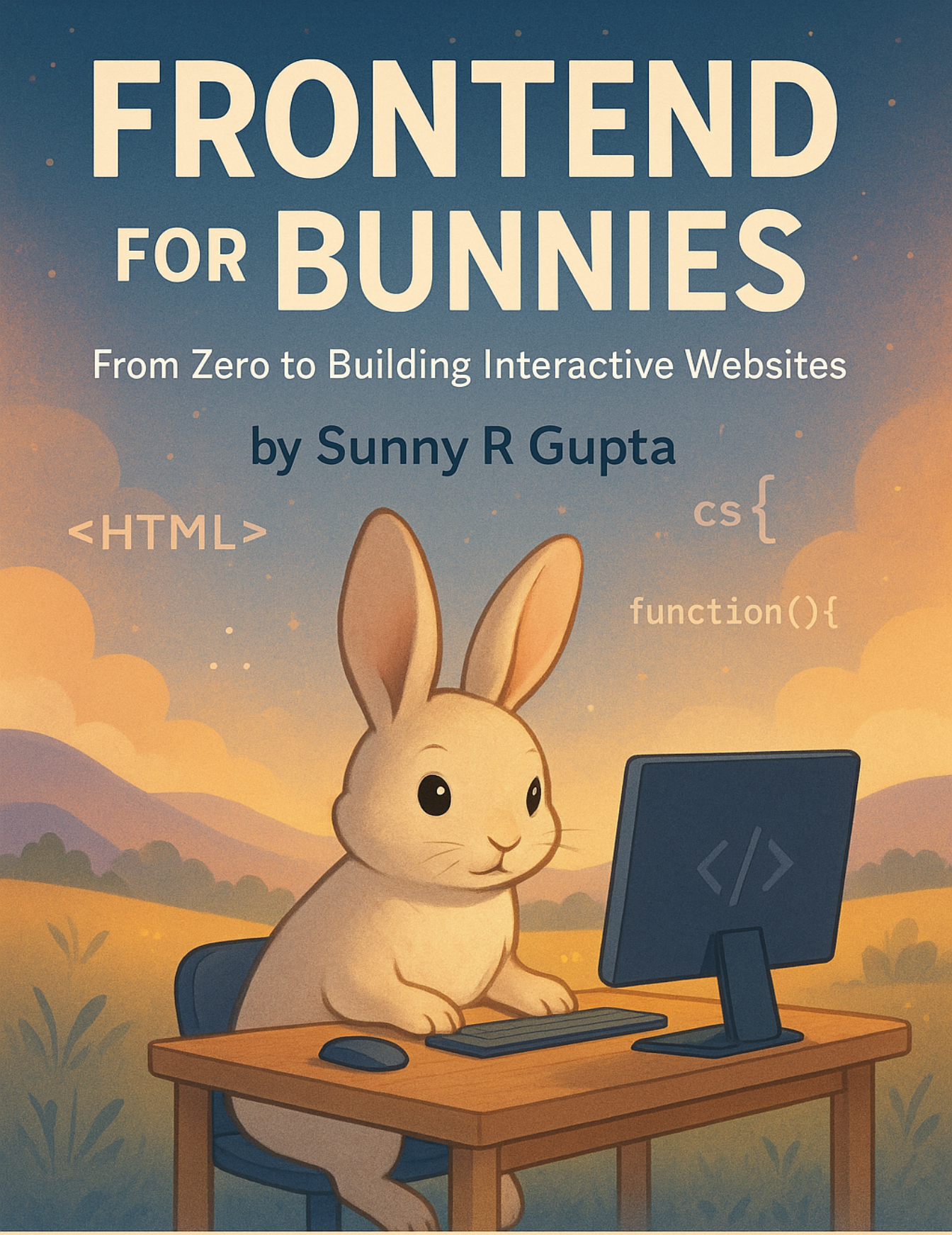
FRONTEND FOR BUNNIES

From Zero to Building Interactive Websites

by Sunny R Gupta



Part 1: FOUNDATIONS

Frontend for Bunnies

From Zero to Building Interactive Websites

Sunny R Gupta

This book is available at https://leanpub.com/frontendforbunnies

This version was published on 2026-01-07

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

Chapter 1: How Computers Work

Introduction

Welcome to the first chapter of your journey into frontend engineering. Before we write a single line of code, before we even open a browser, we need to understand the fundamental machine that powers everything we'll build: the computer.

You might think this chapter is unnecessary. After all, you've been using computers for years, right? But here's the truth: to become a great frontend engineer, you need to understand not just *how* to use a computer, but *why* it works the way it does. This understanding will make everything that comes next make sense. It's the difference between following a recipe and understanding cooking.

Think of it this way: a bunny can learn to hop without understanding its legs, but a bunny trainer needs to understand how those legs work to teach hopping effectively. Similarly, you can write code without understanding computers, but to write good code that performs well and solves real problems, you need to understand the machine beneath your fingers.

In this chapter, we'll journey from the most basic concepts (electricity and switches) to understanding how modern computers store information, process it, and communicate with each other. By the end, you'll have a clear mental model of what's happening inside every device you'll ever write code for.

1.1 What Is a Computer? The Warren's Management System

Let's start with a simple definition: **a computer is a machine that follows instructions to process information and produce results**.

That's it. That's the core. But let's unpack what that means.

Imagine you're managing a massive rabbit warren (a 'warren' is a series of underground tunnels where rabbits live). This warren has thousands of bunnies, and they need to accomplish tasks: gathering carrots, building burrows, organizing food supplies. As the warren manager, you could try to give instructions to every bunny individually, but that would be inefficient and error prone. So, what do you do?

Most people would create a system, like so:

- **A supervisor** (or several) who can process information and make decisions
- **A communication system** so supervisors can relay messages
- **A storage system** for keeping track of important information
- **Execution teams** who actually do the work based on instructions
- **A schedule** that determines 'what' happens 'when'

A computer is exactly this, but instead of bunnies, it has electronic components, and instead of natural language instructions, it uses binary (which we'll explain in a moment). The key insight is that a computer is fundamentally a system that takes instructions, stores information, processes it according to those instructions, and produces output.

In our warren analogy:

- The **supervisor** is the processor
- The **communication system** is the bus
- The **storage** is the memory and hard drive
- The **work teams** are various hardware components
- The **schedule** is determined by the program (the code)

Everything else is just implementation details.

1.2 The Core Components: What Every Computer Has

Every computer, whether it's the powerful desktop on your desk, the laptop you're using, or the smartphone in your pocket, has the same basic components. Understanding these is crucial.

The Processor (CPU): The Supervisor

The **CPU (Central Processing Unit)** is the brain of the computer. It's the component that actually executes instructions. Think of it as the head supervisor in our warren. The CPU does things like:

- Perform arithmetic operations (addition, subtraction, multiplication, division)
- Compare values (is this number bigger than that number?)
- Move data from one place to another
- Make decisions (if this condition is true, do this; otherwise do that)
- Repeat operations (loops)

Every single thing your computer does... from displaying text on your screen to running a complex calculation... ultimately comes down to the CPU executing a sequence of instructions. And it does this *incredibly* fast. A modern processor can execute billions of instructions per second.

