  </form>

  @if(imagePath != ""){
    <div class="result">
      
      <a href="@Html.AttributeEncode(imagePath)" target="_Self">
        실제이미지 보기
      </a>
    </div>
  }
</body>
</html>

```

WebImage 폴더에 ResizeImage.cshtml 파일로 저장하시고 실행하세요.

아무 사진이나 업로드를 하시면 됩니다.코드를 통해 처리되는 내용을 설명 드리면,

- 이미지를 form 으로 업로드 받고
- WebImage 개체인 photo 에 GetImageFromRequest 로 이미지가 로드 됩니다.
- Save 를 통해 업로드 된 이미지를 저장하고,
- 이어서 WebImage.Resize 로 이미지 크기를 줄여
- Thumbnail 이미지를 한번 더 Save 하게 되지요.

WebImage Helper - 이미지에 워터마크(Watermark) 표시

다음으로 워터마크(Watermark) 처리를 확인해 보도록 할게요. 이미지 소스에 대한 보호, 불법적인 이미지 도용 방지 등의 목적으로 보통 이 워터마크 방식을 이용하게 됩니다. 예제를 보면서 설명 드리도록 하겠습니다.

```

@{
    var imagePath = "";
    WebImage photo = new WebImage("images/photo.jpg");
    if(photo != null){
        imagePath = "images/photo.jpg";
        photo.AddTextWatermark("Photo by http://www.sqler.com - 워터마크", fontColor:"Yellow", fontFamily:"Arial");
        photo.Save(imagePath);
    }
}
<!DOCTYPE html>
<html>
<head>
    <title> 워터마크 이미지 처리 </title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
</head>
<body>
    <h1>이미지에 워터마크를 텍스트를 추가합니다.</h1>
    @{if(imagePath != ""){
        <div class="result">
            
        </div>
    }
    </body>
</html>

```

WeblImages 폴더 하위에 watermark.cshtml 로 저장하세요. 실행을 위해선 WeblImages 폴더 아래 images 폴더에 jpg 파일을 두세요. 저는 위의코드처럼 photo.jpg 파일로 두었습니다. 이 파일에 대해 워터마크를 찍는 처리 입니다.

코드 진행에 대해선 설명이 많이 필요하지 않으실 듯 합니다.

이렇게 “WebImage.AddTextWatermark”를 호출해 텍스트기반 워터마크 추가가 가능합니다. 또한,

“WebImage.AddImagesWatermark”를 이용하면 이미지를 이용해 워터마크 처리가 가능해 집니다. 이렇게, Razor 는 즉각적인 이미지 처리가 가능한 Helper 등을 통해, 손쉬운 이미지 기반 처리를 가능하게 개발자 편의성을 제공하고 있어요.
- 생유생유~

실행 결과는 아래 처럼 보이실 겁니다.



아래쪽에 이렇게 아래쪽에 Watermark 텍스트가 찍히실 거예요. (오늘도 자식 자랑 성공~ 아자~ 텃터터~)

이렇게 간단히 이미지 Resize, 워터마크 처리에 대해서 알아 보았습니다.

다음으로는 웹에서 실행 가능한 Video 처리에 대해서 소개해 드리도록 하겠습니다.

WebMatrix & Razor 에서 Video helper 로 Flash, Silverlight, WMV 비디오나 어플리케이션 실행

느끼시는 것처럼, Razor에서는 Adboe Flash 비디오와 WMV 비디오 파일(스트리밍도) 플레이도 가능하며 Microsoft Silverlight 파일인 xap 파일도 처리가 가능합니다. 이번 소개에서는 간단히 Silverlight 를 실행해 보도록 하겠습니다..

```
<!DOCTYPE html>
<html>
  <head>
    <title>Silverlight Video 실행</title>
  </head>
  <body>
```

```

@Video.Silverlight(
    path: "http://www.sqler.com/lab/sl\_install/publish/ClientBin/SLTest01.xap",
    width: "300",
    height: "200",
    bgColor: "Black",
    autoUpgrade: true
)
<p> Silverlight 실행 </p>
</body>
</html>

```

웹사이트 상위에 Video 폴더를 만들고, 여기에 Silverlight.cshtml 파일로 저장 했습니다.

정확히 Flash 나 Silverlight, 미디어 플레이어와 같은 처리에 사용되는 조금은 복잡한 HTML 인 <OBJECT> 태그를 자동으로 생성해주는 처리라고 보시면 됩니다. 위와 같이 Video 를 실행하면 실버라이트를 HTML 에서 수행토록 하는 아래 코드가 생성되지요.

```
<object width="300" height="200" type="application/x-silverlight-2" data="data:application/x-silverlight-2," >
```

```
<param name="source" value="http://www.sqler.com/lab/sl\_install/publish/ClientBin/SLTest01.xap" />
```

```
<param name="background" value="Black" />
```

```
<a href="http://go.microsoft.com/fwlink/?LinkId=149156" style="text-decoration:none">
```

