Creating Evolvable Hypermedia Applications

Building

Hypermedia APIs with HTML5 & Node

O'REILLY®

Mike Amundsen

Building Hypermedia APIs with HTML5 & Node

With this concise book, you'll learn the art of building hypermedia APIs that don't simply run *on* the Web, but that actually exist *in* the Web. You'll start with the general principles and technologies behind this architectural approach, and then dive hands-on into three fully functional API examples.

Too many APIs rely on concepts rooted in desktop and local area network patterns that don't scale well—costly solutions that are difficult to maintain over time. This book shows system architects and web developers how to design and implement human- and machine-readable web services that remain stable and flexible as they scale.

- Learn the H-Factors for representing application metadata across all media types and formats
- Understand the four basic design elements for authoring hypermedia types
- Convert a simple read-only XML-based media type into a successful API design
- Examine the challenges and advantages of designing a hypermedia type with JSON
- Use HTML5's rich set of hypermedia controls in the API design process
- Learn the details of documenting, publishing, and registering media type designs and link-relation types

Twitter: @oreillymedia
facebook.com/oreilly

US $24.99 CAN $26.99
ISBN: 978-1-449-30657-1

O'REILLY®
oreilly.com

Building Hypermedia APIs with HTML5 and Node

Mike Amundsen

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Building Hypermedia APIs with HTML5 and Node
by Mike Amundsen

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Simon St. Laurent
Production Editor: Melanie Yarbrough

Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

Revision History for the First Edition:
2011-11-21 First Release
See *http://oreilly.com/catalog/errata.csp?isbn=9781449306571* for release details.

ISBN: 978-1-449-30657-1

[LSI]

1321983647

"The human mind ... operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain."

—Vannevar Bush, 1945

"*If computers are the wave of the future, displays are the surfboards".*

—Ted Nelson, 1974

"HyperText is a way to link and access information of various kinds as a web of nodes in which the user can browse at will."

—Tim Berners-Lee, 1992

"Hypermedia is defined by the presence of application control information embedded within, or as a layer above, the presentation of information."

—Roy T. Fielding, 2001

"The WWW is fundamentally a distributed hypermedia application."

—Richard Taylor, 2010

Table of Contents

Foreword

You can't talk about something if you don't have the words.

The World Wide Web is driven by hypermedia: the ability of a document to describe its possible states, and its relationship to other documents. Hypermedia is not just a way of making websites that average people can use; it's a new style for distributed computing, powerful and flexible.

There's nothing new about the web technologies or the hypermedia concept: in another world, we could have been using hypermedia for distributed computing since the mid-1990s. Instead, we've been slow to adopt hypermedia for anything but consumer use. It's an easy concept to grasp intuitively—we all use the Web—but it's difficult to understand in a context of development.

Our problems stem from conceptual blocks. The Web invaded our everyday lives years before its architecture was formally described. We've spent the twenty-first century making gradual progress, coming up with new vocabulary to help developers come to terms with the power of the Web—power that was there all along.

The description of hypermedia you'll read in this book is, in my opinion, one of the biggest conceptual advances since Roy Fielding first defined the REST architectural style. Mike Amundsen has taken the blanket term "hypermedia" and taken it apart to see exactly what it can mean and how it works.

What makes a data format useful for some applications and not others? Why is HTML so versatile, even for nonconsumer applications, and where does it fall short? Under Mike's view of hypermedia, these questions have precise answers—answers that I hope will drive the next generation of web services and web-based technologies.

Mike has not only found the words to describe hypermedia, he's given voice to our intuitions about how it works.

—Leonard Richardson, November 2011

Preface

When you set out on your journey to Ithaca,
pray that the road is long, full of adventure,
full of knowledge.

- Constantine P. Cavafy

Hypermedia API Design

This book's primary focus is on designing hypermedia APIs. That may seem a bit strange to some readers. There are many books on programming languages, data storage systems, web frameworks, etc. This is not one of those books. Instead, this book covers the nature of the messages passed between client and server, and how to improve the content and value of those messages. I, personally, find this to be an exciting and fascinating area.

As of this writing anecdotal trends seem to indicate an ever-increasing reliance on APIs in web development. In general, this is a good thing. It means more and more developers are catching on to the notion that the World Wide Web is a great place to share not only data, but also services, a goal of those who championed the web in its early days.

However, I believe that this explosion of web APIs may lead us down a troublesome path. In my experience over the last few years, I have seen too many examples of implementations that rely on concepts of APIs rooted in desktop and local area network patterns that will not scale well at the WWW level, solutions still exhibiting brittleness that can lead to costly and frustrating maintenance issues as time goes by. In short, I don't see enough hypermedia in API offerings for the web.

