Web Fundamentals and HTTP Protocol

Web Development

Raed Felfel - 2025

Introduction to Web Applications

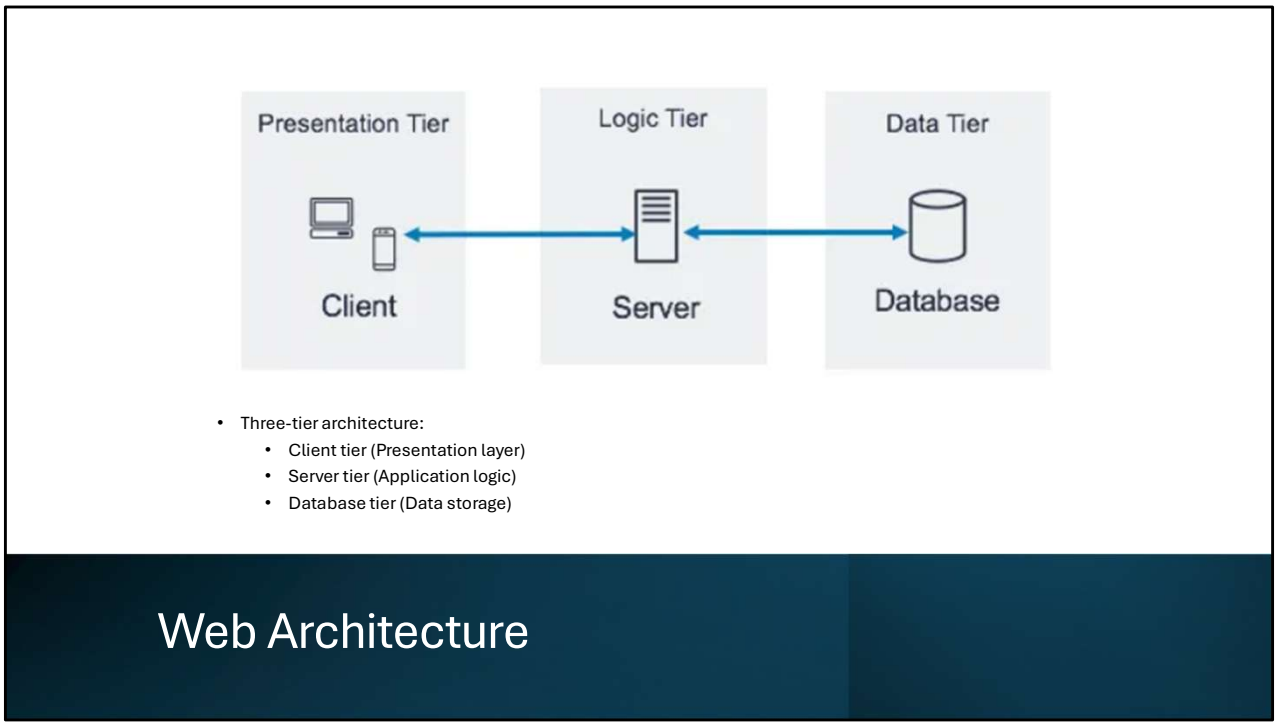
- What is a web application?
- How web applications differ from desktop applications
- Key characteristics:
  - Accessed through web browsers
  - Centralized hosting
  - No installation required
  - Cross-platform compatibility

Welcome to our course on web application development. Before we dive into ASP.NET MVC, let's understand what web applications are. A web application is software that runs on a web server and is accessed through a web browser over the internet or an intranet.
Unlike desktop applications that must be installed on each user's computer, web applications are centrally hosted and require no installation. Users simply navigate to a URL to access the application. This offers several advantages: automatic updates for all users, cross-platform compatibility (works on Windows, Mac, Linux, mobile devices), and accessibility from anywhere with an internet connection.
Think about applications like Gmail, Facebook, or online banking - these are all web applications that you access through your browser rather than installing dedicated software on your device.

مرحبًا بكم في مساقنا حول تطوير تطبيقات الويب. قبل أن نتعمق في ASP.NET MVC، دعونا نفهم ما هي تطبيقات الويب. تطبيق الويب هو برنامج يعمل على خادم ويب ويتم الوصول إليه من خلال متصفح ويب عبر الإنترنت أو الشبكة الداخلية.

على عكس تطبيقات سطح المكتب التي يجب تثبيتها على كل جهاز كمبيوتر للمستخدم، يتم استضافة تطبيقات الويب مركزيًا ولا تتطلب تثبيتًا. يقوم المستخدمون ببساطة بالانتقال إلى عنوان URLللوصول إلى التطبيق. يوفر هذا عدة مزايا: تحديثات تلقائية لجميع المستخدمين، وتوافق عبر المنصات (يعمل على Windowsو Macو Linuxوالأجهزة المحمولة)، وإمكانية الوصول من أي مكان به اتصال بالإنترنت.

فكر في تطبيقات مثل Gmailو Facebookأو الخدمات المصرفية عبر الإنترنت - كلها تطبيقات ويب تصل إليها من خلال متصفحك بدلاً من تثبيت برامج مخصصة على جهازك.



Web applications typically follow a client-server architecture. This model involves three main components:

First, we have the client tier, which is typically a web browser that sends requests to a server and displays responses to the user. This is the presentation layer that the user interacts with.

Second, the server tier contains the application logic. When the web server receives a request, it processes it, executing the application code to generate a response. This is where frameworks like ASP.NET MVC operate.

Third, the database tier stores and manages the application's data. The server communicates with the database to retrieve or update information based on user

requests.

For example, when you log into a social media site, your browser (client) sends a request to the server. The server verifies your credentials against the database, retrieves your feed data, processes it, and sends back HTML, CSS, and JavaScript for your browser to render.

This separation of concerns makes web applications more scalable, maintainable, and secure. Each layer can be optimized, scaled, or replaced independently.