Here's something to wrap your head around: when you're reading this sentence, your computer's CPU is executing millions of instructions per second, even though from your perspective, nothing seems to be happening particularly fast. This is because modern computers do so much work in parallel and manage it all so efficiently that we experience it as smooth performance.

Memory (RAM): The Supervisor's Notepad

The **RAM (Random Access Memory)** is where the computer stores information that it's actively using. If the CPU is the supervisor, RAM is the supervisor's desk. On that desk, the supervisor can quickly jot down notes, remember what they were working on, and pull up information they need right now.

RAM is called "random access" because the CPU can access any piece of information in RAM almost instantly, regardless of where it is. Compare this to a filing cabinet where you have to manually find the file you need... with RAM, it's instantaneous.

Here's the critical thing about RAM: it's **temporary**. When you turn off your computer, everything in RAM is erased. This is why you need to save your files before shutting down... because RAM only holds information that the computer is currently using or thinking about.

In our warren analogy, RAM is the supervisor's immediate workspace. They can quickly jot things down, reference them, erase them, and move on. It's not for long-term storage; it's for active work.

The amount of RAM matters. If a supervisor has a bigger desk, they can work on more things simultaneously. If your computer has more RAM, it can handle more applications running at the same time, or process larger files.

Storage (Hard Drive/SSD): The Warren's Archive

While **RAM** is temporary, **storage** is permanent. This is where all your files live: programs, documents, videos, photos, operating system files... everything.

There are two main types of storage in modern computers:

Hard Disk Drives (HDD): These use spinning magnetic disks, similar to how old record players work. Data is written to these spinning disks using a mechanical read-write head. They're slower but can be very cheap for large capacities. If you bought a computer 10+ years ago, it probably had an HDD.

Solid State Drives (SSD): These use flash memory, the same technology as USB drives and SD cards, but faster and more sophisticated. There are no moving parts, so they're much faster and more reliable than HDDs. Almost all modern computers come with SSDs.

Think of storage as your warren's extensive archive. Unlike the supervisor's desk (RAM), the archive is permanent. When you turn off the warren for the night, the archive is still there. When you turn it back on, all the files are still there waiting for you.

The key difference between RAM and storage: RAM is fast but temporary, storage is slow (comparatively) but permanent. The CPU works with data in RAM, but when you need to keep something around forever or want to move it between computers, you save it to storage.

Other Important Components

The Bus: This is the communication highway inside your computer. Just like the warren needs messengers to carry information between the supervisor and the work teams, computers need a bus to carry information between the CPU, RAM, storage, and other components. Modern buses are incredibly fast and move enormous amounts of data.

The Power Supply: Your computer needs electricity to run. The power supply converts electricity from your wall outlet into the precise electrical currents that the computer's delicate components need. It's like the energy source that keeps the warren functioning.

The Motherboard: This is the main circuit board that connects all the components together. It's the physical infrastructure that makes all the components work as a unified system. In our warren metaphor, it's the actual warren structure itself.

Input/Output (I/O) Devices: These are how the computer communicates with the outside world:

- **Input devices** like keyboards, mice, and touchscreens let humans give instructions to the computer
- **Output devices** like monitors, speakers, and printers let the computer communicate back to humans
- **Network devices** like WiFi cards and Ethernet ports let computers communicate with each other

In our warren, these are like the gates and communication posts that connect the warren to the outside world.

1.3 Binary: How Computers Actually Think

Here's where things get interesting. Computers don't think in English, or numbers as we understand them, or words. They think in **binary**: sequences of 0s and 1s.

But why? Let me explain with an analogy:

Imagine you have a bunny, and you want to communicate with it using only two signals:

- A short whistle = 0
- A long whistle = 1

Any message can be communicated using only these two signals: just string them together. "Long-short-long-long-short" could mean "go to the carrot field." "Short-short-long" could mean "come back home."

This is exactly how computers work. Inside a computer, each "signal" is represented by an electrical voltage:

- Low voltage = 0
- High voltage = 1

Billions of these tiny switches, turning on and off billions of times per second, create everything your computer does. Every email you send, every image you view, every video you watch… at the most fundamental level, it's just patterns of 0s and 1s being processed at incredible speed.

Bits and Bytes: The Bunny Warren's Language

A single 0 or 1 is called a **bit** (short for "binary digit"). It's the smallest unit of information a computer can work with. One bit can represent only two states: 0 or 1, yes or no, true or false.

But one bit isn't very useful for storing much information. So computers group bits together. The standard grouping is **8 bits = 1 byte**.

With 8 bits, you can represent 256 different values (from 0 to 255). Why? Because there are 256 possible combinations of 0s and 1s when you have 8 binary digits:

- 00000000 = 0
- 00000001 = 1
- 00000010 = 2
- …
- 11111111 = 255

This might seem arbitrary, but it's incredibly useful. For example:

- One byte can represent a single letter or character
- One byte can represent a color shade (0-255, black to white)
- Two bytes can represent a number from 0 to 65,535
- Four bytes can represent a much larger number

In our warren analogy, a **byte is like a unit of instruction**. A single bit is too simple (just a yes/no), but a byte gives you enough complexity to represent something meaningful… a letter, a simple command, a small number.

Data Sizes: Building Bigger Units

Just like how the warren might have different storage scales, computers have different size units:

UnitSizeRoughly Equivalent To**Kilobyte (KB)**1,000 bytesA short text document**Megabyte (MB)**1,000 KBA photograph**Gigabyte (GB)**1,000 MBA movie**Terabyte (TB)**1,000 GBA large video library

When you see that your computer has "8 GB of RAM," that's 8 billion bytes of temporary storage. When you see a movie file is "4 GB," that's 4 billion bytes of storage space it will take up on your hard drive.

Why Binary? An Important Question

You might be wondering: "Why do computers use binary? Why not use base-10 (like we do with numbers 0-9)?"

The answer is physics. Binary is the easiest and most reliable way to represent information electrically. An electrical switch is either on or off, not on-or-in-between-or-somewhat-on. Binary maps perfectly to this physical reality: 0 = off, 1 = on.

If computers tried to use base-10 (representing 10 different voltage levels), it would be much harder to distinguish between them reliably. Is this voltage 3.5 volts (representing the digit 3) or 4.2 volts (representing the digit 4)? The difference is tiny, and noise or slight changes in the system could cause errors. With binary, you only need to distinguish between two states, which is robust and reliable.

So binary isn't a choice, it's fundamentally how electronics work. And it turns out that binary is just as expressive as any other number system. You can represent any number, any text, any image in binary. It's just a different language for the same information.

1.4 The Operating System: The Warren's Government

Now that we understand the hardware (the physical components), we need to talk about **software... **the instructions that tell the hardware what to do.

The most important software on any computer is the **operating system (OS)**. This is the master control system that manages everything else. If the hardware is the physical warren, the operating system is the government that runs it.

Think of an operating system as being responsible for:

Managing Resources: The OS decides which programs get CPU time, how much RAM each program can use, how programs access storage, etc. Just like a warren manager allocates resources to different tasks, the OS allocates computer resources to different programs.

Providing Services: The OS provides useful services that programs can use. For example, programs don't directly talk to your printer; they ask the OS to print something, and the OS handles the details of talking to the printer. This abstraction makes life much easier for programmers.