This book is an attempt to improve the chances that new APIs added to the WWW will be easier to use and maintain over time, and that they will take their cue from those who were responsible for the discovery of the value of hypermedia linking; the codification of the HTTP protocol; and the implementation of HTML, Atom/AtomPub, and other native hypermedia formats that still drive the growth of the web today.

Intended Audience

The primary goal of this book is to increase both the quantity and quality of hypermedia content in use on the web. To that end, the audience for this text is two-fold.

First, this book is offered as a guide to system architects. Hopefully the text can be a valuable guide for those responsible for designing systems that rely on hypermedia to improve the evolvability and stability of long-lived implementations. When viewed as an integral part of system architecture, hypermedia provides a wealth of possibilities to architects. Hopefully this book will illustrate that, by treating hypermedia data as a key architectural component (rather than merely a payload to be pushed about by clients and servers), architects can increase future stability and flexibility of their systems.

Second, readers tasked with implementing clients and servers will find valuable advice and examples on how to deal with hypermedia messages themselves. Up to now, most books on web implementations have focused too often on the role of servers in dealing hypermedia. It is the author's view that this oversight too often results in improper client implementations that not only ignore, but often negate the value of hypermedia messages on the web. One of the key advantages of hypermedia as an architectural pillar is that hypermedia encourages clients to "code for the media type" instead of writing applications that treat messages as simple data. Writing hypermedia-aware clients is a skill that takes time to master. And while this book does not focus solely on writing hypermedia clients, the author hopes that it will show enough examples and advantages as to spur other, more talented individuals to establish new practices and techniques aimed at taking direct advantage of hypermedia.

What Is Not Covered

While the examples in this book use HTML5, Node.js, and CouchDB, this book should not be used as a source for learning these technologies. Astute readers may find the author's use of these tools—HTML5, Node.js, CouchDB—somewhat stilted and possibly, to some, blasphemous. The author makes no claims at expertise in these technologies. Instead, in the context of this book, they are used as tools for illustrating points about hypermedia design and implementation. The appendices list several good books on the technologies used in the writing of this text that the reader is encouraged to refer to for a more authoritative voice on these matters.

This book does not cover the details of the HTTP protocol and associated web standards. There is a wealth of writing available and the appendices reference important RFCs and other standards documents used while preparing this text. The reader will also find several book recommendations in the appendices well worth the time to read and become acquainted.

Finally, while the subject of the Representation State Transfer (REST) architectural style comes up occasionally, this book does not explore the topic at all. It is true that REST identifies hypermedia as an important aspect of the style, but this is not the case for the inverse. Increasing attention to hypermedia designs can improve the quality and functionality of many styles of distributed network architecture including REST. Readers who want to learn more about Fielding's style will find helpful recommendations in the appendices.

Contents of This Book

The book is designed to allow readers to jump around to sections they find interesting; you do not need to read it cover-to-cover in sequential order. There are a number of links within the chapters to point the reader to related material that may have been missed when skipping around in the text. Hopefully this format will also make the text more useful as a reference when the reader wants to refer back to content at a later date.

The general layout of the book is as follows:

Chapter 1: Understanding Hypermedia
: This is the conceptual chapter of the book. It provides some historical references for hypermedia, HTTP, and HTML, and then goes on to lay out the basic premise of the text including making a case for more hypermedia, offering an analysis of existing hypermedia content, and a suggested methodology for creating new hypermedia designs.

Chapters 2, 3, and 4: Implementations
: The middle chapters contain complete walk-throughs of fully functional hypermedia examples. These chapters are meant to lead the reader through the process of assessing an application scenario, selecting design elements, creating sample data, and implementing complete server and client solutions that meet the use case requirements. While the examples are kept relatively basic, they are still meant to convey most of the details the reader is expected to encounter when creating real-life production-ready solutions.

Chapter 5: Documenting Hypermedia
: This is the housekeeping chapter of the book. It provides tips on documenting media type designs and registering those designs with standards bodies such as the IANA, IEFT, and WC3. There is a section covering the concepts of Versioning and Extending hypermedia types as well as some general tips on good API and hypermedia designs.

Appendices
: This book contains a number of appendices. These are included as pointers to quoted and referenced materials as well as to hold additional content that did not fit well into the flow of the chapters. The information here may also be valuable for future reference after the reader has already completed the body of the book.