تتبع تطبيقات الويب عادةً بنية العميل والخادم. يتضمن هذا النموذج ثلاثة مكونات رئيسية:

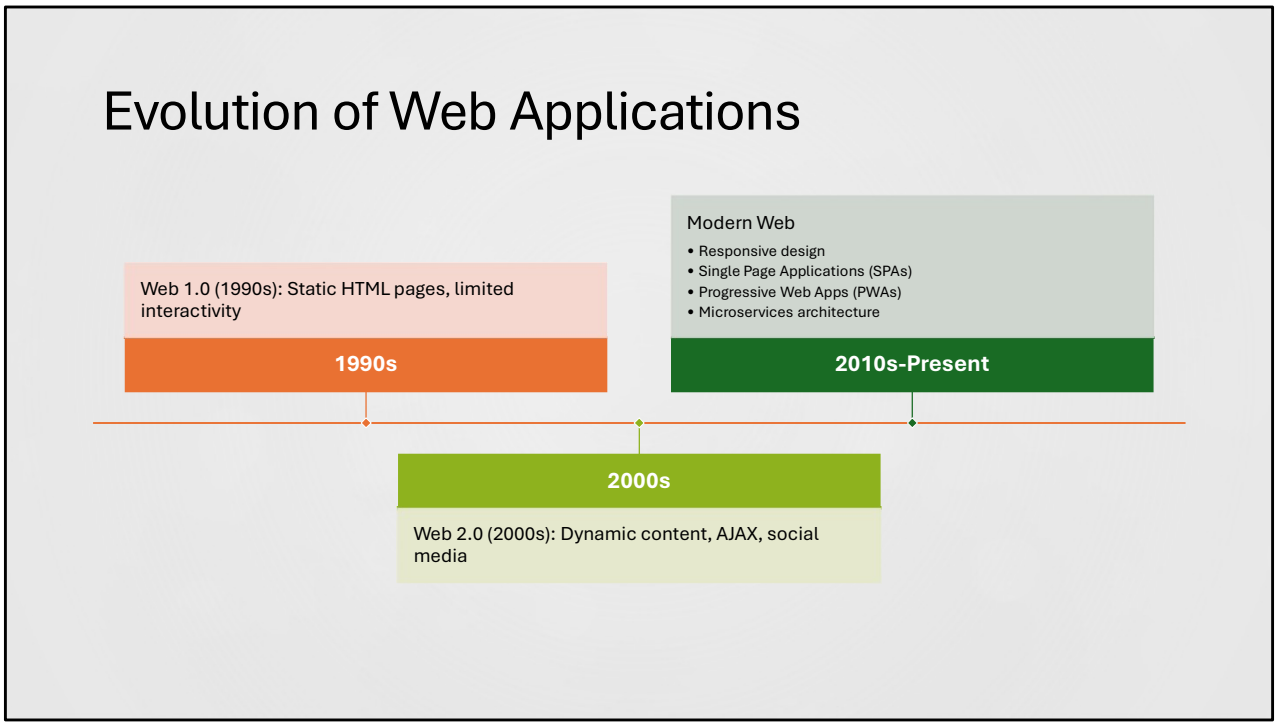
أولاً، لدينا طبقة العميل، وهي عادةً متصفح ويب يرسل طلبات إلى الخادم ويعرض الاستجابات للمستخدم. هذه هي طبقة العرض التي يتفاعل معها المستخدم.

ثانيًا، تحتوي طبقة الخادم على منطق التطبيق. عندما يتلقى خادم الويب طلبًا، يقوم بمعالجته وتنفيذ رمز التطبيق لإنشاء استجابة. هذا هو المكان الذي تعمل فيه أطر عمل مثل .ASP.NET MVC

ثالثًا، طبقة قاعدة البيانات تخزن وتدير بيانات التطبيق. يتواصل الخادم مع قاعدة البيانات لاسترجاع المعلومات أو تحديثها بناءً على طلبات المستخدم.

على سبيل المثال، عندما تسجل الدخول إلى موقع التواصل الاجتماعي، يرسل متصفحك (العميل) طلبًا إلى الخادم. يتحقق الخادم من بيانات اعتمادك مقابل قاعدة البيانات، ويسترجع بيانات التغذية الخاصة بك، ويعالجها، ويرسل مرة أخرى HTMLو CSSو JavaScriptليقوم متصفحك بعرضها.

يجعل هذا الفصل بين المهام تطبيقات الويب أكثر قابلية للتوسع والصيانة والأمان. يمكن تحسين كل طبقة أو توسيع نطاقها أو استبدالها بشكل مستقل.



Web applications have evolved dramatically over the years. In the early days of the web, known as Web 1.0, sites consisted primarily of static HTML pages with minimal interactivity. These sites were essentially digital brochures where information flowed in one direction - from the server to the user.

With Web 2.0 in the 2000s, we saw the emergence of dynamic content and greater user participation. Technologies like AJAX allowed for asynchronous data loading without page refreshes, creating more interactive experiences. This era gave birth to social media platforms, wikis, and blogs where users could contribute content.

Today's modern web applications are characterized by responsive designs that adapt to different screen sizes, from desktops to smartphones. Single Page Applications (SPAs) load a single HTML page and dynamically update content as users interact with the app, providing a more fluid user experience. Progressive Web Apps (PWAs) offer offline functionality and app-like experiences in the browser.

Behind the scenes, many applications now use microservices architecture, where the application is divided into small, independent services rather than built as a monolithic system. This allows for greater scalability and easier maintenance.

ASP.NET MVC, which we'll be studying in this course, fits into this modern web development landscape, providing a structured way to build scalable, maintainable web applications.

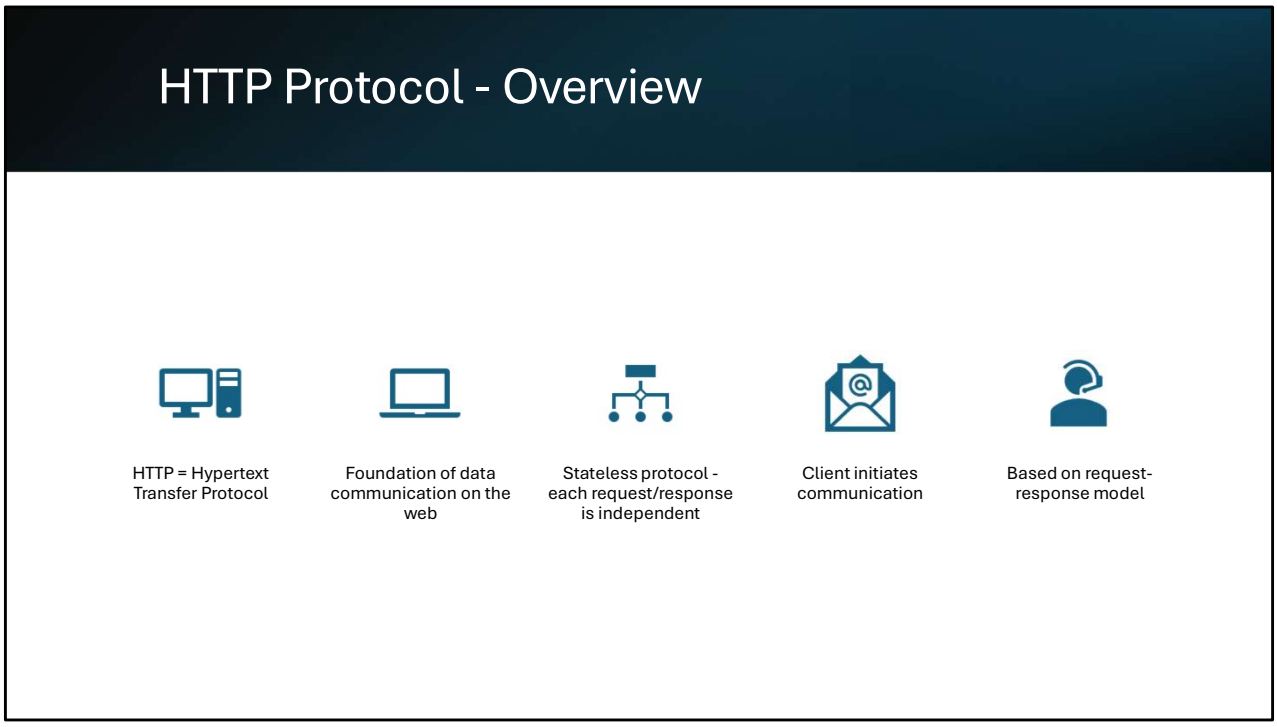
تطورت تطبيقات الويب بشكل كبير على مر السنين. في الأيام الأولى للويب، المعروفة باسم Web 1.0، كانت المواقع تتكون بشكل أساسي من صفحات HTMLثابتة مع الحد الأدنى من التفاعل. كانت هذه المواقع عبارة عن كتيبات رقمية حيث تتدفق المعلومات في اتجاه واحد - من الخادم إلى المستخدم.