```

```

```
</a>
```

```
</object>
```

예상대로(?)시죠? 복잡한 HTML 코드 등을 자동으로 캡슐화해 생성해 줍니다. - 개발자는 여러 종속성이나 복잡성 등을 고민할 필요 없이 개발 작업간 코드를 표준화, 캡슐화 키셔서 손쉽게 개발에 적용은 가능할 것 같아요.

video helper 의 경우 Adobe Flash 는 swf 파일, Silverlight 은 xap 파일만 처리가 가능합니다. 아울러, 꼭 Video 플레이어일 필요는 없습니다. swf 파일이나 xap 파일이면(어플리케이션도) 됩니다. WMV 는 자동으로 미디어 플레이어가 붙게 됩니다.

자~ 이렇게 해서 Helper 들에 대해서 알아 보았습니다. Helper 에 대해서 정리해 보자면,

- Helper 는 웹에서 사용되는 여러 기능들을 손쉽게 이용 가능하도록 구조화시킨 모듈입니다.
- 다양하고, 복잡한 웹의 기능을 개발자가 쉽게 이용 가능하도록 돕는 부품이지요.

이번 강좌를 통해 Image 처리 등을 보시고 아마 그렇게 느끼셨을 거예요. Razor 님은 분명 웹 개발자의 편의성을 위해 신경 좀 써 주신 듯 하네요. ^_^ 앞으로 WebMatrix 와 Razor 가 발전하면서 계속 다양하고 훌륭한 Helper 들이 많이 나오게 될 예정이라고 합니다. 이번 강좌 내용으로 다양한 Helper 를 사용하는 방법이 도움 되셨길 바랍니다.

그럼 다음 강좌인 (10) Razor 강좌 - 디버깅 에서 뵙도록 하겠습니다. 감사합니다. ^_^

참고자료 :

[사용자가 직접 Helper 를 만들고 사용하는 방법](#)

[WebMatrix Helpers 종류](#)

(10) Razor 강좌 – 디버깅

지난 시간에는 (9) Razor 강좌 – Helper 소개(이미지, 비디오)”강좌를 진행 했습니다.

이번 시간에는 바로 이어서, “(10) Razor 강좌 – 디버깅”강좌를 진행 하도록 하겠습니다.

Razor 에서 디버깅(Debugging)

웹 개발 이라는게 워낙 역인 기술들이 짬뽕된, 짜장같은(?) 영역이 많아 우리 웹 개발자 분들 참 고생 많습니다. 특히, 데이터베이스 연계 부분이나, 사이트 제작에 필수적인 게시판이나 CMS 오픈소스 프로젝트 한두개 정도는 꿰고 있어야 하고, 디자인 영역이나 웹 저작 영역인 CSS 까지도 이해가 필요합니다. 그뿐만가요, 최근에는 Open API 나 외부 서비스 연동 관련 기술은 물론, Flash 나 Silverlight 과 같은 RIA 쪽도 이해가 필요한 상황이지요.

이런 짜장들이 짬뽕되어(?) 뒹구는 웹개발 업무에, 다방면에 걸쳐 고민해야할 "디버깅"은 참 만만한 녀석이 아닙니다. 이게 서버의 오류인지, 내려 보낸 Ajax 스크립트의 문제인지, 사용자 브라우저 환경 문제인지, DB 문제인지 걸러내는 것만 해도, 어떤 경우에는 고민스러울 경우가 많지요. 하지만 Razor 에서는 조금 숨 돌리실만 하실 겁니다. WebMatrix 와 Razor 는 가장 최근에 발표된 웹 개발 방식인 만큼, 이러한 복잡한 웹 개발에 대한 디버깅을 위해 여러 기능들을 제공합니다.

저와 같이 Razor 디버깅에 대해서 차근차근 살펴 보시지요~

Razor 와 WebMatrix 의 디버깅 방식을 간단히 정리해 보면

- ServerInfo Helper 를 이용해 서버 설정 정보를 일괄 출력, 디버깅에 이용 가능합니다.(PHP 하신 분들이라면, phpinfo 와 유사하다고 보시면 됩니다.)
- 10 년 넘는 역사를 가지고 있는 기존 ASP.NET 의 디버깅-오류 정보 기능을 그대로 이용합니다.
- WebMatrix 자체 제공하는 Request 처리 기록을 이용해 오류 재현 등이 손쉽게 가능합니다.
- 기존 ASP 나 PHP 의 디버깅 방식인 출력값 처리 디버깅(변수 값 찍어보기) 형태로도 디버깅이 가능합니다.
- ObjectInfo Helper 를 이용해 추적을 원하는 개체 정보를 정확히 얻어 디버깅에 도움을 받을 수 있습니다.
- WebMatrix 에 기본 포함된 SEO 리포트 툴을 이용해 SEO 관련 오류를 쉽게 처리 가능합니다.

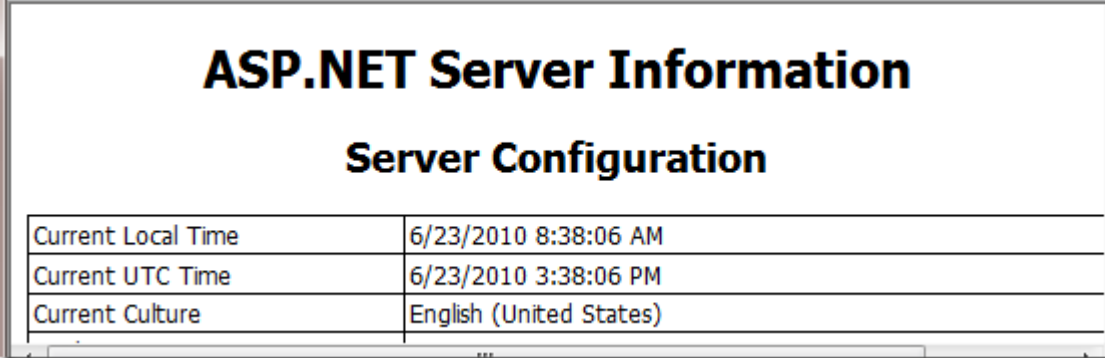
ServerInfo 로 서버 설정 및 환경 정보 조회

서버의 설정이나 환경 정보를 일괄 조회해 디버깅이나 설치 등이 잘 이루어졌는지 등을 알아볼 수 있습니다. 바로 디버깅의 시작이지요. cshtml 동작 여부 역시 이 테스트를 통해 확인 가능하지요. 아래처럼 수행 가능합니다.

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
  </head>
  <body>
```

```
@ServerInfo.GetHtml()  
  
</body>  
  
</html>
```

웹사이트 루트에 Debug 라는 폴더를 만들고 하위에 ServerInfo.cshtml 파일을 만들고 위와 같이 올려 두었습니다.
이렇게 ServerInfo Helper 를 실행하시면, 서버의 설정, 구성 환경 정보를 한눈에 보실 수 있습니다.



The screenshot shows a web browser window displaying the 'ASP.NET Server Information' page. The page has a title 'ASP.NET Server Information' and a subtitle 'Server Configuration'. Below the subtitle is a table with three rows of server configuration data.

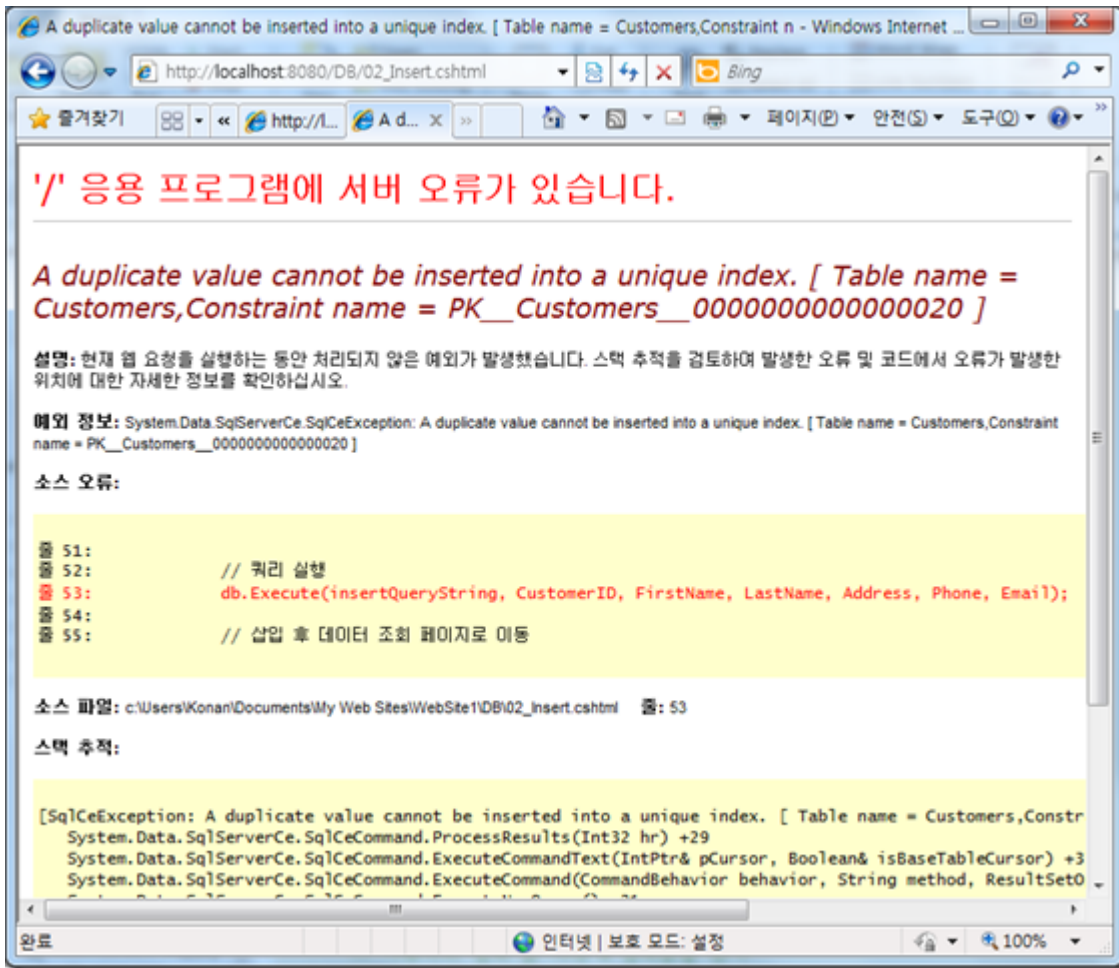
Server Configuration	
Current Local Time	6/23/2010 8:38:06 AM
Current UTC Time	6/23/2010 3:38:06 PM
Current Culture	English (United States)

PHP 에 경험이 있는 분이라면 바로 감이 오실겁니다. phpinfo() 와 같은 역할을 수행 한다고 보시면 됩니다. 다음으로 ASP.NET 디버깅 인터페이스를 봐 보도록 할께요.