Managing Security: The OS enforces permissions. It makes sure that one program can't secretly access another program's data, and that programs can't do things they're not supposed to do.

Managing Files: The OS maintains the file system… the organized structure of folders and files on your storage. When you save a file, you're not actually managing the bits and bytes yourself; the OS handles all that complexity.

Handling Input and Output: When you click your mouse or press a key, the OS detects this and passes the information to the appropriate program.

Common Operating Systems

The main operating systems you'll encounter are:

Windows: Made by Microsoft, it's the most popular OS for personal computers. Most office computers run Windows.

macOS: Made by Apple, it runs on Apple's computers (Macs). It's popular among creative professionals and developers.

Linux: An open-source OS that powers many servers and also runs on personal computers. It's very popular among developers and system administrators.

Android and iOS: Mobile operating systems for smartphones and tablets. They work on the same principles but are optimized for touch input and power efficiency.

For our purposes (learning frontend engineering), the specific OS doesn't matter. The concepts are the same.

The Warren Metaphor Extended

Let's extend our warren analogy to include the operating system:

- The **CPU** is the head supervisor
- The **RAM** is the supervisor's desk
- The **Storage** is the warren's archive
- The **Operating System** is the warren government that makes sure everything runs smoothly, manages resources, and prevents conflicts

When you want to accomplish a task, you don't directly tell the supervisor (CPU). You ask the government (OS) for what you need. The government figures out how to make it happen using the available resources.

1.5 Programs: The Warren's Daily Operations

A **program** is a sequence of instructions that tells a computer what to do. Programs are written by programmers (that will be you!) in programming languages, and then the computer executes them.

When you run a program (like opening an email client, or a game, or a web browser), here's what happens:

1. The operating system reads the program from storage
2. It loads the program into RAM
3. It passes control to the CPU, telling it to start executing the program's instructions
4. The CPU executes the instructions one by one (or in some cases, many in parallel)
5. The program uses services provided by the OS (drawing to the screen, reading from storage, sending data over the network, etc.)
6. When the program finishes or you close it, the OS reclaims the memory and resources it was using

Every program follows this pattern. From your perspective, opening a program feels instant, but in reality, all these steps are happening at incredible speed.

Processes and Threads: Warren Teams

When a program is running, the OS tracks it as a **process**. A process is an instance of a program that's currently executing. If you open two email windows, that's two processes.

Within a process, there can be multiple **threads**. A thread is like a mini-supervisor within the process that can execute code independently. A single-threaded program has one thread doing all the work. A multi-threaded program can do multiple things at once.

In our warren analogy: a **process** is like a team assigned to accomplish a task, and **threads** are individual bunnies within that team who can work on subtasks in parallel.

This is important for frontend engineering because browsers use multiple threads to keep your web page responsive. One thread might be handling your mouse clicks while another thread is loading images from the internet.

1.6 How Computers Process Information: The Fetch-Execute Cycle

Now let's understand what actually happens when a computer runs a program. The fundamental process is called the **fetch-execute cycle**:

1. **Fetch**: The CPU reads the next instruction from RAM
2. **Decode**: The CPU figures out what the instruction is asking it to do
3. **Execute**: The CPU performs the operation
4. **Store**: If necessary, the CPU stores the result somewhere (in RAM, a register, or storage)
5. **Repeat**: Move on to the next instruction

This cycle repeats billions of times per second. Every calculation, every decision, every action your computer takes boils down to this simple cycle repeated over and over, incredibly fast.

Let's trace through an example. Imagine a simple program that adds two numbers:

```
1. Load the number 5 into the CPU's working memory
2. Load the number 3 into the CPU's working memory
3. Add these two numbers together (8)
4. Store the result (8) back into working memory
5. Display the result
```

Here's what happens at the hardware level:

1. **Fetch**: CPU reads instruction 1 from RAM
2. **Decode**: This is a "load" instruction
3. **Execute**: CPU loads 5 into a special CPU memory location called a **register**
4. **Store**: 5 is now in the register
5. **Fetch**: CPU reads instruction 2 from RAM
6. **Decode**: This is a "load" instruction
7. **Execute**: CPU loads 3 into another register
8. **Store**: 3 is now in another register
9. **Fetch**: CPU reads instruction 3 from RAM
10. **Decode**: This is an "add" instruction
11. **Execute**: CPU adds the values in the two registers
12. **Store**: The result (8) is stored in a register or back to RAM

This whole sequence happens in microseconds. And your computer is doing this not just for one simple operation, but for billions of operations simultaneously as it runs your programs.

Registers: The CPU's Immediate Memory

We mentioned **registers** above. These are tiny memory locations built directly into the CPU. They're the fastest memory available to the CPU because they're physically part of the CPU itself. The CPU keeps intermediate values and current work in registers, then moves them to RAM when done.

Think of registers as the supervisor's clipboard... they can grab it, jot something down, reference it immediately, and then set it aside.

1.7 Speed and Performance: Why Timing Matters

Different components of your computer operate at different speeds. Understanding this is crucial for understanding why computers are designed the way they are.

The Speed Hierarchy

Here's roughly how fast different components are (measured in nanoseconds, i.e. a billionth of a second):

ComponentAccess TimeWhy?**CPU Registers**<1 nsPart of the CPU, zero distance**CPU Cache**~4 nsVery close to CPU**RAM**~100 nsConnected via bus**SSD Storage**~100,000 nsMust read from storage**HDD Storage**~10,000,000 nsMust wait for disk to spin and move**Network**~1,000,000,000 nsData travels over internet

This is crucial. Accessing data from RAM takes about 100 times longer than accessing data from registers. Accessing data from storage takes about 1,000 times longer than accessing from RAM. And accessing data over the network takes about 10,000 times longer than accessing from storage!

This hierarchy explains a lot of computer design:

Why we have multiple levels of memory: Registers are tiny but fast. RAM is bigger but slower. Storage is huge but much slower. The computer uses all three levels, moving data between them strategically to balance speed and capacity.

Why RAM is important: Programs that use RAM efficiently run much faster than programs that constantly access storage.

Why network latency matters so much: When your web page makes a request to a server, it might take a million times longer than accessing local data. Frontend engineers spend a lot of effort minimizing network requests.

Why caching is a thing: The computer stores frequently-used data in faster locations so it doesn't have to fetch it repeatedly from slow storage.

Bits Per Second: Network Speed

When computers communicate over networks (like when you browse the internet), the speed is measured in bits per second.

- **Kilobits per second (Kbps)**: Thousands of bits per second (old dial-up internet)
- **Megabits per second (Mbps)**: Millions of bits per second (typical home internet)

- **Gigabits per second (Gbps)**: Billions of bits per second (fast connections)

A typical home internet connection might be 100 Mbps, which means it can transfer 100 million bits per second. To download a 1 MB (8 million bits) file would take about 0.08 seconds, assuming no other overhead.

This matters for frontend engineering because when you load a website, every image, every JavaScript file, every stylesheet has to be downloaded over the network. If you're serving a 1 MB image on a slow connection, it might take several seconds to download, and the user will see a blank space where the image should be.

1.8 Networks and Communication: How Computers Talk to Each Other

A single computer is useful, but the real power comes when computers can talk to each other. This is where **networks** come in.