مع Web 2.0في الألفينات، شاهدنا ظهور المحتوى الديناميكي ومشاركة المستخدمين بشكل أكبر. سمحت تقنيات مثل AJAXبتحميل البيانات بشكل غير متزامن دون تحديث الصفحة، مما أدى إلى تجارب أكثر تفاعلية. أدى هذا العصر إلى ظهور منصات التواصل الاجتماعي والويكي والمدونات حيث يمكن للمستخدمين المساهمة بالمحتوى.

تتميز تطبيقات الويب الحديثة اليوم بتصميمات متجاوبة تتكيف مع أحجام الشاشات المختلفة، من أجهزة سطح المكتب إلى الهواتف الذكية. تقوم تطبيقات الصفحة الواحدة ((SPAsبتحميل صفحة HTMLواحدة وتحديث المحتوى ديناميكيًا مع تفاعل المستخدمين مع التطبيق، مما يوفر تجربة مستخدم أكثر سلاسة. توفر تطبيقات الويب التقدمية ((PWAsوظائف دون اتصال بالإنترنت وتجارب تشبه التطبيقات في المتصفح.

خلف الكواليس، تستخدم العديد من التطبيقات الآن بنية الخدمات المصغرة، حيث يتم تقسيم التطبيق إلى خدمات صغيرة ومستقلة بدلاً من بنائه كنظام متكامل. يسمح هذا بقابلية أكبر للتوسع وصيانة أسهل.

يناسب ASP.NET MVC، الذي سندرسه في هذه الدورة، مشهد تطوير الويب الحديث هذا، مما يوفر طريقة منظمة لبناء تطبيقات ويب قابلة للتوسع وقابلة للصيانة.



Now let's turn our attention to the HTTP protocol, which is fundamental to understanding web applications. HTTP stands for Hypertext Transfer Protocol, and it's the foundation of data communication on the World Wide Web.

HTTP is a stateless protocol, which means each request-response cycle is completely independent. The server doesn't maintain any memory of previous interactions with the client. This statelessness is important to understand because it creates challenges for maintaining user state across requests, leading to solutions like cookies, session storage, and tokens.

In the HTTP model, communication is always initiated by the client - typically a web browser, but it could also be a mobile app or another service. The client sends a request to a server, and the server responds with the requested data or an error message.

This request-response model is the foundation of all web interactions. Even complex web applications with sophisticated user interfaces ultimately rely on this simple pattern of communication.

Understanding HTTP is crucial for web developers because it affects everything from application design to performance optimization and security implementation.

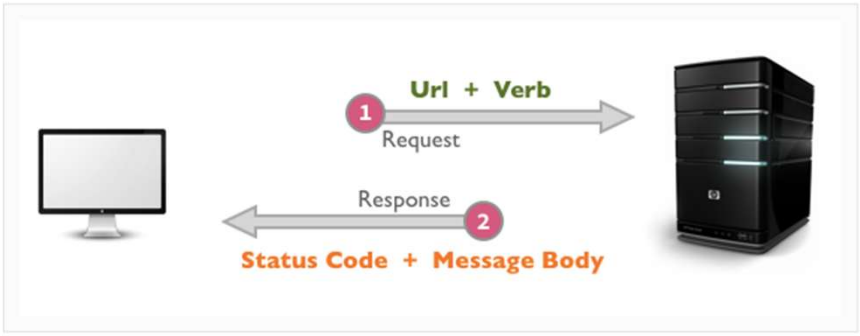
الآن دعونا نوجه انتباهنا إلى بروتوكول HTTP، وهو أمر أساسي لفهم تطبيقات الويب. HTTPهو اختصار لـ Hypertext Transfer Protocol (بروتوكول نقل النص التشعبي)، وهو أساس نقل البيانات على شبكة الويب العالمية.

HTTPهو بروتوكول بلا حالة، مما يعني أن كل دورة طلب واستجابة مستقلة تمامًا. لا يحتفظ الخادم بأي ذاكرة للتفاعلات السابقة مع العميل. من المهم فهم هذه الخاصية لأنها تخلق تحديات للحفاظ على حالة المستخدم عبر الطلبات، مما يؤدي إلى حلول مثل ملفات تعريف الارتباط وتخزين الجلسات والرموز المميزة.

في نموذج HTTP، يتم بدء الاتصال دائمًا بواسطة العميل - عادةً متصفح ويب، ولكن يمكن أن يكون أيضًا تطبيق جوال أو خدمة أخرى. يرسل العميل طلبًا إلى الخادم، ويستجيب الخادم بالبيانات المطلوبة أو رسالة خطأ.

نموذج الطلب والاستجابة هذا هو أساس جميع تفاعلات الويب. حتى تطبيقات الويب المعقدة ذات واجهات المستخدم المتطورة تعتمد في النهاية على هذا النمط البسيط من الاتصال.

فهم HTTPأمر بالغ الأهمية لمطوري الويب لأنه يؤثر على كل شيء من تصميم التطبيق إلى تحسين الأداء وتنفيذ الأمان.



- Client initiates request to server
- Server processes request
- Server returns response
- Client renders or processes response

HTTP Request-Response Cycle

Let's examine the HTTP request-response cycle in detail. This process begins when a user takes an action in their browser, like clicking a link or submitting a form.