Coding Style for This Book

One of the reasons Node.js and CouchDB were selected for this book is that, from the beginning, these products are HTTP-aware. That means the software works well using the existing HTTP application protocol and in a state-less environment like the World Wide Web. As a result, there is very little friction between the components I created using Node and CouchDB, and the protocol used to communicate with those components.

It is also an advantage that these software systems all use the same front-facing programming language for scripting (Javascript). While not all readers will be proficient in Javascript, hopefully this single language format can reduce the need for mental context-switching when moving between client code, server code, and data storage implementation.

The important point, though, is that the *software* is not the focus for this book; it is merely the medium for the hypermedia message. You will likely find that many of the examples contain code that is either too brief or too fragile to run in a production environment. This is mostly a matter of expediency; I'm anxious to illustrate the details of the hypermedia, not the code used to implement that design. These designs will work well when implemented for any platform, using any language, running on any operating system. I suspect many readers will find better ways to implement these media type designs using their own languages and platforms, and that's all the better.

Conventions Used in This Book

The following typographical conventions are used in this book:

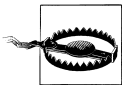
Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Building Hypermedia APIs with HTML5 and Node* by Mike Amundsen (O'Reilly). Copyright 2012 O'Reilly Media, Inc., 978-1-449-30657-1."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at *http://my.safaribooksonline.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.

1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://shop.oreilly.com/product/0636920020530.do

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgements

There are quite a number of people who deserve acknowledgement for the completion of this book.

Several people volunteered to review early drafts of this book and provided excellent feedback and suggestions. Thanks to Leonard Richardson, Erik Wilde, Ian Robinson, Jan Algermissen, Mike Kelly, Will Hartung, William Martinez Pomares, Erlend Hamnaberg, Darrel Miller, Glenn Block, David Zulke, Erik Morgensen, Kevin Burns Jr., Jonathan Moore, and Subbu Allamaraju. If the book is accurate, clear, and concise, that is likely due to the contributions of these individuals. Any remaining shortcomings are solely the responsibility of the author.

Julian Reschke, Mykyta Yevstifeyev, Frank Ellermann, and others were very helpful and generous as I worked to learn the details of IETF and IANA procedures and processes.

I'd like to thank Benjamin Young for introducing me to CouchDB and for all his efforts to teach me how to improve my understanding and coding of CouchDB. His willingness to spend one-on-one time to help me get past some hurdles was invaluable. If there are shortcomings or errors in the CouchDB code, it's most likely because I failed to grasp what Benjamin tried to teach me; my apologies to the reader and to Benjamin Young.

Simon St. Laurent, my editor, has been a great champion of this book. Without his tireless efforts, this book would never have seen the light of day. Thanks also to Melanie Yarbrough for all her proofreading and editing work.

I want to thank the organizers of CodeStock, Stir Trek, JAOO (aka Gotocon), and OSCON. My presentations at these and other events over the last few years has given me a chance to explore, refine, and correct initial concepts and methodologies.

I owe special thanks to all of the attendees of REST Fest 2010 and 2011. It was during very productive and enjoyable weekends in lovely Greenville, South Carolina, that I was able to first publicly describe and refine my ideas about analyzing and designing hypermedia.

I've benefited quite a bit from the conversations on the REST IRC channel on freenode.org. My thanks to all who hang out there for all of the great feedback and ideas.

Thanks, finally, to the moderators and members of the REST-Discuss list. Over the years I've posted many questions, assertions, and comments to that list in an effort to learn more about Fielding's style and HTTP implementation in general. In most cases, many of the correct things I learned about REST and HTTP were a result of my REST-Discuss experience.

CHAPTER 1
Understanding Hypermedia

There is no end to the adventures that we can have if only we seek them with our eyes open.

- Nehru

Designing scalable, flexible implementations that live on the web relies on many concepts, technologies, and implementation details. Understanding the history behind the way the World Wide Web (WWW) works today and the various standards and practices created to support it is an essential part of developing skills as a web architect and hypermedia designer.

In addition to understanding the technologies behind the Web, designers also need to be aware of the differences between implementing applications on a distributed network that leverages coarse-grained message protocols like HTTP that span multiple platforms, programming languages, and storage systems, and more traditional local networked applications where most components in the network share similar storage, programming, and operating details. Programming for distributed hypermedia environments usually means that message transfers must carry more than just data; they must carry additional information including metadata and higher-level application flow control options. The Web thrives on this style of rich hypermedia, and it is important to design APIs that support this method of sharing understanding of the data sent between network participants.