A **network** is simply a group of computers connected together so they can share information. The Internet is a global network of billions of computers.

Basic Network Concepts

IP Addresses: Just like you have a mailing address so people can send you letters, each computer on a network has an **IP address** so other computers can send it information. An IP address looks like this: `192.168.1.1`. Each number is between 0 and 255 (remember our byte discussion from earlier? 256 possible values).

Packets: When data travels over a network, it's broken into small chunks called **packets**. Each packet contains:

- The destination IP address (where it's going)
- The source IP address (where it came from)
- A chunk of the actual data
- Error-checking information so the recipient can verify the packet arrived intact

Think of a packet like a postcard: it has an address, return address, a small message, and a signature (the error-checking).

Routers: These are devices that forward packets between different parts of the network. When you send data, it doesn't go directly to its destination; it hops through several routers, each one deciding where to send it next to get it closer to the destination. A router is like a post office that reads the address on a packet and decides which outgoing mail slot to put it in.

The Internet: A Network of Networks

The Internet (capital I) is not one big network owned by anyone. It's a network of networks. Your computer connects to your ISP's (Internet Service Provider's) network. That network connects to larger regional networks. Those connect to even larger backbone networks that span continents.

Packets traveling from your computer to a website on the other side of the world will hop through many different networks and routers. Despite this complexity, the Internet routes the packets reliably and efficiently (usually!).

The Web: One Application on the Internet

It's important to understand: **the Internet is not the Web**. The Internet is the infrastructure—the billions of computers connected together. The Web is one application that runs on top of the Internet.

Other applications on the Internet include:

- Email
- Video streaming services
- Online games
- Video calls (like Zoom)
- File sharing services

Each of these uses the Internet infrastructure but in different ways. The Web uses HTTP/HTTPS protocol (we'll learn about this in Chapter 3).

Client-Server Model: Warren Communication

Most applications on the Internet use a **client-server model**:

- **Client**: Your computer (or more specifically, a program on your computer like a web browser)
- **Server**: A remote computer running server software that can serve information

When you request a website, here's what happens:

1. Your browser (the client) sends a request to the server (e.g., "Please give me the home page of example.com")
2. The server receives the request, processes it, and sends back a response (e.g., "Here's the HTML code for the home page")
3. Your browser receives the response and displays it to you

This is like the warren sending a message to another warren: "Can you send us some carrot seeds?" The other warren receives the request, gathers the carrot seeds, and sends them back.

1.9 Energy and Heat: The Physical Reality

We've been talking about computers in abstract terms, but they're physical machines that consume real electricity and generate real heat.

Every operation a computer performs requires electrical energy. When the CPU executes billions of instructions per second, it's consuming significant power. This energy gets converted to heat, which is why computers have fans and cooling systems.

This has several implications:

Power consumption matters: Laptop and smartphone batteries only last a few hours because running billions of computations per second drains a lot of energy. This is why programmers optimize for performance... more efficient code means less energy consumption.

Cooling is necessary: A modern CPU can generate significant heat. Without proper cooling (fans, heat sinks), a computer will overheat and shut down or

be damaged. Data centres (huge facilities full of servers) spend enormous amounts of money on cooling because they have thousands of computers running simultaneously.

Environmental impact: Every computation, every website load, every email sent consumes electricity. Global data centres consume about 4-5% of the world's electricity. As frontend engineers, we should be aware that our code has an environmental cost and try to write efficient code.

The Warren's Energy Source

In our warren analogy, think of electrical power as the energy that keeps the warren functioning. Without it, nothing happens. More powerful computers need more power (bigger farms to grow food to sustain more bunnies). And just like physical activity generates heat in a bunny, computational activity generates heat in a computer.

1.10 Reflection: Why This Matters for Frontend Engineering

You might be wondering: "I want to build websites. Why do I need to know all this stuff about CPUs and RAM and network packets?"

Here's why:

Performance Awareness: When you understand how computers work, you understand why certain practices are important. Why is it bad to load huge image files? Because it takes time for them to download over the slow network. Why is it important to minimize JavaScript? Because parsing and executing JavaScript takes CPU time. Why should you be careful with animations? Because they take CPU time and drain battery on mobile devices. Understanding the "why" makes you a better engineer.

Debugging Skills: When something goes wrong with your website (and it will), understanding how computers work helps you figure out what's happening. Is the problem the CPU (too much calculation)? Is it the network (slow downloads)? Is it memory (too many things loaded)? Knowing how to ask these questions is crucial.

Design Decisions: As your projects grow more complex, you'll need to make decisions about architecture. Should you download all the data upfront or load it gradually? Should you do calculations on the client or send them to the server? Should you store data in RAM or storage? These decisions are impossible to make well without understanding how computers work.

Big Picture Understanding: Frontend engineering doesn't exist in isolation. You're writing code that runs on computers, communicates over networks, and is displayed on screens. All of this is built on the fundamentals we've discussed in this chapter.

Future Learning: Once you understand how computers work, learning about specific technologies becomes much easier. New frameworks, new tools, new languages... they all build on these fundamentals.

Key Takeaway

A computer is a machine that follows instructions to process information. It has hardware (CPU, RAM, storage, etc.) that performs operations, and software (operating system, programs) that tells the hardware what to do. Information is represented as binary (0s and 1s), organized into bytes and larger units. Programs are sequences of instructions that the CPU executes billions of times per second in the fetch-execute cycle. Different components operate at different speeds, and understanding this hierarchy is crucial for writing efficient code. Computers communicate with each other over networks using packets and IP addresses, creating the global infrastructure that enables the Internet and the Web.

In the next chapter, we'll explore how this global network of computers came to be, and how the Internet evolved into the amazing platform we use today. We'll meet the visionaries who created it, see how it changed the world, and understand why you're living in the most exciting time to build for the web.

Hands-On Exploration: Getting Comfortable with Your Computer

Before moving to the next chapter, spend some time exploring your computer's basic information. This will help ground the abstract concepts we just learned.

Task 1: Find Your Computer's Specifications

On Windows:

1. Right-click on "This PC" or "My Computer"
2. Select "Properties"
3. Look for:

    - Processor (your CPU)
    - Installed RAM
    - System type (32-bit or 64-bit)

On Mac:

1. Click the Apple menu (top-left)
2. Select "About This Mac"
3. Look for:

    - Processor (your CPU)
    - Memory (your RAM)

On Linux:

```
uname -a
grep -i "processor" /proc/cpuinfo
grep -i "memtotal" /proc/meminfo
```

Once you find this information, write it down. What CPU do you have? How much RAM? These are the actual resources your computer has available.

Task 2: Check Your Storage

On Windows:

1. Open File Explorer
2. Right-click on "C:" (or another drive)
3. Select "Properties"
4. Note the total size and free space

On Mac:

1. Click the Apple menu
2. Select "About This Mac"
3. Click "Storage"
4. See the total storage and what's using it

On Linux:

```
df -h
```

Note how much storage you have total and how much is being used. This is the size of your permanent storage.