First, the client (browser) creates an HTTP request containing information like the URL, method type, headers, and possibly a body with data. This request is sent to the web server.

Second, the server receives this request and processes it. In an ASP.NET MVC application, this typically involves routing the request to the appropriate controller action, which may interact with a database or other services.

Third, once processing is complete, the server generates an HTTP response. This includes a status code indicating success or failure, headers with metadata, and the response body containing the requested data - often HTML, but also possibly JSON,

images, or other formats.

Finally, the client receives this response and processes it accordingly. For HTML responses, the browser renders the page. For API responses, the data might be processed by JavaScript code.

This entire cycle happens with every interaction in a web application. Even actions that seem instantaneous to users, like clicking a button to show a dropdown menu, may involve this full request-response cycle, although modern applications often use JavaScript to handle some interactions client-side without a round trip to the server.

دعونا نفحص دورة طلب واستجابة HTTPبالتفصيل. تبدأ هذه العملية عندما يتخذ المستخدم إجراءً في متصفحه، مثل النقر على رابط أو إرسال نموذج.

أولاً، ينشئ العميل (المتصفح) طلب HTTPيحتوي على معلومات مثل عنوان URLونوع الطريقة والرؤوس، وربما نص مع البيانات. يتم إرسال هذا الطلب إلى خادم الويب.

ثانيًا، يستقبل الخادم هذا الطلب ويعالجه. في تطبيق ASP.NET MVC، يتضمن هذا عادةً توجيه الطلب إلى إجراء وحدة التحكم المناسبة، والذي قد يتفاعل مع قاعدة بيانات أو خدمات أخرى.

ثالثًا، بمجرد اكتمال المعالجة، ينشئ الخادم استجابة HTTP. يتضمن ذلك رمز حالة يشير إلى النجاح أو الفشل، ورؤوس بها بيانات وصفية، ونص الاستجابة الذي يحتوي على البيانات المطلوبة - غالبًا HTML، ولكن أيضًا قد يكون JSONأو الصور أو تنسيقات أخرى.

أخيرًا، يتلقى العميل هذه الاستجابة ويعالجها وفقًا لذلك. بالنسبة لاستجابات HTML، يقوم المتصفح بعرض الصفحة. بالنسبة لاستجابات API، قد تتم معالجة البيانات بواسطة رمز JavaScript.

تحدث هذه الدورة الكاملة مع كل تفاعل في تطبيق الويب. حتى الإجراءات التي تبدو فورية للمستخدمين، مثل

النقر على زر لإظهار قائمة منسدلة، قد تتضمن دورة الطلب والاستجابة الكاملة هذه، على الرغم من أن التطبيقات الحديثة غالبًا ما تستخدم JavaScriptللتعامل مع بعض التفاعلات من جانب العميل دون رحلة ذهاب وإياب إلى الخادم.

HTTP Methods

Method	Purpose	Example
GET	Request data	Retrieving a webpage
POST	Submit data	Submitting a form
PUT	Update existing resource	Updating user profile
DELETE	Remove a resource	Deleting an account
PATCH	Partial update	Changing one field
HEAD	Get headers only	Checking if resource exists
OPTIONS	Available communications options	CORS preflight

HTTP defines several methods, sometimes called "verbs," that indicate the desired action to be performed on a resource. Let's look at the most important ones:

GET is the most common method, used to request data from a server. When you type a URL in your browser or click a link, a GET request is sent. GET requests should only retrieve data and not modify any server state. Parameters are included in the URL and are visible to users.

POST is used to submit data to be processed. It's commonly used for form submissions that create new records. Unlike GET, POST data is included in the request body, not the URL, so it's not visible in the browser address bar and has no size limitations.

PUT is used to update an existing resource. It replaces the entire resource with the

data sent in the request. For example, updating a complete user profile would use PUT.

DELETE, as the name suggests, removes the specified resource. It's used when you want to delete data from the server.

There are other methods like PATCH for partial updates, HEAD for retrieving just the headers without the body, and OPTIONS which returns the HTTP methods supported by the server.

In ASP.NET MVC, these HTTP methods are mapped to controller action methods using attributes like [HttpGet], [HttpPost], etc., which we'll see in more detail later in the course.

يحدد HTTPعدة طرق، تسمى أحيانًا "أفعال"، تشير إلى الإجراء المطلوب تنفيذه على مورد. دعونا ننظر إلى أهمها:

GETهي الطريقة الأكثر شيوعًا، وتستخدم لطلب البيانات من الخادم. عندما تكتب عنوان URLفي المتصفح أو تنقر على رابط، يتم إرسال طلب .GETيجب أن تسترجع طلبات GETالبيانات فقط ولا تعدل أي حالة للخادم. يتم تضمين المعلمات في عنوان URLوتكون مرئية للمستخدمين.

POSTتستخدم لإرسال البيانات ليتم معالجتها. يتم استخدامها بشكل شائع لتقديمات النماذج التي تنشئ سجلات جديدة. على عكس GET، يتم تضمين بيانات POSTفي نص الطلب، وليس في عنوان URL، لذا فهي غير مرئية في شريط عنوان المتصفح وليس لها قيود على الحجم.

PUTتستخدم لتحديث مورد موجود. تستبدل المورد بأكمله بالبيانات المرسلة في الطلب. على سبيل المثال، سيتم استخدام PUTلتحديث ملف تعريف مستخدم كامل.

DELETE، كما يوحي الاسم، تزيل المورد المحدد. يتم استخدامها عندما تريد حذف البيانات من الخادم.

هناك طرق أخرى مثل PATCH للتحديثات الجزئية، و HEAD لاسترجاع الرؤوس فقط بدون النص، و OPTIONS التي تعيد طرق HTTP المدعومة من قبل الخادم.

في ASP.NET MVC، يتم ربط طرق HTTP هذه بطرق إجراء وحدة التحكم باستخدام سمات مثل [[HttpGet و[[HttpPost، وما إلى ذلك، والتي سنراها بمزيد من التفصيل لاحقًا في الدورة.

HTTP Status Codes

Range	Meaning and example
1xx - Informational	100 means Continue
2xx - Success	200 OK 201 Created 204 No Content
3xx - Redirection	301 Moved Permanently 302 Found (Temporary Redirect)
4xx - Client Error	400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found
5xx - Server Error	500 Internal Server Error 503 Service Unavailable

HTTP status codes are three-digit numbers that the server includes in responses to indicate the outcome of a request. They're grouped into five categories:

1xx codes are informational and indicate that the request has been received and the process is continuing. These are relatively rare in everyday web development.

2xx codes indicate success. The most common is 200 OK, which means the request was successful and the response contains the requested data. 201 Created is used when a resource has been successfully created, like after a successful POST request. 204 No Content indicates the request was successful but there's no content to return.