Razor 는 10 년동안 서비스된 ASP.NET 의 정확하고 상세한 오류 페이지 정보를 이용 합니다.

Razor 는 ASP.NET 의 성능과 안정성에 PHP 처럼, 아니 제 경험상 PHP 이상으로 더 쉬운 개발 방식을 제공하는 기술입니다.
기본 베이스 엔진을 ASP.NET 으로 이용하기 때문에 근간이 되는 디버깅이나 오류 정보를 확인 가능하지요.

예를 들어, Razor 로 데이터베이스 삽입을 하다가 이런 오류가 발생한다면.(예시 오류 입니다. 재현하려 애쓰지 마세요~
^_^(;;))

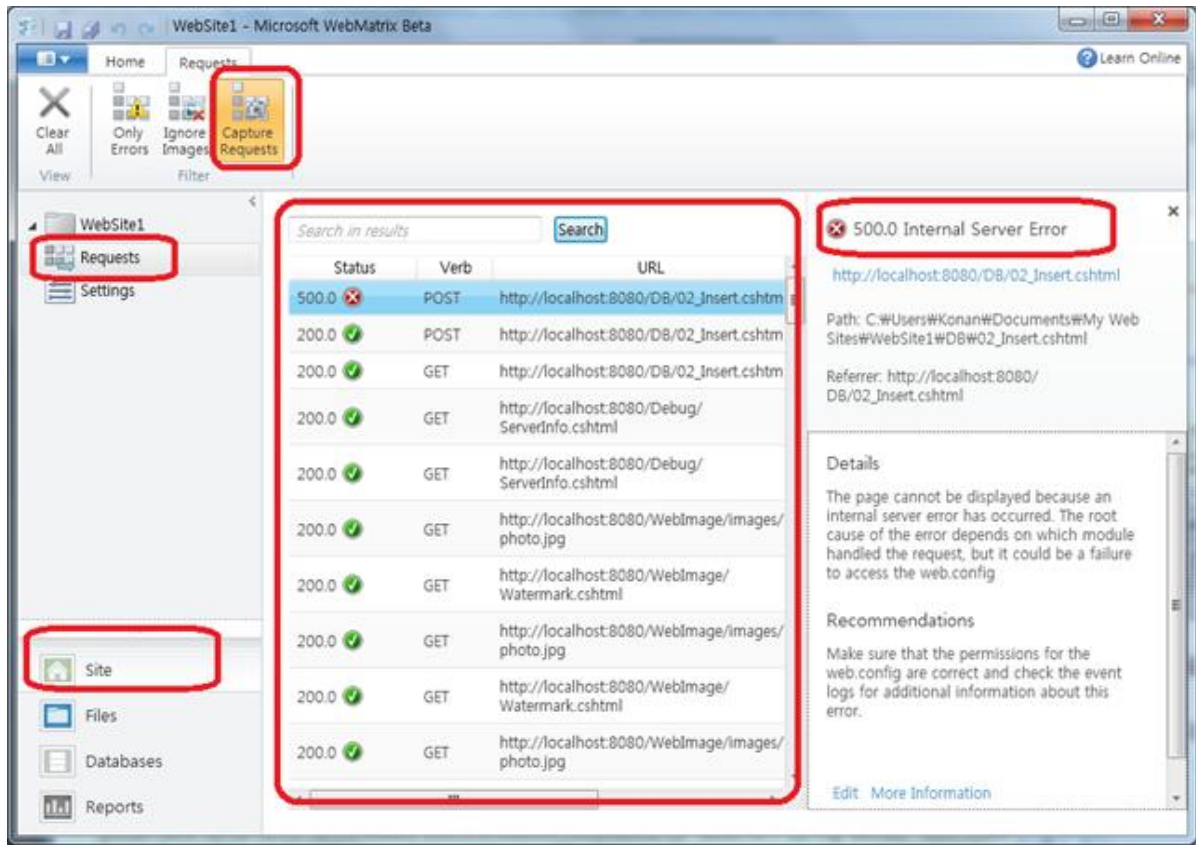


개발하면서 이렇게 잘 뽑혀져 나온 오류 정보만 쓱쓱~ 읽어도 어디 코드가 어떤 오류를 유발했는지 예측과 확인이 가능합니다. - 상세한 추적을 원할 경우에는 Call Stack 정보를 보셔도 어떤 루틴을 타다가 오류가 발생했는지도 확인 가능하지요.(물론 대부분의 운영-Production 환경에서는 이런 화면은 내부적으로 저장만 시키고, 예쁜 일반 사용자 대상 UI 의 화면을 출력 하고 계실 겁니다.)

잊지마세요, Razor 는 어느날 똑딱 나타난 아이돌같은 녀석이 아니라 ASP.NET 이라는 엄청난 스폰서(?)가 뒤에서 후원 해주고 있으며 마이크로소프트가 차세대 웹 개발 기술의 한 축으로 엄청나게 밀고 있는 녀석이라는 것 입니다.

WebMatrix 에서 제공하는 Request 처리 요청 로깅을 이용한 디버깅

이거는 이 WebMatrix 개발 도구의 기능인데요. 개발하는 과정에서 로깅 기능을 이용해 발생하는 오류를 확인 / 상세한 오류 해결을 위한 가이드도 제공이 가능합니다.



이렇게, WebMatrix 왼쪽 하단의 Site 섹션을 선택하고, 왼쪽 위의 Requests 를 선택하고 상단의 “Capture Request 가 선택” 되어 있다면-기본선택 개발간 발생한 요청 처리를 자동 기록해 확인 가능합니다. 만약 오류가 발생한 게 있다면 해당 요청을 클릭하세요. 바로 상세 정보와 오류의 해결 방안에 대한 추천 가이드까지 제공 받을 수 있습니다. 와우~

예전 같으면, 개발 환경일지라도, IIS 웹로그 열어 죽어라 파거나 오류 보고 루틴을 직접 만들어 이용하셨을 겁니다. Razor에서는 그럴 필요 없어요~ 넵~ 나름 훌륭합니다. 개발 환경에서 이용하기에는 아주 딱이실 겁니다.

출력 값 처리 디버깅(변수 값 찍어보기)

기존 ASP 나 PHP 의 디버깅 방식인 출력 값 처리 디버깅을 물론 해 보실 수 있습니다. 예를 들어, 이런 형태입니다.

```
@{
    var weekday = DateTime.Now.DayOfWeek;

    if(weekday == "Saturday") {
        <p>"토요일"</p>
    }
    else {
        <p>"아마 일요일"</p>
    }
}
```

```
}  
}
```

debug 폴더 하위에 output.cshtml 파일로 저장하고 실행했습니다. 아마도, if 문에서 형변환(Casting) 관련 오류가 발생했을 겁니다.

이렇게 수정하면서 디버깅을 해 보시죠.

```
@{  
    var weekday = DateTime.Now.DayOfWeek;  
    @weekday;  
    /*  
    if(weekday == "Saturday") {  
        <p>"토요일"</p>  
    }  
    else {  
        <p>"아마 일요일"</p>  
    }  
    */  
}
```

아마도, 이런 식으로 @weekday 를 찍어보는 처리를 하실 수 있을 겁니다. 이런 방식을 찍어보는, 출력 디버깅 방법이라고 보통 이야기 합니다. 타입 형변환 관련 이슈이기 때문에 weekday 개체가 어떤 타입이길래? 이런 정보도 궁금 하실 겁니다. 이럴때 “ObjectInfo” Helper 를 이용 가능합니다.

```
@{  
    var weekday = DateTime.Now.DayOfWeek;  
    @ObjectInfo.Print(weekday);  
    /*  
    if(weekday == "Saturday") {  
        <p>"토요일"</p>  
    }  
    else {  
        <p>"아마 일요일"</p>  
    }  
    */  
}
```

```
}
```

찍어보니 weekday 는 "DayOfWeek" 형이네요. String 과 비교하려 했으니 안됩니다. 따라서, String 타입으로 형변환 하면 되겠지요.

```
@{  
    var weekday = DateTime.Now.DayOfWeek;  
    //@ObjectInfo.Print(weekday);  
    if(weekday.ToString() == "Saturday") {  
        <p>"토요일"</p>  
    }  
    else {  
        <p>"아마 일요일"</p>  
    }  
}
```

이렇게 ToString 으로 형변환해 처리가 잘 되는 것을 확인 가능합니다. 이런 출력 디버깅 방식도 가능합니다.