Task 3: Monitor Your CPU and RAM Usage

On Windows:

1. Open Task Manager (Ctrl+Shift+Esc)
2. Click the "Performance" tab
3. Watch how CPU and Memory usage change as you open different applications

On Mac:

1. Open Activity Monitor (Applications > Utilities > Activity Monitor)
2. Watch how CPU and Memory usage change as you use different applications

On Linux:

```
top
```

Open a web browser and watch how RAM and CPU usage increase. Open another application and watch it increase more. Close them and watch it decrease. This real-time observation will help cement the concepts we discussed.

Task 4: Understand File Sizes

Open a few files on your computer and check their sizes:

On Windows/Mac:

- Right-click on a file and select "Properties" or "Get Info"
- Note the size (usually shown in KB, MB, or GB)

Common File Sizes:

- A text document: ~5-50 KB
- A photograph: ~2-5 MB
- A song: ~4-8 MB
- A movie: ~700 MB to 2 GB

Notice how text files are tiny compared to photos, which are tiny compared to videos? This relates back to the fact that a computer stores everything as bits and bytes—more complex media requires more bytes to represent.

Task 5: Open Your Developer Tools (Preview of Chapter 3)

We'll dive deep into browser developer tools in Chapter 3, but let's open them now just to see what they look like:

1. Open a web browser (Chrome, Firefox, Safari, or Edge)
2. Press `F12` or `Ctrl+Shift+I` (Windows) or `Cmd+Option+I` (Mac)
3. You'll see a panel open at the bottom of your screen with lots of tabs and information

Don't worry about understanding it yet. Just notice:

- The HTML code of the website you're viewing
- The network requests and their sizes
- The console where messages appear
- The timeline showing performance

We'll explore this deeply in Chapter 3, but for now, just notice that this window is showing you what's happening behind the scenes when you load a website.

Reflection Questions

After completing these tasks, think about these questions:

1. **CPU Cores**: You might see that your processor has multiple "cores" (e.g., "8 cores"). Why do you think computers have multiple cores? (Hint: Remember threads?)
2. **RAM Usage**: When you monitor RAM, you'll see that even with nothing open, a significant amount is being used. What do you think is using that RAM? (Hint: The operating system!)
3. **Network Requests**: When you opened the developer tools and looked at the network tab, you probably saw many requests. Why do you think a single website requires multiple requests to load?
4. **File Sizes**: Think about a movie file (2 GB). How many bytes is that? (Remember: 1 GB = 1 billion bytes) That seems enormous, but a movie can contain hundreds of thousands of individual images. How is this possible?

These questions will prepare you well for the chapters ahead. Don't worry if you don't know the answers yet... we'll explore them in detail.

Further Learning

While we've covered a lot in this chapter, there are deeper dives available if you're curious:

- **javascript.info's "JavaScript Fundamentals"**: While focused on JavaScript, they have good explanations of how browsers work at a hardware level
- **Khan Academy's "Introduction to Computer Science"**: Excellent free course covering these concepts with great visualizations
- **YouTube search "How Computers Work"**: There are excellent animated explanations by channels like Crash Course
- **Your Computer's Documentation**: Every computer comes with documentation about its specifications. Reading yours will ground these concepts in reality

Summary of Key Terms

TermDefinition**CPU**The processor; the brain that executes instructions**RAM**Temporary memory where the CPU works**Storage**Permanent memory (hard drive or SSD)**Bit**A single 0 or 1; the smallest unit of information**Byte**8 bits grouped together**Operating System**Software that manages hardware and provides services**Program**A sequence of instructions**Process**A running program**Thread**A mini-process within a program**IP Address**An address for a computer on a network**Packet**A small chunk of data sent over a network**Binary**A number system using only 0s and 1s**Network**Computers connected together**Internet**A global network of networks

You've now completed Chapter 1! You understand the fundamental hardware and software that powers every computer you'll ever interact with. In Chapter 2, we'll take this understanding and see how computers around the world connected to create the Internet and the Web... and how you can build amazing things on top of this infrastructure.

Chapter 2: The Origins of the Internet and the Birth of a Connected World

Introduction

In Chapter 1, we learned how computers work… the nuts and bolts of processors, memory, networks, and bits. Now we're going to see what happened when thousands, then millions, then billions of these computers started connecting to each other.

This is where it gets exciting.

The story of the Internet is not a dry technical history. It's a story of vision, competition, accidents, brilliant decisions, and missed opportunities. It's a story of people like you… engineers and entrepreneurs… who saw possibilities and built them. And here's the remarkable part: the web as we know it today was built by regular people solving problems they cared about. There was no master plan. There was no company that "invented the Internet." It emerged organically from thousands of people collaborating and competing.

Understanding this history will do three things for you:

1. **It will inspire you**: You'll see that the Internet wasn't built by giants sitting in boardrooms. It was built by engineers like you, working on interesting problems. Many of them started in garages, dorm rooms, and small offices. If they could build the Internet, what can you build?
2. **It will teach you patterns**: The Internet's history shows us why certain architectural decisions were made and how they've evolved. Understanding why we do things the way we do helps us make better decisions.
3. **It will humble you**: You'll see how even the smartest, most successful people and companies have missed opportunities and made strategic errors. Yahoo had the entire Internet in its hands and missed Google. Microsoft had market dominance and lost it partly due to overconfidence. These lessons are invaluable.

So, let's travel through time. Let's meet the people who built the Internet. And let's see how a few key moments shaped the world we live in today.

2.1 Before the Internet: The Isolated Computer

To appreciate what the Internet did, we need to understand what came before it.

For the first few decades of computers (1950s-1970s), computers were isolated machines. They were massive… room-sized machines that filled entire buildings. A company might have one computer in their computing centre, and it was used for critical calculations: payroll, accounting, important scientific work.

If one office building had a computer that needed to send information to another office building across town, someone would have to:

1. Prepare the information on the first computer
2. Put it on magnetic tape or punch cards
3. Physically transport the tape or cards by hand or mail
4. Load it into the other computer

Also called "sneakernet"… the data travelled by foot (sneakers) rather than electronically.

Scientists and researchers wanted to share data more efficiently. Some universities and research institutions connected their computers with telephone lines, but this was expensive and slow. The question everyone was asking was: "Can we build a network that allows computers to share information reliably and efficiently?"

In the early 1960s, researchers at the RAND Corporation were asking this question with a specific concern: What if a nuclear war destroyed parts of the telephone network? Could a network be designed that could survive partial destruction and still function?

2.2 ARPANET: The First Network (1969)

In 1969, the U.S. Department of Defense funded a project called **ARPANET** (Advanced Research Projects Agency Network). It connected four computers at four different universities:

1. University of California, Los Angeles (UCLA)
2. Stanford Research Institute (SRI)
3. University of California, Santa Barbara (UCSB)
4. University of Utah

On October 29, 1969, a researcher named Charley Kline at UCLA sent the first message over ARPANET. He was trying to send the word "LOGIN" to the computer at Stanford. The system crashed after transmitting "LO"... just two characters. But those two characters were historic. For the first time, information had been sent from one computer to another through a network.

This wasn't the Internet yet. It was just four computers. But it was the seed from which everything else would grow.

The Key Innovation: Packet Switching

The crucial innovation that made ARPANET work was **packet switching**. Remember from Chapter 1 how we talked about packets? This is where the concept was born.