The way in which hypermedia information is shared varies between data formats, but the actual hypermedia concepts—links, templated queries, idempotent updates, etc.—are the same across the Web. Having a clear understanding of the various hypermedia factors that can be expressed within a message is an essential part of developing the skills needed to implement successful hypermedia APIs.

Finally, the implementation details of creating hypermedia types include selecting an appropriate data format and state transfer style, expressing the application domain details properly within the format, and determining the way in which application flow

control options are exposed in responses. These basics of hypermedia design are similar regardless of data to be shared or the application domain involved.

This chapter covers some key technologies that make the Web possible, the importance of hypermedia as an architectural approach, and the concepts and details of hypermedia designs. Armed with this understanding the reader will have the tools necessary to build scalable, flexible implementations that do not simply run *on* the Web but that actually exist *in* the web in a way that acknowledges, as stated by Richard Taylor, that the "WWW is fundamentally a distributed hypermedia application."

HTTP, MIME, and Hypermedia

Designing hypermedia applications for the Web is just that: a design process. This process depends on a handful of important standards and technologies. Chief among these are the protocols, message standards, and message content that allows participants to drive the system forward, often toward some designated goal (completing a search, purchasing an item online, etc.).

The most commonly used protocol for transferring content on the Web today is HTTP. Initially designed as a read-only protocol for exchanging HTML documents, HTTP quickly expanded from a file-based transfer protocol into a more generalized protocol that supports both read and write operations across multiple intermediaries that allow real-time negotiation of representation formats for a wide range of information stored on servers.

The ability to represent data instead of just mirror files derives from HTTP's use of the Multipurpose Internet Mail Extensions (MIME) media type system. Originally created for supporting electronic mail transfers, the MIME typing standard allows HTTP transfers to support a wide range of data formats including ones that are designed specifically for transferring application-related requests such as search parameters and data storage operations. Also important is the built-in ability to support new media types over time in order to support new, unanticipated uses for data transfer on the Web.

The creation of the Web was heavily influenced by the notion of hypermedia and the ability to link related material and easily follow these links in real time. To this end, media types have been designed to natively support hypermedia controls as a way to enable client applications to make selections found within responses and drive the state of the application to the desired outcome. This allows client applications to discover the specific controls within the media type message with which to modify the state of the Web (or at least that client's view of the Web). It is hypermedia, and the design and implementation of it, that makes the Web a unique and powerful environment for building distributed network applications.

HTTP Is the Transfer Protocol

The first version of HTTP (documented in 1991 as HTTP/0.9) was a simple read-only protocol. It allowed clients to send a request string made up of the letters GET followed by a space and a document address (what we know as a URI today) and servers could then respond by returning the associated document. HTTP/0.9 had no metadata headers and the response was always assumed to be in HTML form.

By 1992, a Basic HTTP (considered a full specification) was documented. This version included several new methods including HEAD, PUT, POST, and DELETE. Other methods such as CHECKIN, CHECKOUT, LINK, UNLINK, SEARCH, and several others are no longer in the specification today. This document included the concept of response codes (1xx, 2xx, 3xx, 4xx, 5xx), an early form of content negotiation, and meta-information (later known as HTTP Headers).

Although the 1992 document was never officially released through a governing body (IETF, etc.), over the next few years, browser clients and web servers extended the read-only features of HTTP/0.9 to include many of the headers and additional request methods identified as Basic HTTP. The results of these additions were documented in 1996 in RFC1945 and named HTTP/1.0. Not long after RFC1945 was released, in January of 1997, HTTP/1.1 was documented and released as RFC2068. This refined many of the implementation details of HTTP as we know it today. An additional update was released in June of 1999 as RFC2616. This is the version of HTTP that is most commonly deployed today.

Throughout the history of specifying HTTP, the protocol was always designed to work as a client-initiated, stateless protocol for transferring messages between parties running over TCP/IP. There are other transfer protocols including FTP, BitTorrent, and rsync, but HTTP has remained the dominant transfer protocol for the Web. This focus on stateless transfer as well as HTTP's support for negotiating the format used to represent server data (via the Accept and Content-Type headers) are key to understanding how to best utilize HTTP and design efficient and effective applications for the Web. Architects and developers who create applications that run counter to these design principles not only ignore the strengths of HTTP but also add unnecessary complexity and complications to their implementations.

MIME Is the Media Type Standard

HTTP's standard for defining the body of the message sent between parties is based on the MIME standard (RFC2046). Although this standard was designed to support exchanging messages via email, MIME was adopted by HTTP in order to support the notion of resource representations and to allow for future extensibility.