참고로, 웹개발 경험이 많으신 분들만을 위한 부가 설명...

웹개발 경력자 분이시라면, Break point(중단점)을 걸고 단계별로 따라가면서 오류를 확인하고 싶으실 경우에는 무료로 제공되는 Visual Web Developer Express(무료 웹 개발 버전의 비주얼 스튜디오) 나 Visual Studio 이용해 개발하시면 가능합니다. 이 강좌는 WebMatrix 와 Razor 강좌로 Visual Studio 디버깅은 보여드리기 좀 그렇네요. 하지만, 나중에 따로, WebMatrix & Razor 강좌 마무리 후에 준비해 보여 드리도록 할게요. ^_^ - 아 알고 있으시지요? Visual Studio 에서도 ASP.NET MVC3 부터 Razor 개발이 가능합니다.

참고자료 : [Introducing ASP.NET MVC 3 \(Preview 1\)](#) - 영문

위 내용은 본 강좌와 무관하니, 그냥 그러려니 하고 넘어가셔도 됩니다. Razor 를 위해 MVC 나 ASP.NET 을 모두 이해하실 필요는 없습니다.

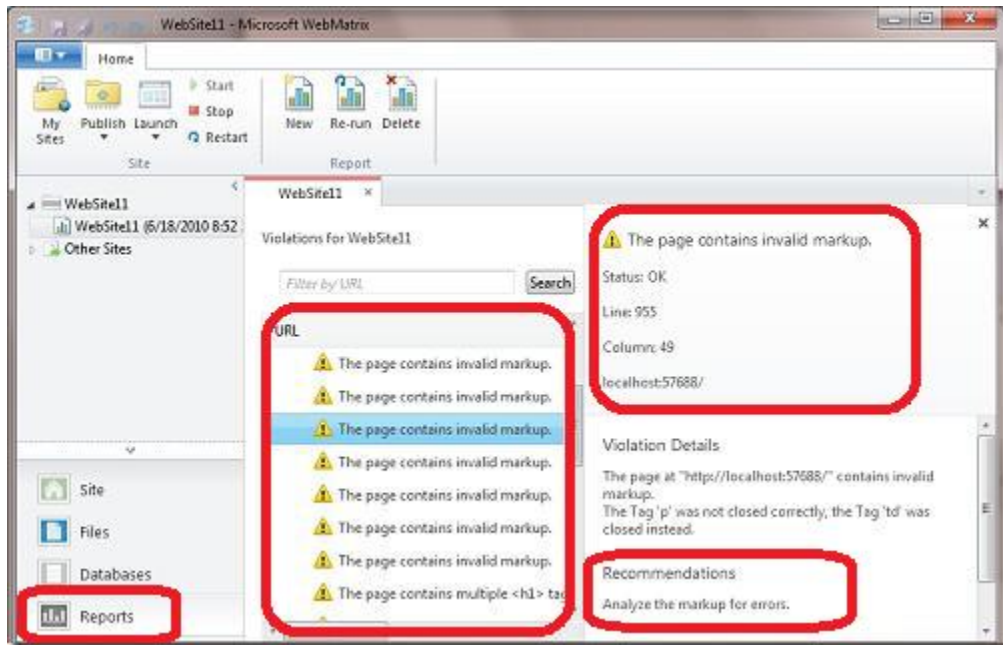
WebMatrix 에 기본 포함된 SEO 리포트 툴을 이용해 검색엔진과 맞지 않는 SEO 관련 오류를 쉽게 처리 가능합니다.

예전에 WebMatrix 소개 동영상 강좌에서 언급해 드렸습시다만, 요즘은 웹사이트에 대한 사용자 유입에 검색엔진 만한 게 없지요. 이 검색엔진에 의해 잘 검색되도록 SEO 관련 처리도 많이 하고 계실 겁니다. 이 SEO 최적화를 검사 하는 게 만만치 않고 고가의 유료 서비스 등을 이용해야 할 수 있는데요. IIS7 과 WebMatrix 에서는 이런 SEO 최적화 기능을 제공하고 있습니다.

참고자료 :

[\[동영상 강좌\] \(2\) WebMatrix 5 분 리뷰~](#)

[IIS7 의 SEO Toolkit 소개](#)



왼쪽 아래 Reports 를 선택하시고, 웹사이트를 선택하시면 이렇게 자동으로 리포트를 만들고, SEO 최적화를 위한 상세한 가이드라인까지 알려 줍니다. - 훌륭하죠~ 이러한 SEO 관련 작업도 WebMatrix 와 내장해서 제공하고 있습니다.

자~ 이렇게 해서 WebMatrix 와 Razor 가 제공하는 디버깅 방안과 접근 절차에 대해서 소개해 드렸습니다. 기존 웹 개발 경험이 있는 분들은 쉽게 감이 오실 것 같아요~ 개인적인 소견으로 Razor 는 어떤 면에서는 PHP 의 디버깅 장점과 ASP.NET 의 디버깅 장점들을 모두 가지고 있다고 느껴지네요. ^_^

그럼 다음 강좌는 마지막 WebMatrix & Razor 하는 강좌 (11) Razor 강좌 - 캐시 처리 에서 뵈도록 하겠습니다. 감사합니다.

^_^

참고자료 :

[IIS7 의 SEO Toolkit 소개](#)

[Introducing ASP.NET MVC 3 \(Preview 1\)](#)

(11) Razor 강좌 - 캐시 처리

지난 시간에는 “(10) Razor 강좌 - 디버깅” 강좌를 진행 했습니다.

이번 시간에는 바로 이어서, “(11) Razor 강좌 - 캐시 처리 ”강좌를 진행 하도록 하겠습니다.

웹 개발에서 캐시(Cache)란? -캐시 소개

웹 개발에서 캐시를 간단히 소개해 드리자면, 엄청나게 많은 사용자가 몰리는 웹사이트에서 매번 웹 페이지 정보를 데이터베이스나 외부의 서버와 연동해 동적으로 생성해 사용자에게 제공할 경우 웹 서버의 성능이 아무리 좋아도 사용자가 몰리게 되면 성능이 저하될 수밖에 없습니다.

캐시는 이렇게 대규모 트래픽이 몰리는 웹사이트에서 웹 페이지를 웹서버의 메모리에 일정시간, 또는 특정 조건으로 저장하고 있다가 재사용 하는 기술입니다.

예를 들어, 나의 웹사이트에서 날씨 정보를 보여주는 경우를 생각해 보세요. 날씨 정보가 기상청 등에서 전달되지만, 매분, 매초마다 새로운 정보가 오는 건 아닐 거예요. 아마도, 대략, 한두 시간 정도에 한번 정도 정보가 오겠죠. 이 정보를 매번 동적으로 우리 웹사이트에서 제공하는 것보다 캐시로 한 시간에 한번만 동적으로 정보를 만들고, 한 시간 동안은 캐시된 정보를 제공해도 문제 없으실 거예요.

어떠세요? 캐시 기능 좋아 보이시죠?