Instead of establishing a single continuous connection between two computers (like a telephone call), packet-switching broke messages into small chunks (packets), each with a destination address. These packets could take any available route to reach their destination, and the receiving computer would reassemble them.

This had a huge advantage: if one part of the network went down, packets could find alternate routes. The network was resilient. This solved the Department of Defense's original question: yes, a network could survive partial destruction.

From ARPANET to the Internet

Over the 1970s, ARPANET grew. Other networks were created independently: CSNET (Computer Science Network), BITNET (Because It's Time Network), and others. Each had its own design, its own rules, its own way of sending information.

Scientists realized that all these separate networks could be even more powerful if they could talk to each other. The problem was that they used different protocols... different languages, essentially.

In 1973, researchers Vint Cerf and Bob Kahn designed a new protocol called **TCP/IP** (Transmission Control Protocol / Internet Protocol) that could work across different networks. This protocol became the universal language that allowed different networks to interconnect.

By 1983, ARPANET switched to TCP/IP, and this is often considered the birth of the modern Internet. Now you had a network of networks, all speaking the same language, all able to communicate. The Internet was born.

By the late 1980s, the Internet connected hundreds of thousands of computers. But it was still primarily used by researchers, scientists, and a few government agencies. Most people had no idea it existed.

2.3 The World Wide Web: Tim Berners-Lee's Vision (1989-1991)

Now here's where it gets really interesting. The Internet existed, but it was complicated. To access information, you needed to know computer commands. It was a tool for technical experts.

In 1989, a British scientist named **Tim Berners-Lee** was working at CERN (the European Organization for Nuclear Research) in Switzerland. He was frustrated. CERN had many scientists from around the world working on the same projects, but there was no easy way to share information between them. Different computers, different systems, different ways of organizing information.

Berners-Lee had a vision: what if there was a simple system for publishing and accessing information that could work across the network? What if you could create documents that contained links to other documents, creating a "web" of interconnected information?

He designed three things:

1. **HTML (HyperText Markup Language)**: A simple language for writing documents with links
2. **HTTP (HyperText Transfer Protocol)**: A protocol for requesting and sending these documents over the network
3. **The Web Browser**: A program that could request documents and display them to humans

On December 25, 1990, Berners-Lee wrote the first web browser. He was the first person to use it. He viewed the first website (which was documentation about the World Wide Web itself, running on his computer at CERN).

But here's the crucial part: Berners-Lee did something remarkable. He made these technologies **free and open**. He didn't try to patent them, didn't try to charge for them. He released them freely to the world.

This decision was monumentally important. If Berners-Lee had tried to commercialize these technologies, they might have failed. Instead, by making them free and open, he enabled anyone to build on them, improve them, and create the web as we know it today.

The First Website (1991)

On August 6, 1991, Berners-Lee made the first website publicly available. It was incredibly simple—just a page with links to other pages. But it was revolutionary.

The URL was: `http://info.cern.ch/hypertext/WWW/TheProject.html`

That URL tells us something important: this was at CERN, it was in a folder called "hypertext," in a folder called "WWW," and the file was called "TheProject.html"

The first website looked like this (roughly... no fancy graphics, no colors, just text):

```
WORLD WIDE WEB

The World Wide Web (W3) is a wide-area hypermedia information
retrieval initiative aiming to give universal access to a large
universe of documents.

Everything there is online about W3 is linked directly or indirectly
to this document.

What's out there?

   Pointers to the various nodes of the information web:

   Technical
   - Getting started with the Web
   - Web Browsers
   - How to write Web documents
```

```
    - CGI - Common Gateway Interface specification

    References
    - Hypertext
    - WWW project
    - Line Mode Browser
```

It was beautiful in its simplicity. No JavaScript, no CSS, no fancy design. Just text and links. And that was enough to revolutionize how information was shared.

Why the Web Exploded

Several things happened almost immediately that made the web explode in popularity:

1. **It was simple**: Unlike other systems that required technical expertise, anyone could create a web page. You just needed to know a bit of HTML, which was incredibly simple.
2. **It was visual**: Early web browsers made it easy to see what you were creating. You write HTML, you hit refresh, you see the result. This immediate feedback loop was powerful.
3. **It was free**: Just like the underlying technologies, browsers were free and easy to use.
4. **It was connected**: The web made it easy to link between documents on different computers. This web of connections made information discovery and sharing natural.
5. **It solved real problems**: Businesses realized they could use the web to distribute information, take orders, and connect with customers.

By 1993, there were about 50 websites in the world. Most people had never heard of the Internet.

By 1994, there were tens of thousands of websites.

By 1995, there were millions.

The exponential growth was underway.

2.4 The Early Web (1993-1995): Wonder and Possibility

When the web first became available to the public (around 1993), it was a time of pure wonder. Most people had never used the Internet before. You had to have a computer, a modem (a device that converted digital data to sounds that could travel over phone lines), and an Internet Service Provider (ISP) account.

Connecting to the Internet meant dialing up with a modem. You'd hear the screech and the beeping sounds, see the lights blink on your modem, and then… slowly… your browser would connect.

Then you'd wait. Page loads took minutes, not seconds. Images were especially slow… they would download slowly from top to bottom on your screen.

But the wonder of it! You could access information from anywhere in the world. You could see what people halfway across the globe were publishing. The world felt smaller and more connected.

Mosaic and the First Graphical Browser (1993)

In 1993, a researcher named Marc Andreessen was working at the National Center for Supercomputing Applications (NCSA) at the University of Illinois. He and his team created **Mosaic**, the first graphical web browser.

Before Mosaic, web browsers were text-based. You could see text and links, but not images in a nice way. Mosaic changed everything. It could display text, images, and links together in a beautiful way. It had buttons for "back" and "forward." It had a progress bar showing download progress.

Mosaic was revolutionary because it made the web accessible to regular people. You didn't need to be technical. You could click and explore.

Here's a remarkable detail: Marc Andreessen, who created Mosaic, later founded **Netscape**, which created the **Netscape Navigator** browser… the first commercially successful browser. But more importantly, he wrote on the Mosaic documentation: "There is no reason for the average person to use the Internet or a personal computer. There is no reason why anyone would want a computer in their home."

He said this about a technology that would change the world and that he himself was helping to create. This shows how even the smartest people sometimes can't predict which innovations will matter.

The Early Websites

The first websites were created by universities, government agencies, and early enthusiasts. NASA created a website. The Library of Congress started digitizing information. Individuals created personal pages with information about themselves, their interests, their families.

One of the most famous early websites was the **"Dancing Baby"** page. Someone uploaded an animated GIF of a baby dancing, and it became wildly popular. People linked to it. People shared the URL. It became viral by 1995 standards. It seems silly now, but it demonstrated the social nature of the web… people wanted to share interesting things.

Another early website was the **"Net Nanny" webcam**, where someone pointed a camera at their desk and uploaded a picture every few minutes. People would check throughout the day to see if anyone was at the desk. It seems mundane by today's standards, but then it was amazing: you could see a live picture from someone's office thousands of miles away, updated throughout the day.

The Business Realization

By 1994-1995, businesses were realizing that the web was not just a fad. Companies started rushing to create websites. They weren't entirely sure what their websites should do, but they knew they needed them.