3xx codes indicate redirection. 301 Moved Permanently and 302 Found (temporary redirect) tell the client that the resource has moved to a different URL.

4xx codes indicate client errors. 400 Bad Request means the server couldn't understand the request due to bad syntax. 401 Unauthorized means authentication is required. 403 Forbidden means the client doesn't have permission to access the resource. 404 Not Found is the famous one - the requested resource doesn't exist.

5xx codes indicate server errors. 500 Internal Server Error is a generic server error message. 503 Service Unavailable means the server is temporarily unavailable, often due to maintenance or overload.

In ASP.NET MVC, you'll work with these status codes when creating controller actions, especially when handling errors or implementing security.

رموز حالة HTTPهي أرقام من ثلاثة أرقام يضمنها الخادم في الاستجابات للإشارة إلى نتيجة الطلب. يتم تجميعها في خمس فئات:

رموز 1 xxهي إعلامية وتشير إلى أنه تم استلام الطلب وأن العملية مستمرة. هذه نادرة نسبيًا في تطوير الويب اليومي.

رموز 2 xxتشير إلى النجاح. الأكثر شيوعًا هو 200 OK، والذي يعني أن الطلب كان ناجحًا وتحتوي الاستجابة على البيانات المطلوبة. يتم استخدام 201 Createdعند إنشاء مورد بنجاح، مثل بعد طلب POSTناجح. 204 No Contentيشير إلى أن الطلب كان ناجحًا ولكن لا يوجد محتوى للعودة.

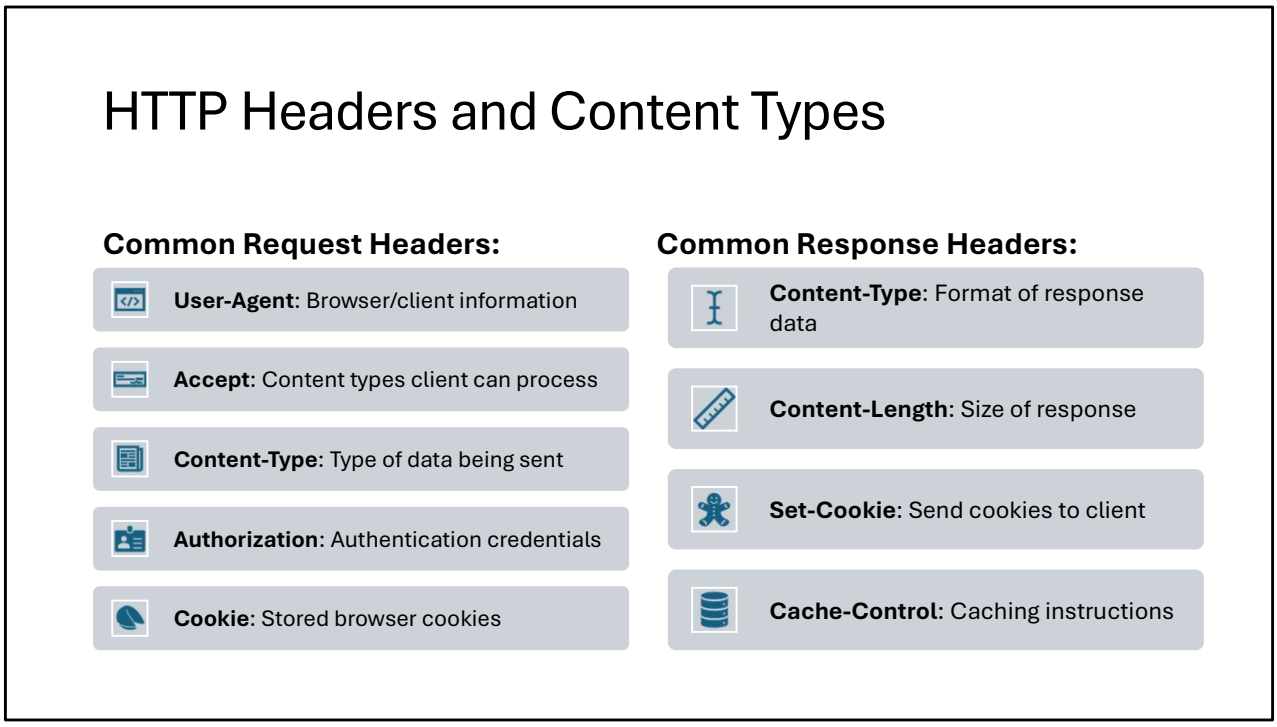
رموز 3 xxتشير إلى إعادة التوجيه. 301 Moved Permanentlyو 302) Foundإعادة توجيه مؤقتة) تخبر العميل أن المورد قد انتقل إلى عنوان URLمختلف.

رموز 4 xxتشير إلى أخطاء العميل. 400 Bad Requestيعني أن الخادم لم يتمكن من فهم الطلب بسبب بناء جملة سيئ. 401 Unauthorizedيعني أن المصادقة مطلوبة. 403 Forbiddenيعني أن العميل ليس لديه

إذن للوصول إلى المورد. 404 Not Foundهو الرمز المشهور - المورد المطلوب غير موجود.

رموز 5 xxتشير إلى أخطاء الخادم. 500 Internal Server Errorهي رسالة خطأ عامة في الخادم. 503 Service Unavailableيعني أن الخادم غير متاح مؤقتًا، غالبًا بسبب الصيانة أو الحمل الزائد.

في ASP.NET MVC، ستعمل مع رموز الحالة هذه عند إنشاء إجراءات وحدة التحكم، خاصة عند التعامل مع الأخطاء أو تنفيذ الأمان.



HTTP headers provide additional information about the request or response. They are key-value pairs sent along with the HTTP request or response.

Request headers are sent from the client to the server. The User-Agent header identifies the browser or application making the request. Accept tells the server what content types the client can handle. If you're submitting data, Content-Type specifies the format (like form data or JSON). Authorization headers carry credentials for protected resources, and Cookie headers send stored cookies back to the server.

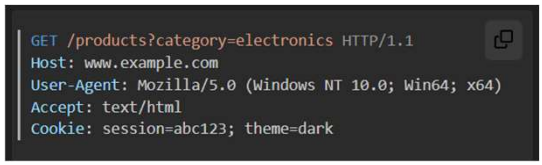
Response headers come from the server to the client. Content-Type tells the browser what kind of data is being returned, like HTML, JSON, or an image. Content-Length indicates the size of the response body. Set-Cookie instructs the browser to store cookies for future requests. Cache-Control provides directives on how the response should be cached.

MIME types (Multipurpose Internet Mail Extensions) identify the format of the content. For example, text/html is for HTML documents, application/json is for JSON data, image formats have types like image/jpeg, and application/pdf is for PDF documents.