하지만, 캐시 기능을 잘 못 사용할 경우에는 더 큰 문제가 발생할 수 도 있습니다.

캐시 사용시 주의사항

예를 들어, 초단위로 정보 신속성을 주로 제공해야 하는 증권 시세 정보 제공이나 속보 뉴스를 제공해야 하는 포털의 뉴스 속보 섹션 같은 경우에는 이런 캐시 기능을 이용해 “한 시간 동안 캐싱 해라~” 이렇게 설정한다면 문제가 될 겁니다. 정보의 성격과 서비스 성격에 따라 캐시 구성을 달리 하셔야 합니다.

이제 대략 캐시에 대해 감이 잡히셨는지요?

이렇게 캐시는 웹개발에서 트래픽이 몰리는 대규모 웹사이트에 필수적인 기능이고, 캐시를 사용해도 괜찮을지 결정 여부가 매우 중요한 포인트 입니다. 그렇다면 Razor 와 캐시가 어떤 관계가 있을까요?

Razor 의 캐시 지원 - WebCache Helper

Razor 는 가장 최신의 웹 개발 방식으로 페이지에 대해 손쉽게 이용 가능한 캐시 기능을 제공합니다. 이는 WebCache Helper 를 통해 구현 가능하며 아래의 예제를 통해 살펴 보도록 하겠습니다.

```

@{
    var cacheltemKey = "Time";
    var cacheHit = true;
    var time = WebCache.Get(cacheltemKey);
    if (time == null) {
        cacheHit = false;
    }
    if (cacheHit == false) {
        time = DateTime.Now;
        WebCache.Set(cacheltemKey, time, 1, false); //1 분 동안 캐시
    }
}
<!DOCTYPE html>
<html>
    <head>
        <title>WebCache Helper 예제</title>
    </head>
    <body>
        <div>
            @if (cacheHit) {
                @:캐시에서 정보를 찾았습니다.
            }
            else {
                @:캐시에서 정보를 찾지 못했습니다.
            }
        </div>
        <div>
            이 페이지는 @time 에 캐시 되었습니다.
        </div>
    </body>
</html>

```

웹사이트 최상위에 Cache 라는 폴더를 만들고 cache.cshtml 파일로 저장해 실행 했습니다.

WebCache 는 캐시 설정을 돕는 Helper 입니다.

시간을 1 분으로 구성해 캐시 되도록 구성 했기 때문에 처음 요청하면 캐시가 잡히고, 1 분간은 캐시된 페이지를 보시게

될겁니다. 이어서, 1 분이 지나 한번 더 리프레시 하시면 캐시가 만료(Expire)되었기 때문에 새로운 페이지를 보시게 될겁니다.

Razor 는 우리 웹 개발자에게 손쉬운 페이지 캐싱 제어를 위해 이런 WebCache Helper 캐시 기능을 사용하기 간편하게 제공하고 있습니다.

Razor & WebMatrix 기본 시리즈 강좌를 마치면서

짧은 시간이지만, 시리즈 Razor 기본 강좌를 제공해 보았습니다.

강좌를 하면서 받은 느낌은 참 웹 개발자를 위한 깔끔한 작한 기능 들을 제공하는 녀석이구나... 생각이 들었어요.

Razor 를 한마디로 말씀 드리자면, ASP.NET 만큼 안정적이고 빠르며 PHP 만큼 개발하기 쉽다고 할까요.

다음으로는 컬럼이나 팁 강좌 형태로 부족했던 내용을 보충해 드리려고 생각하고 있습니다. ASP 나 ASP.NET 그리고 여건이 된다면 PHP 개발자 분들을 위한 Razor 강좌도 생각해 보고 있구요. 다음 강좌도 기대해 주세요~

앞으로도 계속 Razor 와 WebMatrix 의 발전이 기대됩니다. ^_^

우리 모두 화이팅!

(12) Razor 강좌 – SMTP 메일전송(Live 메일과 Gmail 지원. SSL 지원)

개인적으로 궁금하기도 했고, 나름 필요한게 메일 서비스인지라 추가 강좌 하나 더 올려 봅니다.

Razor 에서 SMTP 메일 전송

참고로, Auth SMTP(인증 SMTP)를 이용 가능하고 SSL 을 이용하는 Live 메일(MSN 메일) 이나 Gmail 도 이용 가능합니다. 저는 Live mail 로 해 보도록 할게요. 이 강좌를 보시는 개발자 분들이라면, Live 메신저(MSN 메신저) ID 는 하나씩 있으실 거예요.

```
@{
    //Live 메일 테스트 입니다. (gmail 도 됩니다.)
    //시작 하기 전에 Live ID(MSN 메신저 ID)로 메일 접속을 테스트 해 보세요.(계정 비활 가능성)
    //Live 메일 테스트는 http://mail.live.com 에서 로그인 해 보세요.
    //gmail 테스트는 http://gmail.com 에서 로그인 해 보세요.

    try {
        // Mail Helper 사용
        //SMTP 서버 주소 – Live 메일이나 gmail 도 됩니다. gmail 의 경우 smtp.gmail.com
        Mail.SmtpServer = "smtp.live.com";

        //SMTP 서버 포트(Live 메일과 gmail 모두 같음. gmail 공식 문서로 SSL 은 465 인데 여하간 587 도 됩니다.
        //아래 참고자료)
        //대부분의 ISP 가 스팸 방지 목적으로 25 번 포트를 막기 때문에 요즘의 SMTP 제공자는 SSL 이나 587 를
        //기본 지원합니다.
        Mail.SmtpPort = 587;

        // SSL 사용 여부(Live 메일과 gmail 모두 같음)
        Mail.EnableSsl = true;

        //라이브(MSN) 메신저 주소를 넣으세요. Live ID 는 email 주소 입니다.
        //gmail 은 gmail ID 메일 주소
        Mail.UserName = "Live ID 를 넣으세요";

        //라이브(MSN) 메신저 주소를 넣으세요. Live ID 는 email 주소 입니다.
        //gmail 은 gmail ID 메일 주소
```

```

Mail.From = "보내는-메일-주소 - Live ID(라이브(MSN) 메신저 ID 를 넣으세요.)";

Mail.Password = "Live-ID 의-암호";

// 메일 전송
Mail.Send(
    to: "받는-사람-메일-주소", //받는 사람 메일 주소 - 자신에게 보내도 됨.
    subject: "SQLER 의 Razor 메일 전송 강좌",
    body: "메일 전송 성공"
);
<text>
    <b>이메일 전송 성공</b>
    메일이 전달 되었습니다.
</text>
}
catch (Exception ex) {
    <text>
        <b>이메일 전송 실패</b>
        아마도 메일 관련 구성이 잘못된 것 같습니다. 다시 확인해 보세요. <br />
        오류 메시지<br />
        @ex.ToString();
    </text>
}
}

```

웹사이트 최 상위에 Mail 폴더를 만들고 MailProcess.cshtml 로 만들고 실행 했습니다.

Live Mail 과 Gmail 모두 SSL - SMTP 를 지원합니다. Razor 는 기본적으로 이를 지원하구요. 가능한 예제 코드에 주석으로 설명해 두었으니, 복사해 실행해 보시면 도움 되실 겁니다.

여담으로...

요즘은 추세가, 호스팅사나 자체적으로 SMTP 를 메일 서버를 운영하기 보다는 이런 서비스의 무료 SMTP 를 이용하는게 추세인 듯 합니다. 특히, Windows Live 관리센터나 google apps 를 이용하면 우리 회사나 커뮤니티의 도메인을 이용하는 메일 주소로 메일 서비스 전체 이용이 가능하지요. 윈도우 라이브 메일을 이용하면, 웹기반이 아니라 클라이언트 어플리케이션으로 동작해 오프라인에서 메일도 볼 수 있고, 다양한 편의 기능을 활용 가능합니다.