A company called **InterNIC** registered domain names. The first .com domain was registered in 1985 (Symbolics.com). By the mid-1990s, companies were fighting over the best domain names.

Advertising appeared. The first online advertisement (a banner ad) was placed on HotWired.com in 1994. It advertised AT&T and had a 44% click-through rate. People were so intrigued by advertising on the web that they clicked on it! Advertisers realized they could reach customers online.

E-commerce emerged. Amazon was founded in 1994. eBay (originally AuctionWeb) launched in 1995. These companies realized that the web wasn't just for information… you could sell things through it.

The Dot-Com boom was beginning, though people didn't realize it yet.

2.5 Yahoo: The Home of the Internet (1994-2000)

As the web exploded with thousands, then millions of websites, a problem emerged: how do you find anything?

There was no Google yet. No search engine that really worked. The few that existed used basic keyword matching and weren't very helpful.

In 1994, two Stanford students named **Jerry Yang** and **David Filo** created a solution. They made a directory of websites... a human-curated list, organized by category. They called it "Jerry's Guide to the World Wide Web," but they renamed it to **Yahoo!** (which stood for "Yet Another Hierarchical Officious Oracle," though that's just a backronym... they just liked how it sounded).

Yahoo wasn't a search engine. It was a directory. You could browse categories:

```
textYahoo!
├── Arts
├── Business and Economy
├── Computers and Internet
├── Education
├── Government
└── Health
    ├── Alternative Medicine
    ├── Dentistry
    ├── Medicine
    └── ...
```

For each category, Yahoo had real humans who had visited and evaluated websites, and then organized links to the best ones. It was quality control in an era of exponentially growing chaos.

Yahoo's Dominance

Yahoo became the default starting point for people using the web. When you got online, you went to Yahoo. You'd browse their categories to find websites. If you wanted to find a website about a specific topic, you'd go to Yahoo and browse or search within their directory.

Yahoo was so important that it was featured on the "home page" of every major browser. When you opened Netscape Navigator or Internet Explorer, Yahoo was the default starting page.

The company grew explosively. In 1996, Yahoo went public, and the stock soared. The company was valued at billions of dollars.

Yahoo's Expansion: From Directory to Portal

Yahoo expanded rapidly throughout the late 1990s, transforming from a simple directory into a comprehensive "portal"... a one-stop shop for everything on the web. The company added:

- **Yahoo Mail** (1997): Free email service that became one of the most popular in the world
- **Yahoo Messenger**: Instant messaging to compete with AOL's AIM
- **Yahoo News**: Aggregated news from various sources
- **Yahoo Finance**: Stock quotes and financial information
- **Yahoo Groups**: Community forums and mailing lists
- **Yahoo Shopping**: Online shopping services

The strategy was clear: keep users on Yahoo as long as possible. If users came to Yahoo for search, stayed to check email, read news, and chat with friends, Yahoo could show them more ads and generate more revenue.

This portal strategy worked incredibly well throughout the late 1990s. Yahoo became truly the "home of the internet"... the first place people went when they opened their browser. Millions of people started every single internet session at Yahoo.

The Limitations of a Directory

But Yahoo's strength was also its weakness. A human-curated directory doesn't scale. As the web grew to millions of websites, thousands of new ones being added every day, it became impossible for humans to keep up.

By 1998, there were millions of websites, and Yahoo's directory had only cataloged a fraction of them. When you searched Yahoo for something specific, often you wouldn't find it. The directory was becoming less and less useful.

This limitation created an opening. And in that opening, something new emerged.

2.6 Google: The Search Revolution (1998)

Two Stanford PhD students, **Larry Page** and **Sergey Brin**, were working on a research project about analyzing the structure of the web. They noticed something interesting: websites that were linked to by many other websites were probably important. If many websites linked to you, you were probably worth visiting.

They developed an algorithm called **PageRank** (named after Larry Page, not "page rank" in general... a fortunate naming coincidence). The algorithm ranked websites based on how many other websites linked to them. More links = higher rank = probably more important.

They built a search engine called **BackRub** that used this algorithm. It worked remarkably well. When you searched for something, instead of getting results based on keyword matching (which often returned irrelevant results), you got results ranked by importance.

In 1998, they renamed the project **Google** (a play on "Googol," the number 1 followed by 100 zeros, representing the vast amount of information on the web).

Google's strength was simplicity and relevance:

Simplicity: The Google homepage was famously bare. Just a search box and a button. Compare this to Yahoo's portal, which was cluttered with categories, news, email, and dozens of other features.

Relevance: Google's PageRank algorithm actually worked. When you searched for something, you got useful results. The first result was usually what you were looking for.

The Technical Innovation

While PageRank was clever, there was another technical innovation that made Google work: **infrastructure**.

Google was built to scale. Instead of relying on expensive mainframe computers (which is how other companies at the time worked), Google used a bunch of cheap, commodity computers working together. They built systems that could automatically distribute work across many machines, automatically recover if a machine failed, and grow by simply adding more machines.

This approach was revolutionary at the time. Traditional database and search companies thought Google couldn't possibly work. But Google proved that you could build massive systems using cheap, commodity hardware and smart algorithms.

Google's Growth

Google was launched in 1998, but it wasn't immediately obvious that it would dominate. Several other search engines existed: Altavista, Lycos, Excite, Ask Jeeves.

But Google was better. And the web was growing. More searches were being done. More data. And Google's algorithm and infrastructure handled it better than anyone else.

By 2000, Google was processing a large percentage of web searches. By 2001, Google had become the dominant search engine.

2.7 The Missed Opportunity: How Yahoo Lost the Internet

Here's where the story gets interesting and sad for Yahoo.

In 2002, Google approached Yahoo with a proposal: Yahoo could use Google's search technology instead of building its own. Google would power Yahoo's search, and Yahoo would display Google's ads.

Yahoo's leadership said no.

Why? They thought they could build search themselves. They thought their directory approach was still viable. They thought their portal approach… being a one-stop shop for everything… was the future.

They were wrong.

Throughout the late 1990s and early 2000s, Yahoo's focus was on being a portal. Email, instant messaging, news, weather, shopping… they wanted to own everything. But people didn't want a portal. They wanted to search. They wanted a fast way to find what they were looking for.

Meanwhile, Google stayed focused on one thing: *search*. And they did it better than anyone.

By 2004, Google went public. The IPO was historic. The company was valued at about $23 billion. Investors saw Google's potential.

Yahoo's value was declining. Their search engine was worse than Google's. Their portal approach wasn't resonating with users. Users were spending less time on Yahoo and more time searching on Google.

In 2006, Yahoo finally made a move. They **bought Overture**, a company that had built an advertising system. They also tried various search initiatives. But it was too late. Google had built an unassailable lead in search. And search was the future.

The Greatest Missed Opportunity

But there's an even greater missed opportunity in this story.

In 1998, **Google was a startup with no revenue**. Yahoo could have bought Google for a few million dollars (or maybe a few billion at most). Yahoo had the resources, the users, the infrastructure.

If Yahoo had recognized Google's potential and bought them, the company could have dominated the 2000s and 2010s. Instead, Yahoo let Google slip through their fingers.