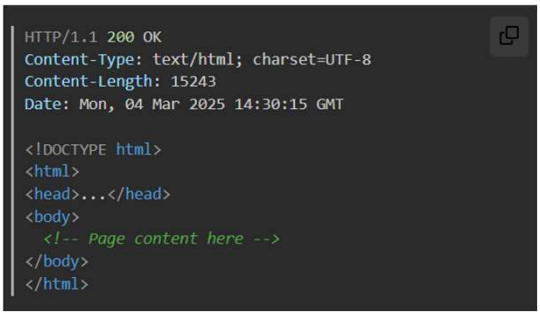
In ASP.NET MVC, most header handling is done automatically, but you can explicitly set headers when needed. For example, you might set Content-Type to application/json when returning JSON data from an API endpoint.

Understanding headers is crucial for handling authentication, caching, content negotiation, and troubleshooting web applications.

HTTP Example - Request and Response



```
GET /products?category=electronics HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html
Cookie: session=abc123; theme=dark
```



```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 15243
Date: Mon, 04 Mar 2025 14:30:15 GMT

<!DOCTYPE html>
<html>
<head>...</head>
<body>
  <!-- Page content here -->
</body>
</html>
```

Let's look at a concrete example of HTTP communication to solidify our understanding. Here's a typical HTTP request and response pair for accessing a product listing page.

In the request, we see GET /products?category=electronics HTTP/1.1. This indicates it's a GET request to retrieve products in the electronics category. The Host header specifies the domain, while User-Agent identifies the browser. The Accept header indicates the client can process HTML content, and Cookie sends stored cookies from previous interactions.

The server responds with HTTP/1.1 200 OK, indicating the request was successful. Content-Type tells the browser it's receiving HTML content encoded in UTF-8, and Content-Length specifies the size of the response in bytes. The Date header shows when the response was generated.

Following the headers is the actual HTML content of the page that will be rendered by the browser.

When developing with ASP.NET MVC, you won't usually see the raw HTTP messages as shown here, but the framework processes them behind the scenes. Understanding this communication helps you troubleshoot issues, optimize performance, and implement security features correctly.

For example, in an ASP.NET MVC controller action, returning View() generates an HTTP response with Content-Type: text/html, while returning Json(data) generates a response with Content-Type: application/json.

دعونا ننظر إلى مثال ملموس للاتصال HTTPلتعزيز فهمنا. هنا زوج من طلب واستجابة HTTPنموذجيين للوصول إلى صفحة قائمة المنتجات.

في الطلب، نرى .GET /products?category=electronics HTTP/1.1هذا يشير إلى أنه طلب GET لاسترداد المنتجات في فئة الإلكترونيات. يحدد رأس Hostالمجال، بينما يحدد User-Agentالمتصفح. يشير رأس Acceptإلى أن العميل يمكنه معالجة محتوى HTML، ويرسل Cookieملفات تعريف الارتباط المخزنة من التفاعلات السابقة.

يستجيب الخادم بـ HTTP/1.1 200 OK، مما يشير إلى أن الطلب كان ناجحًا. يخبر Content-Typeالمتصفح أنه يتلقى محتوى HTMLمشفرًا بـ UTF-8، ويحدد Content-Lengthحجم الاستجابة بالبايت. يوضح رأس Dateمتى تم إنشاء الاستجابة.

بعد الرؤوس يأتي محتوى HTMLالفعلي للصفحة التي سيتم عرضها بواسطة المتصفح.

عند التطوير باستخدام ASP.NET MVC، لن ترى عادةً رسائل HTTPالخام كما هو موضح هنا، ولكن الإطار يعالجها في الخلفية. يساعدك فهم هذا الاتصال على استكشاف المشكلات وإصلاحها وتحسين الأداء وتنفيذ ميزات الأمان بشكل صحيح.

على سبيل المثال، في إجراء وحدة تحكم ASP.NET MVC، يؤدي إرجاع View()إلى إنشاء استجابة HTTP مع Content-Type: text/html، بينما يؤدي إرجاع Json(data)إلى إنشاء استجابة مع Content-Type: application/json.

Statelessness and State Management

HTTP is stateless - each request is independent

Challenges for web applications:

- User authentication
- Shopping carts
- Multi-step processes
- User preferences

Common state management techniques:

- Cookies (client-side data)
- Session state (server-side storage)
- Hidden form fields
- Query strings
- Local/Session storage (HTML5)
- Token-based authentication

One of the most important characteristics of HTTP is that it's stateless - each request is completely independent from previous requests. The server doesn't naturally "remember" who you are or what you've done before.

This creates challenges for web applications that need to maintain state. For example, how does a shopping cart remember what items you've added? How does a website know you're logged in as you navigate between pages? How do multi-step processes like checkout flows work?

To address these challenges, web developers use various state management techniques:

Cookies are small pieces of data stored in the browser that are sent with every request to the server. They're commonly used for remembering login state, user preferences, and tracking information.

Session state stores information on the server linked to a session ID (typically stored in a cookie). This allows larger amounts of data to be maintained without sending it back and forth.

Hidden form fields embed data within HTML forms that get submitted with the form. They're useful for maintaining state through a specific process flow.

Query strings append data to URLs, like ?userid=123. They're visible to users and have length limitations but are simple to implement.

Modern applications also use HTML5 technologies like Local Storage and Session Storage for client-side state management, and token-based authentication systems like JWT (JSON Web Tokens) for secure state management.

In ASP.NET MVC, you'll work with these techniques, particularly session state and cookies, to maintain state across requests. The framework provides built-in support for these mechanisms.

إحدى أهم خصائص HTTPهي أنه بلا حالة - كل طلب مستقل تمامًا عن الطلبات السابقة. لا "يتذكر" الخادم بشكل طبيعي من أنت أو ما فعلته من قبل.

هذا يخلق تحديات لتطبيقات الويب التي تحتاج إلى الحفاظ على الحالة. على سبيل المثال، كيف تتذكر سلة التسوق العناصر التي أضفتها؟ كيف يعرف موقع الويب أنك مسجل الدخول أثناء التنقل بين الصفحات؟ كيف تعمل العمليات متعددة الخطوات مثل تدفقات الدفع؟

لمعالجة هذه التحديات، يستخدم مطورو الويب تقنيات مختلفة لإدارة الحالة:

ملفات تعريف الارتباط ((Cookiesهي قطع صغيرة من البيانات مخزنة في المتصفح ويتم إرسالها مع كل

طلب إلى الخادم. يتم استخدامها بشكل شائع لتذكر حالة تسجيل الدخول وتفضيلات المستخدم ومعلومات التتبع.