모두 무료로, 우리 회사의 도메인을 사용하는 메일 주소를 무료로 직원에게 그냥 생성 가능하고, SMTP 도 지원하며,

*무료*로 다양한 기능의 메일 클라이언트 어플리케이션도 지원해 편리합니다.~

[Windows Live 관리 센터\(구. Custom Domains\)가 3 번째 물결을 탔습니다](#)

[윈도우 라이브 메일](#)

Razor 는 기본적으로 Mail Helper 로 이런 SSL SMTP 를 지원하기 때문에 이런 메일 서비스들과 연동하기도 쉬울 것 같아요.

감사합니다.

SMTP 구성 관련 참고

[라이브 메일\(Hotmail\) SMTP 관련 정보](#) - 영문

[GMail SMTP 관련 정보](#) - 영문

(13) Razor 강좌 - 웹사이트 전체, 또는 폴더 내 파일 요청 시 항상 실행 되는 모듈

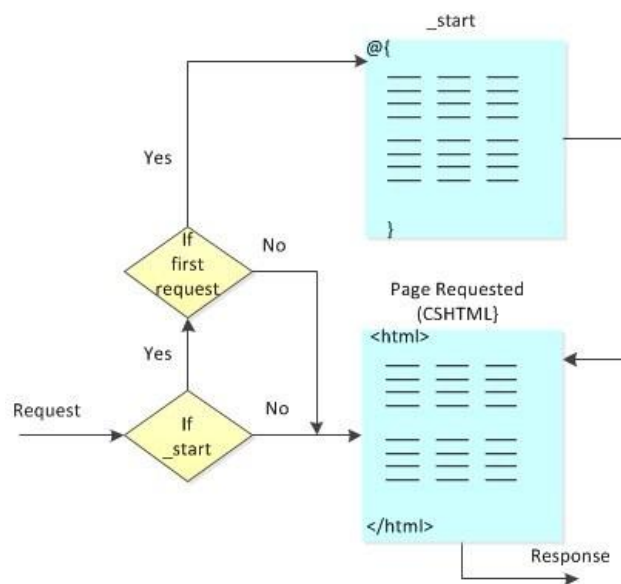
쓰다 보니 강좌가 조금씩 늘어나네요. 나름 꼭 필요할 내용으로 예상되어 두 개의 강좌를 추가 했습니다.

웹사이트를 개발하다 보면, 모든 페이지 요청에 대해 처리해야 하는 공통 모듈이 반드시 있습니다. 예를 들어, “데이터베이스 연결 문자열”과 같은 “전역변수”와 유사한 정보들을 처리해야 할 경우가 있습니다. 또는, 여러 처리 루틴을 폴더 별로 정의해 쓰기 원하실 경우에는 “폴더마다” 폴더 하위의 파일 실행 시마다 동작되길 원하는 처리가 있을 것입니다.

Razor에서는 웹사이트 전체 페이지 요청 시 동작하는 모듈 - “_start.cshtml”과 폴더 별로 요청 시 항상 처리되는 모듈인 “_inti.cshtml”이 존재합니다.

웹사이트 전체 아무 페이지나 요청 시 동작하는 모듈 “_start.cshtml”

강좌 내용 진행을 위해 꼭 아래 흐름을 이해 하시면 좋을 것 같아요.



사용자 요청이 들어올 때 만약 “_start.cshtml”이 존재하고, 처음 요청이 경우 실행하고, 이어서 요청된 페이지가 처리됩니다. “_start.cshtml” 파일이 존재하지 않거나, 처음 요청이 아닐 경우에는 실행되지 않습니다. (꼭 cshtml 만 가능한 건 아닙니다. vbhtml 도 당연히 됩니다. 편의상 cshtml 로 적었습니다.^_;;)

주의

당연히, 모든 페이지 요청 시 실행되는 모듈이 _start.cshtml 파일에 오류가 있으면 전체 웹사이트 실행에서 오류가 발생할 수 있습니다. 사용에 주의 하세요.

참고

페이지 파일 앞에 “_” 즉, 언더바가 있을 경우에 이 파일은 사용자의 직접적인 요청에 의해 실행되지 않습니다. 다른 파일 요청에 의한 참조로만 동작하게 됩니다.

그렇다면, 예제를 통해 살펴 보도록 하겠습니다.

```
@{
    AppData["customAppName"] = "SQLER 웹사이트";
}
```

현재 웹사이트의 루트에 _start.cshtml 파일로 만들고 저장합니다.

```
@{
    var appName = ApplicationInstance.Application["customAppName"];
}

<!DOCTYPE html>
<html>
    <head>
        <title>웹사이트의 이름을 보여 주세요~</title>
    </head>
    <body>
        <h1>@appName</h1>
    </body>
</html>
```

웹사이트의 루트에 AppName.cshtml 파일로 저장하고 실행합니다.

실행해보니 감이 오시죠? 제가 실행한 건 AppName.cshtml 파일인데 “_start.cshtml” 페이지에 전역으로 구성된 변수인 “customAppName”의 값을 불러와 개별 페이지에서 이용 가능합니다. 이렇게, 전역변수로 사용 가능한 처리를 이용하시면 유용합니다.

조금 실전적인(?) 전역변수 처리 예제를 소개해 드리려고 해요. 또 하나의 Helper 인 reCAPTCHA Helper 를 이용하는 예제를 살펴 보도록 하겠습니다.(캡차는 악의적인 봇을 걸러내 스팸 등을 사이트에서 실행하지 못하도록 사람인지 검사하는 기능입니다. 아마 사이트 댓글 쓰기나 포털 카페 가입 시 나오는 숫자나 단어를 넣으라는 처리 보셨을 거예요. 그게 캡차입니다.)

국내에서는 좀 뜸하지만, 해외에서 기본 캡차로 많이 쓰이는 reCAPTCHA 를 이용해 볼게요. Helper 하나 더 공부하는 셈~ 치시면 될 겁니다. 사용을 위해 등록을 해야 합니다. 사이트에 접속합니다. <http://recaptcha.net> – 웹사이트에 대해 계정을 생성(구글 계정 이용)하시면 publicKey 와 PrivateKey 문자열이 나오게 됩니다. 이 문자열을 복사해 이용하시면 됩니다. (참고로, Razor 의 Helper 는 Helper 는 누구나 추가로 만들어 사용이 가능합니다.)

reCAPTCHA helper 사용에 필요한 문자열을 _start.cshtml 파일에 저장하고 사용하는 예제 입니다.

```
@{
    // 국내외에서 캡차로 많이 쓰이는 reCAPTCHA 사이트에 접속합니다.
    // http://recaptcha.net – 웹사이트에 대해 계정을 생성(구글 계정 이용)하시면
    // publicKey 와 PrivateKey 문자열이 나오게 됩니다.
    //이 문자열을 복사해 이용하시면 됩니다.

    ReCaptcha.PublicKey = "PublicKey 문자열을 복사해 넣으세요.";
    ReCaptcha.PrivateKey = "PrivateKey 문자열을 복사해 넣으세요.";
```