One of HTTP's important concepts is the idea that responses are only *representations* of the actual data. For example, under the FTP (File Transfer Protocol) specifications, the goal is to send an exact copy of the data between parties. However in HTTP, servers

are free to represent the data in various ways and clients are encouraged to inform servers which representation formats are preferred. In order to support this additional feature, the MIME standard is used to indicate the current representation format.

MIME Type, Media Type, and Content Type

The terms MIME Type, Media Type, and Content Type as often used in similar ways. The term MIME comes from the initial RFCs describing media type handling for SMTP. Part two of that document collection regarding MIME (RFC2046) carries the sub-title of "Media Type." Subsequent RFC documents (e.g. RFC4288) refer to Media Type as the object of public registration for use in MIME and other Internet protocols. "Content Type" is the name of the HTTP Header that carries the media type information for the response message. Usually, when people use any one of these three phrases, they are referring to the media type registration string (e.g. application/xml, text/plain, etc.). Throughout this book the phrase "Media Type" is most often used unless there is a reference directly to the HTTP Header (Content-Type) or an historical reference to the original standards (MIME).

HTTP's support for varying response representations via a media type indicator opens the door to using the message as a key component in web architecture. Messages no longer need to be relegated to simply carrying raw data. Instead, designers and architects can leverage this opportunity to create new message formats and standards that can allow responses and requests to convey not just the raw data, but also metadata about the content. It also means that formats can be created to support very specific purposes independent of the application in use or the data that is transferred between parties.

For example, the same set of data points could be represented for use in a spreadsheet (text/csv), for display in a tabular view (text/html), or as a graphical pie chart display (image/png). Data sent from client to server can be represented as simple name-value pairs (application/x-www-formurlencoded), as plain text (text/plain), or even as part of a multiple-format collection (multipart/form-data).

There are more than a dozen multipart media types registered with the Internet Assigned Numbers Authority (IANA). These types are designed to support multiple unique media types within the same message body, each separated by a boundary. The HTTP/1.1 spec also defines message/http and application/http as similar container media types. These container media types are not covered in this book, but their very existence and use is evidence of flexibility and extensibility of the MIME media type standard.

The reliance on MIME media types also points to an important aspect of HTTP's message model: it was designed to send coarse-grained messages. Unlike some transfer

protocols whose aim is to send the smallest data packets possible, HTTP is concerned with including as much descriptive information as possible with each message, even if this means the message is longer than it needs to be. While there are some efforts underway to reduce message sizes (mostly by shortening or compressing HTTP Headers), designers and architects are free to use the body of the message to carry whatever is deemed important.

This freedom to design new message bodies for HTTP via the MIME standard leads to another unique aspect of the use of HTTP on the Web: in-message hypermedia. HTTP and MIME together make it not only possible, but common to include hypermedia information such as links and forms directly in the body of the response message. It is this ability to include hypermedia controls within messages that makes HTTP so well suited for use in distributed networks like the World Wide Web.

Hypermedia Is the Engine

The concept of hypermedia has been with us for quite a while. Vannevar Bush's 1945 article "As We May Think" envisioned a device (the "Memex") that allowed researchers to discover and follow links between topics and phrases in projected documents. Doug Engelbart's 1968 NLS (oN-Line System), one of the first graphical computer systems, used a new mechanical pointer device dubbed "the mouse" and allowed users to click on links to display related data on the screen. Similar examples of enabling links via computer display cropped up in the 1970s and 1980s. Ted Nelson coined the terms "hypertext" and "hypermedia" in the mid-1960s, and his work "Computer Lib/Deam Machines," published in 1974, is considered by many to be the first to establish the notion of "surfing the 'net" and cyberculture in general.

From links to controls

The initial concept for hypermedia was as a read-only link between related items and to this day, this is the most common way hypermedia is used on the Web. For example, many media types only support read-only links between elements. However, with the introduction of graphical user interfaces based on the use of Englebart's mouse as a way to activate elements of the interface (including buttons), the idea of using links as a way to perform other actions (sending a message, saving data, etc.) became accepted.

The development of HTTP mirrored this development from read-only (HTTP/0.9) to read/write linking (HTTP/1.0 and 1.1). Along the way, the de facto media type for HTTP (HTML) developed to include controls within messages that allowed users to supply arguments and send this data to remote servers for processing. These hypermedia controls included the `FORM` and `INPUT` elements among others. This ability to support not only navigational links (HTML anchor tags) and in-place rendering of related content (e.g. the `IMG` tag) but also parameterized queries and write operations helped HTML be become the *lingua franca* of the Web.