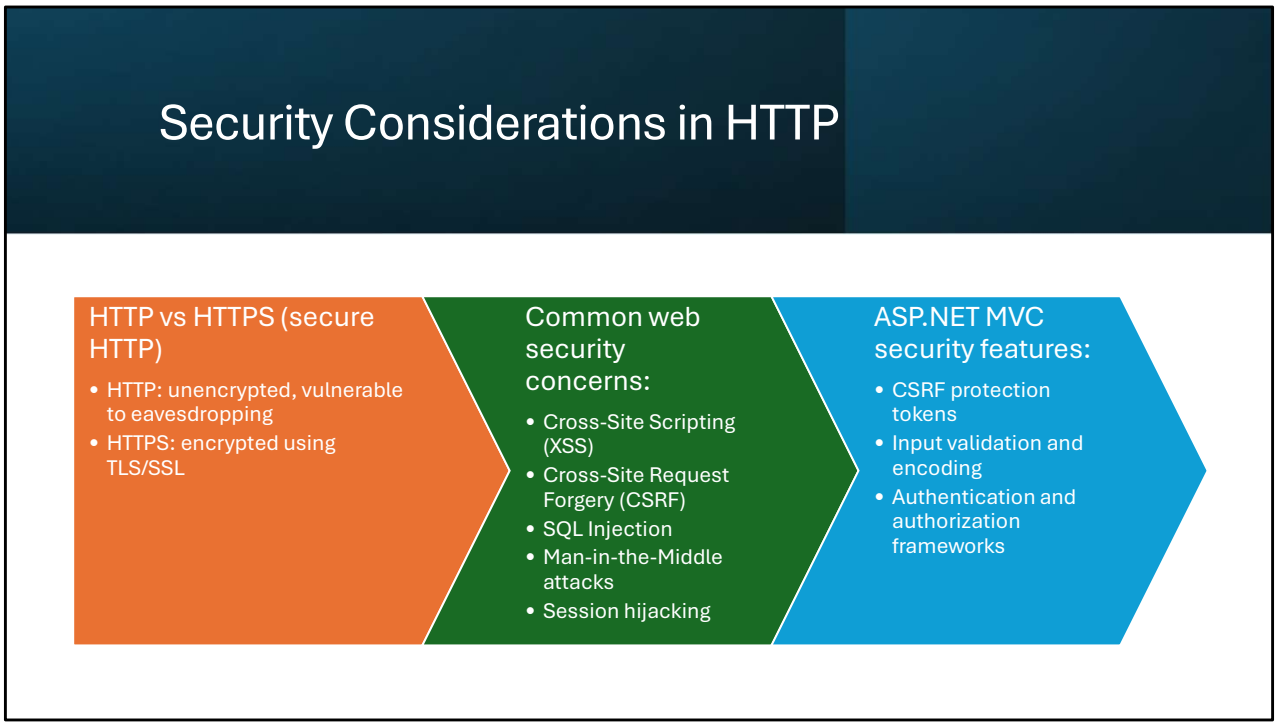
حالة الجلسة ((Session stateتخزن المعلومات على الخادم مرتبطة بمعرف الجلسة (عادة ما يتم تخزينه في ملف تعريف ارتباط). هذا يسمح بالحفاظ على كميات أكبر من البيانات دون إرسالها ذهابًا وإيابًا.

حقول النموذج المخفية ((Hidden form fieldsتضمن البيانات داخل نماذج HTMLيتم إرسالها مع النموذج. إنها مفيدة للحفاظ على الحالة خلال تدفق عملية محددة.

سلاسل الاستعلام ((Query stringsتضيف البيانات إلى عناوين URL، مثل ? .userid=123وهي مرئية للمستخدمين ولها قيود طول ولكنها بسيطة التنفيذ.

تستخدم التطبيقات الحديثة أيضًا تقنيات HTML5مثل التخزين المحلي ((Local Storageوتخزين الجلسة ((Session Storageلإدارة الحالة من جانب العميل، وأنظمة المصادقة القائمة على الرموز المميزة مثل) JWTرموز الويب (JSONلإدارة الحالة الآمنة.

في ASP.NET MVC، ستعمل مع هذه التقنيات، خاصة حالة الجلسة وملفات تعريف الارتباط، للحفاظ على الحالة عبر الطلبات. يوفر الإطار دعمًا مدمجًا لهذه الآليات.



Security is a critical consideration when developing web applications. Standard HTTP is unencrypted, which means data transferred between client and server can be intercepted and read by malicious actors. This is especially problematic for sensitive information like passwords or credit card numbers.

HTTPS addresses this vulnerability by encrypting communications using TLS (Transport Layer Security) or its predecessor SSL (Secure Sockets Layer). Always use HTTPS for production applications, especially those handling sensitive user data.

Beyond encryption, web applications face other security challenges:

Cross-Site Scripting (XSS) occurs when attackers inject malicious scripts into pages viewed by users. ASP.NET MVC helps prevent this by automatically encoding output.

Cross-Site Request Forgery (CSRF) tricks users into performing unwanted actions on sites where they're authenticated. ASP.NET MVC provides anti-forgery tokens to prevent this.

SQL Injection attempts to execute malicious SQL commands by manipulating input data. This can be prevented using parameterized queries and Entity Framework.

ASP.NET MVC includes several built-in security features to help address these concerns, including:

- Anti-forgery tokens for forms
- Output encoding to prevent XSS
- Input validation attributes for models
- Authentication and authorization frameworks

Throughout this course, we'll incorporate these security practices into our applications. Remember that security is not a feature but a continuous process that should be integrated throughout the development lifecycle.

الأمان هو اعتبار حاسم عند تطوير تطبيقات الويب. HTTPالقياسي غير مشفر، مما يعني أن البيانات المنقولة بين العميل والخادم يمكن اعتراضها وقراءتها من قبل الجهات الخبيثة. هذا مشكلة خاصة للمعلومات الحساسة مثل كلمات المرور أو أرقام بطاقات الائتمان.

يعالج HTTPSهذه الثغرة عن طريق تشفير الاتصالات باستخدام) TLSطبقة أمان النقل) أو سلفها) SSLطبقة المقابس الآمنة). استخدم دائمًا HTTPSلتطبيقات الإنتاج، خاصة تلك التي تتعامل مع بيانات المستخدم الحساسة.

بالإضافة إلى التشفير، تواجه تطبيقات الويب تحديات أمنية أخرى:

يحدث Cross-Site Scripting (XSS)عندما يقوم المهاجمون بحقن نصوص برمجية ضارة في الصفحات التي

يشاهدها المستخدمون. يساعد ASP.NET MVCفي منع ذلك عن طريق تشفير المخرجات تلقائيًا.