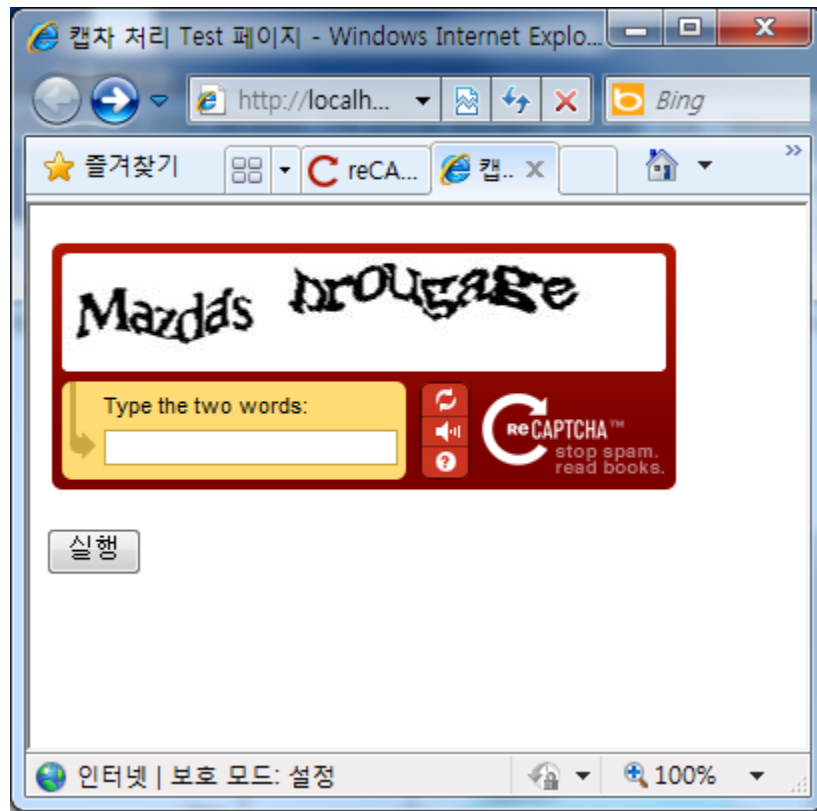
```
}
```

루트 폴더에 위치한 _start.cshtml 파일을 위처럼 캡차 수행해 필요한 변수로 수정합니다.

```
@{
    var showRecaptcha = true;
    if (IsPost) {
        if (ReCaptcha.Validate()) {
            @:통과! 사람이군요!!!
            showRecaptcha = false;
        }
        else{
            @:너 봇이지?!
        }
    }
}

<!DOCTYPE html>
<html>
    <head>
        <title>캡차 처리 Test 페이지 - 단어를 정확히 입력하세요.</title>
    </head>
    <body>
        <form action="" method="post">
            @if(showRecaptcha == true){
                if(ReCaptcha.PrivateKey != ""){
                    <p>@ReCaptcha.GetHtml()</p>
                }
                else {
                    <p>아마도 인증키가 틀리거나 오류가 있는 것 같습니다. 다시 확인해 보세요. </p>
                }
            }
            <div>
                <input type="submit" value="실행" />
            </div>
        </form>
    </body>
</html>
```

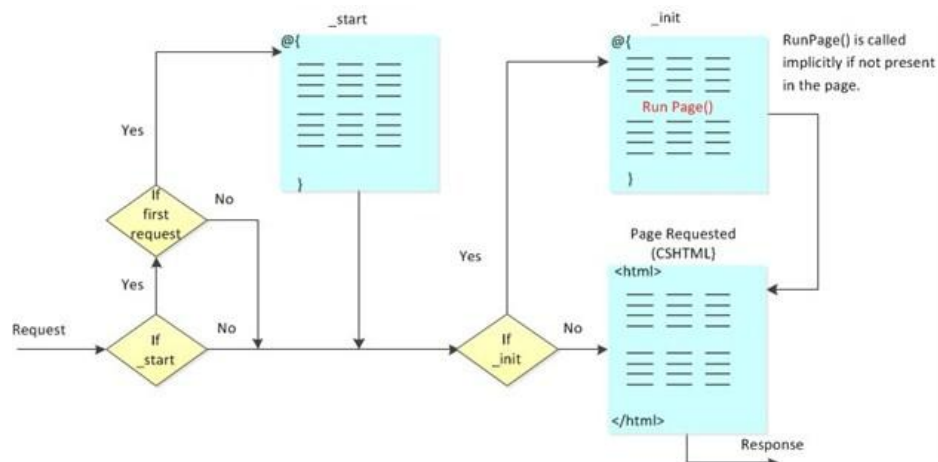
Recaptcha.cshtml 파일로 저장하고 실행합니다.



단어를 넣고 처리가 되면 통과하게 됩니다. - 이렇게 간단히 _start.cshtml 를 이용한 전역변수 처리를 알아 보았습니다. 다음은 웹사이트 전체가 아니라, 폴더 내 파일들에 대한 처리를 알아보도록 할게요.

"폴더" 내 파일 요청 시 항상 실행 되는 모듈 - "_init.cshtml"

전체 웹사이트에 대해 처리되는 공통모듈인 _start 와 달리 "폴더" 하위에 위치하는 페이지 파일들에 대해서만 처리될 필요가 있는 경우가 있습니다. 이때 사용하는 처리가 "_init.cshtml"입니다. 웹 개발 경험이 있는 분들은 바로 감이 딱 오실 거예요. 이 녀석이 사용되는 대표적인 경우도 느낌이 오시죠? 예~ 바로 사용자 정의 에러처리 일거예요. 흐름은 어렵지 않습니다. 아래를 꼭 이해해 주세요.



_start 처리 모듈은 위에서 설명해 드린 것과 같습니다. 이어서, 폴더 안에 _init.cshtml 파일이 위치하게 되면 해당 파일을 실행하고 실행 후에 사용자가 요청한 페이지가 실행되게 됩니다.

_start 와 달리 _init 에서는 “RunPage()”를 실행해 요청될 페이지를 그 위치에서 실행하게 됩니다. – 왜 이런 경우가 있는고 하니, 개발 경험이 있는 분이라면 에러 핸들 시 try 에서 시도할 경우 입니다. 감이 팍팍 오시죠? 그럼 간단한 예제를 통해 살펴 보도록 하시지요.

```
@{
    PageData["Color1"] = "빨강색";
    PageData["Color2"] = "파랑색";
}
```

웹사이트 루트에 _init.cshtml 파일을 만들고 저장합니다.

```
@{
    PageData["Color2"] = "노랑색";
    PageData["Color3"] = "초록색";
}
```

InitPages 라는 폴더를 만들고 이 폴더에 _init.cshtml 로 저장합니다.

```
@PageData["Color1"]
<br/>
@PageData["Color2"]
<br/>
@PageData["Color3"]
```

InitPages 폴더에 default.cshtml 파일을 만들고 실행합니다.

InitPages 폴더의 페이지를 요청했는데 루트 폴더의 _init 파일과 InitPages 폴더의 _init 파일이 모두 실행됩니다. 그럼 Color2 는 어떻게 나오나요? 계층 처리로 종단에 위치한 값, “노랑색”이 출력됩니다. 감이 오시지요? 그렇다면, 기대하고 있으실 에러 핸들 처리를 해 보도록 할게요.

_init 을 이용한 폴더 내 페이지 실행 시 에러 처리 모듈

```
<!DOCTYPE html>
<html>
    <head>
        <title>오류가 발생했습니다.</title>
    </head>
    <body>
<h1>SQLER 고객님의 불편을 드려 죄송합니다. 조속히 오류를 해결 하겠습니다.</h1>
```

```
<p> 이 페이지에서 오류가 발생 했습니다. @Request["source"]</p>
</body>
</html>
```

웹사이트 루트 폴더에 Error.cshtml 파일로 저장합니다.

```
@{
    try
    {
        RunPage();
    }
    catch (Exception ex)
    {
        Response.Redirect("~/Error.cshtml?source=" +
            HttpUtility.UrlEncode(Request.AppRelativeCurrentExecutionFilePath));
    }
}
```

루트 폴더에 InitCatch 폴더를 만들고 _init.cshtml 파일로 저장합니다.

```
@{
    var db = Database.OpenFile("오류 유발 - 없는 DB 를 열기 위해 시도");
}
```

InitCatch 폴더에 Exception.cshtml 파일로 저장합니다.

이어서, 에러를 발생시키는 Exception.cshtml 파일을 실행하세요.

감이 오시는지요? InitCatch 폴더의 _init 중간 RunPage()가 수행되어 요청된 페이지인 Exception.cshtml 이 실행 됩니다. 오류가 발생하고 오류는 핸들 되어 루트의 error 페이지로 redirect 되며 쿼리문자열로 에러 페이지 정보를 전달하게 됩니다. error.cshtml 페이지는 넘겨받은 에러 페이지 정보를 출력하게 됩니다.

여담으로, 웹 개발자 분들을 위한 참고

이렇게 폴더 내 오류 처리와 같은 모듈을 쉽게 사용가능 한 처리가 _init.cshtml 입니다. 만약, 웹 개발 경험이 많으시고 좀더 다듬어진 error 를 구현하고 싶다면, 적절히 error 핸들 페이지에 catch 된 exception 개체의 에러 정보를 string 으로 변환해 넘겨서 가공 후 사용자에게 보여주고, 이어서 관리자에게 오류에 대해 자동 알림 기능(SMTP 메일이나 SMS 전송)도 가능할 겁니다.

도움 되시길 바라며, 다음 강좌에서 뵙겠습니다.

참고자료 :

<http://www.asp.net/webmatrix/tutorials/15-customizing-site-wide-behavior>
[사용자가 직접 Helper 를 만들고 사용하는 방법](#) - 영문

(14) Razor 강좌 - URL 라우팅(Routing) 으로 SEO 최적화 구현

이번에 소개해 드릴 내용은, Razor 의 URL 라우팅 관련 내용입니다. 아마 블로그나 최신의 오픈 소스 CMS 를 이용해 보신 분들은 이런 SEO 와 Fancy URL 이나 Permanent URL 에 대해서 잘 아실 겁니다. 복잡하고, 프로그래밍 친화적인 URL 보다는 사용자와 검색엔진에 친화적인 URL 을 이용하는 방법을 의미합니다.

특히, 콘텐츠 사이트 트래픽 유입에 가장 큰 기여를 하는 녀석인 검색엔진이 이런 SEO(Search Engine Optimizer)차원에서 URL 에 높은 가중치를 두고 있기 때문에 최근 웹사이트 제작에 기본 요소이기도 합니다.



Razor 는 가장 최신의 웹 개발 방식입니다. 당연히 기본 이런 SEO 제공을 위한 Fancy & Permanent URL 을 이용하는 기능을 기본 제공합니다. 바로 “URL 라우팅” 기능입니다. – 바로 상세히 알아 보도록 하지요.

먼저 예제를 실행해 보도록 하겠습니다.

```
@{
    var FruitName = UrlData[0].ToString();
}

<!DOCTYPE html>
<html>
    <head>
        <title>URL 라우팅</title>
    </head>
    <body>
        @FruitName
    </body>
</html>
```

루트 폴더에 fruit.cshtml 파일을 만들고, 아래 링크와 같이 URL 을 만들어 실행합니다.

<http://localhost:8080/fruit/사과> (링크로 사용하는 서버나 포트가 저와 다를 수 있습니다. 조절해 넣으세요.)

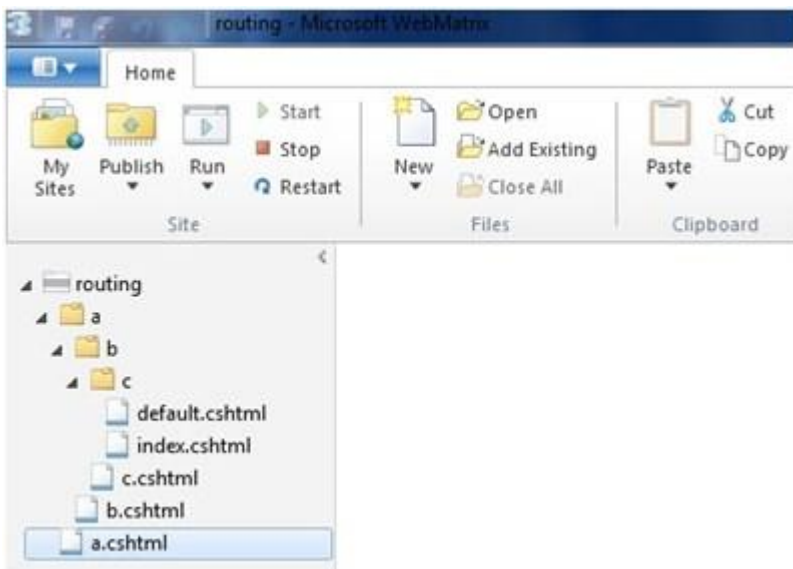
그럼 결과가 보이실 겁니다. "사과"가 잘 찍혀 나오시지요. ^_^

제가 만든 파일은 분명 fruit.cshtml 파일이고, 라우팅을 위한 아무 처리를 안 해도 자동으로 URL 라우팅을 해 줍니다. 이럴때 Razor 참 땡큐하죠. 다른 기술이라면 라이브러리를 만들어 쓰거나, 라우팅을 웹 서버 플러그인 - rewrite 모듈로 구현해야 하는데, 이걸 그냥 쉽게 종속성이나 귀찮은 거 없이 자체 해결해 줍니다. 그리고, 파라미터 값 처리를 위해 UriData 만 쓰면 땡이죠.

그런데, 만약, 진짜 <http://localhost:8080/fruit/사과/default.cshtml> 과 같은 파일이 존재한다면 어떻게 될까요? UriData 가 어떻게 도움을 주는 건지?

그렇다면, 이런 자동 라우팅이 어떤 우선 순위에 의해 동작하는지 이해할 필요가 있습니다.

아래 라우팅 우선순위를 참고하세요.



이런 살짝 아리까리한 폴더와 파일들이 위치하고 있다고 가정해 보세요. 폴더 이름과 파일 이름에 주의하셔야 할겁니다.

<http://www.SQLERTEST.com/a/b/c> URL 요청이 들어올 경우에 다음과 같은 순서로 라우팅이 처리합니다. 해당 단계에 맞는 파일이 없다면 이어서 아래 파일을 찾는 절차를 진행하게 됩니다.

- (1) /a.cshtml 파일이 존재할 경우 "b/c" 값이 UriData 로 전달됩니다.
 - (2) /a/b.cshtml 파일이 존재할 경우 "c" 값이 UriData 로 전달됩니다.
 - (3) /a/b/c.cshtml 파일이 존재할 경우 아무 값도 UriData 로 전달되지 않습니다. 그냥 실행만 됩니다.
- cshtml 파일 매칭이 없을 경우 다음 순서로 파일을 찾게 됩니다.
- (4) /a/b/c/default.cshtml 파일을 찾게 됩니다.
 - (5) /a/b/c/index.cshtml 파일을 찾게 됩니다.

이렇게 간단히 URL 라우팅 기능을 알아보았습니다. Razor 의 URL 라우팅 기능, 어떠세요? 깔끔하게 느껴 지시나요? 앞으로도 많은 좋은 내용으로 인사 드리겠습니다.

좋은 하루 되세요~

전문 개발자를 위한 참고

Razor 는 ASP.NET MVC 의 View 엔진이며 MVC 에 사용되는 ASP.NET Routing 을 아무 전처리 없이 이용가능합니다. 여러번 말씀 드리지만, Razor 공부를 위해 ASP.NET 이나 MVC 를 모두 알 필요는 절대로 없습니다. IIS 의 URL Rewrite 와 ASP.NET Routing 에 대한 내용은 예전에 제가 적은 강좌를 참고하세요. ([URL Rewrite 1.1 \(URL 재작성\) - \(4\) ASP.NET 라우팅과 URL Rewrite](#)) 추가적으로, ASP.NET MVC 의 라우팅에 대한 내용은 링크를 참고하세요. ([ASP.NET Routing](#))

참고자료 :

<http://www.asp.net/webmatrix/tutorials/15-customizing-site-wide-behavior>

URL Rewrite 에 대한 SQLER 의 IIS 강좌 링크 : <http://www.sqler.com/bIISLec>