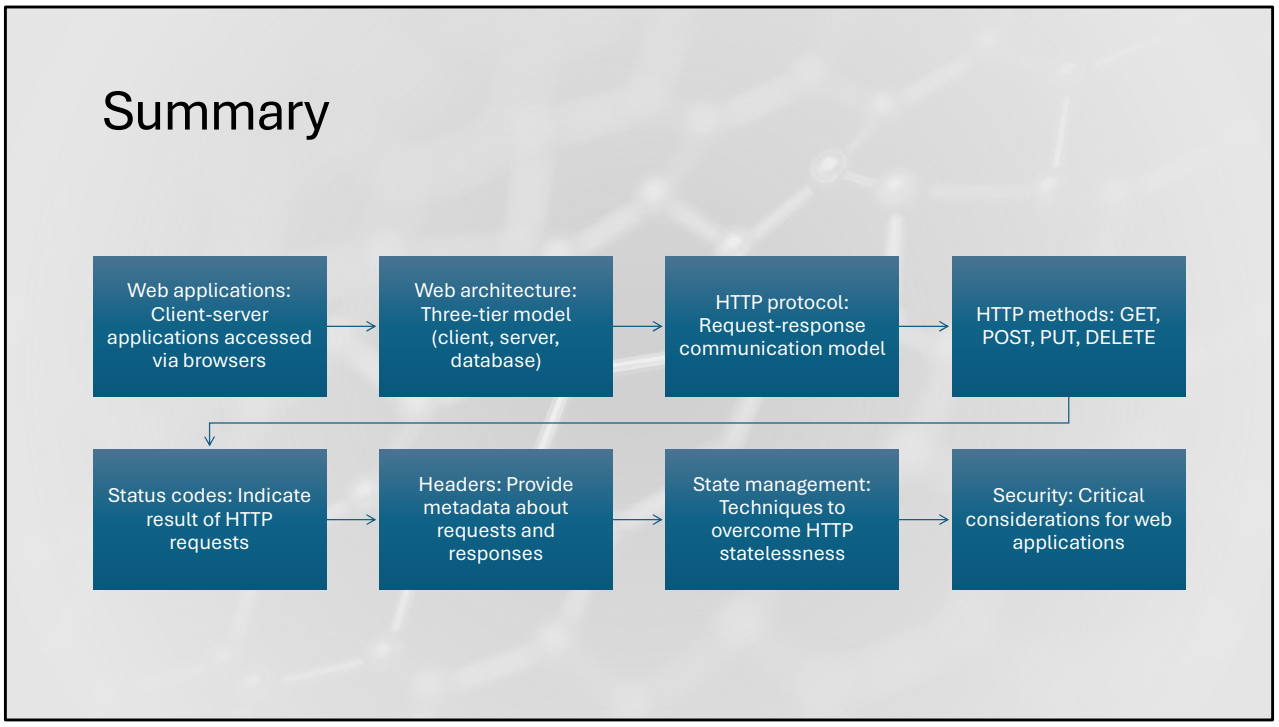
Cross-Site Request Forgery (CSRF)يخدع المستخدمين لأداء إجراءات غير مرغوب فيها على المواقع التي تم مصادقتهم عليها. يوفر ASP.NET MVCرموز مكافحة التزوير لمنع ذلك.

تحاول حقن SQL (SQL Injection)تنفيذ أوامر SQLضارة عن طريق التلاعب ببيانات الإدخال. يمكن منع ذلك باستخدام الاستعلامات المعلمة و.Entity Framework

يتضمن ASP.NET MVCالعديد من ميزات الأمان المدمجة للمساعدة في معالجة هذه المخاوف، بما في ذلك:

- رموز مكافحة التزوير للنماذج
- تشفير المخرجات لمنع XSS
- سمات التحقق من صحة الإدخال للنماذج
- أطر المصادقة والتفويض

خلال هذه الدورة، سنقوم بدمج ممارسات الأمان هذه في تطبيقاتنا. تذكر أن الأمان ليس ميزة ولكنه عملية مستمرة يجب دمجها طوال دورة حياة التطوير.



Let's summarize what we've covered in this theoretical section. We've explored the fundamentals of web applications - software that runs on a web server and is accessed through a browser. They follow a three-tier architecture separating presentation, application logic, and data storage.

We've learned about the HTTP protocol, which is the foundation of communication on the web. HTTP follows a request-response model where clients send requests and servers respond with the requested data or appropriate error messages.

We've examined the key components of HTTP:

- HTTP methods like GET, POST, PUT, and DELETE that indicate the desired action
- Status codes that communicate the result of requests
- Headers that provide metadata about requests and responses

- Content types that specify the format of data

We've also discussed the stateless nature of HTTP and strategies for maintaining state across requests, such as cookies and session management. Finally, we touched on important security considerations, including the need for HTTPS and protection against common web vulnerabilities.

These concepts form the foundation for understanding ASP.NET MVC, which we'll start exploring in the next section. The MVC framework provides a structured approach to building web applications on top of the HTTP protocol, handling many of the details automatically while giving you control when needed.

دعونا نلخص ما تناولناه في هذا القسم النظري. لقد استكشفنا أساسيات تطبيقات الويب - البرامج التي تعمل على خادم ويب ويتم الوصول إليها من خلال متصفح. فهي تتبع بنية ثلاثية الطبقات تفصل العرض ومنطق التطبيق وتخزين البيانات.

لقد تعلمنا عن بروتوكول HTTP، الذي هو أساس الاتصال على الويب. يتبع HTTPنموذج الطلب والاستجابة حيث يرسل العملاء الطلبات ويستجيب الخوادم بالبيانات المطلوبة أو رسائل الخطأ المناسبة.

لقد درسنا المكونات الرئيسية لـ :HTTP

- طرق HTTPمثل GETو POSTو PUTو DELETEالتي تشير إلى الإجراء المطلوب
- رموز الحالة التي توصل نتيجة الطلبات
- الرؤوس التي توفر البيانات الوصفية حول الطلبات والاستجابات
- أنواع المحتوى التي تحدد تنسيق البيانات

لقد ناقشنا أيضًا طبيعة HTTPالتي لا تحتفظ بالحالة واستراتيجيات الحفاظ على الحالة عبر الطلبات، مثل ملفات تعريف الارتباط وإدارة الجلسة. أخيرًا، تطرقنا إلى اعتبارات الأمان المهمة، بما في ذلك الحاجة إلى HTTPS والحماية ضد نقاط ضعف الويب الشائعة.

تشكل هذه المفاهيم الأساس لفهم ASP.NET MVC، والذي سنبدأ في استكشافه في القسم التالي. يوفر إطار MVCنهجًا منظمًا لبناء تطبيقات الويب على رأس بروتوكول HTTP، حيث يتعامل مع العديد من التفاصيل

تلقائيًا مع منحك التحكم عند الحاجة.