Yahoo tried to buy Google in 2002, when the company was already dominant, and the price was much higher. By then, it was too late.

Google's Other Great Advantage: Marissa Mayer and the Importance of Excellence

One of the most important figures in Google's early success was **Marissa Mayer**, Google's first female engineer (employee #20). She joined Google in 1999, right after the company was founded.

At Google, Mayer was instrumental in:

- **Keeping Google's homepage famously simple and fast**: While other companies cluttered their homepages with features, Mayer fought to keep Google's homepage bare... just a search box. She understood that every added feature slowed down the page load, and in the early days of slow internet, speed was crucial.

- **Leading the development of key products**: She led projects like Google Maps, Gmail, and Google News, which became essential products for billions of users.
- **Championing user experience and design**: While engineers often want to add more features, Mayer pushed for elegance and simplicity. She made sure Google products were beautiful and easy to use.
- **Fighting against feature creep**: She was famous for saying "no" to ideas that would make products worse, even if they seemed clever. This disciplined approach to product development was key to Google's success.

She became one of Google's most visible executives and was a key part of Google's dominance throughout the 2000s. Mayer represented the kind of talent and vision that Yahoo was missing.

The Contrast: Marissa Mayer was at Google during its crucial early years (1999-2012), helping build one of the most successful companies in history. Only much later, in 2012, did she join Yahoo as CEO, when the company was already in decline. By then, her talent couldn't save a company that had missed the fundamental shift in how the internet was used.

This is a crucial insight: Yahoo didn't just miss Google as a competitor. They missed being part of Google. The talent, the focus, the simplicity of approach... all of these were available to Yahoo, but they chose a different path.

What Happened to Yahoo?

Yahoo's story in the 2000s is a story of decline. They went from being the "home of the internet" to becoming a company searching for relevance.

In 2016, Verizon bought Yahoo for $4.5 billion. This was notable because in the late 1990s, Yahoo's market value was around $100 billion. From $100 billion to $4.5 billion.

In 2021, Verizon sold off Yahoo to a private equity firm.

What was once the king of the internet was now a footnote in history.

Why Did Yahoo Miss Google?

This is a crucial lesson for entrepreneurs and engineers. Why did the most dominant company in web miss the opportunity to own the future?

Several reasons:

Complacency: Yahoo was winning. They were the leader. Sometimes when you're winning, you stop paying attention to what's coming next.

Organizational Inertia: Yahoo had built a huge business around the portal concept. Admitting that Google's search-first approach was better would require admitting they were wrong. Changing direction would have been disruptive.

Not Invented Here: Yahoo had invested in building their own search technology. There's a tendency in big companies to prefer building things internally rather than using external solutions, even if the external solution is better.

Wrong Metrics: Yahoo's leadership was probably watching metrics like "time on site" and "pages per visit." Users stayed longer on Yahoo's portal and visited more pages. But the metric that actually mattered was "did we help users find what they wanted?" On that metric, Google was winning.

The Curse of Incumbency: The company that wins in one era often struggles in the next because they're optimized for the previous era. Yahoo was optimized for the web of 1998. Google was optimized for the web of 2004 and beyond.

This story is worth understanding because it shows that even dominant companies with brilliant people and plenty of resources can miss the future. Staying ahead requires constant vigilance, humility, and willingness to cannibalize your own business model for something better.

2.8 The Dot-Com Bubble and What Survived (2000-2002)

The 1990s saw explosive growth in internet companies. The "Dot-Com Boom" was on. Every entrepreneur wanted to start an internet company. Companies with no revenue, no profit, no clear business model were getting billions in funding.

Some examples of the insanity:

- **Pets.com**: An online pet store that burned through tens of millions of dollars before collapsing. They became famous for their "Sock Puppet"

mascot, which cost more to keep on air than the company made in revenue.
- **Webvan**: An online grocery delivery service that spent over $300 million before going bankrupt without ever achieving profitability.
- **Flooz.com**: An online currency that... actually, nobody really understood what they did before they collapsed.

The bubble was inevitable. Companies without sustainable business models can't survive forever. Eventually, reality catches up.

In 2000, the bubble burst. Stock prices collapsed. Companies that had been valued at billions were worthless. Thousands of startups went bankrupt. People called it the "Dot-Com Crash."

This was devastating for the internet industry. But here's what's important: the companies that **did** have real value and sustainable business models survived and thrived.

Google survived because they had a search product people loved and an advertising model that worked.

Amazon survived because they were building a real e-commerce business, taking customer orders, shipping products, and developing logistics systems.

eBay survived because people actually used their platform to buy and sell things.

Yahoo survived, though they were wounded.

The crash killed the bad companies but didn't kill the web. In fact, it made the web stronger by clearing out the garbage and letting the real innovations flourish.

2.9 The Modern Web Emerges (2000s-Present)

After the Dot-Com crash, the internet settled down and got to work. What emerged was the foundation of the modern web.

Broadband Becomes Standard (2000s)

Through the 2000s, internet speeds improved dramatically. Dial-up (56k modems, taking minutes to download a single image) gave way to broadband

(ADSL and cable, hundreds of times faster). By the late 2000s, most people in developed countries had fast internet.

This speed enabled new possibilities:

- **Video streaming**: YouTube launched in 2005, enabled by fast enough speeds for video
- **Rich applications**: Web applications became more complex and interactive
- **Social networks**: Bandwidth-heavy features like photo uploads and instant notifications became feasible

Web 2.0 and User-Generated Content (2004-present)

The term "Web 2.0" was coined around 2004. It didn't mean a new version of the technology (HTML, HTTP, etc. didn't change). It meant a shift in how the web was being used.

In Web 1.0 (the 1990s), the web was about publishing information. Companies and organizations created websites and published information that people could read.

In Web 2.0, users became creators. **Blogs** let anyone publish. **Flickr** let people share photos. **YouTube** let people share videos. **Wikipedia** let people collaborate to create an encyclopedia. **MySpace** and later **Facebook** (anyone remembers **Orkut**?) let people create profiles and connect with others.

The pivot was from publishing to participation. The web became not just a place to read information but a place to create and share.

Mobile and Apps (2007-present)

In 2007, Apple released the iPhone. This was a shift as big as the invention of the web itself.

For the first time, most people had a powerful computer in their pocket. And it was designed for touch, not keyboards and mice. The web, which had been designed for desktop computers, had to adapt.

At first, people thought mobile would have "apps"... separate software applications, not web-based. And that did happen. But the web adapted too.

Responsive design became crucial—websites that could adapt to phone-sized screens. **Mobile-first** design became the norm—designing for mobile first, then scaling up to desktop.

Social Media and Real-Time Web (2006-present)

Twitter (now X) launched in 2006, introducing the real-time stream of short messages. Facebook, which started as a college social network in 2004, expanded to the world and became the largest social network ever.

These platforms didn't use the web's traditional architecture of pages and links. Instead, they used **APIs** (Application Programming Interfaces) to feed data to web applications and mobile apps. They used real-time technologies to push notifications to users.

Machine Learning and Personalization (2010s-present)

As companies collected more data about user behavior, they used machine learning to personalize experiences. Amazon recommended products you might like based on your browsing and purchase history. Netflix recommended shows and movies. YouTube recommended videos. Facebook showed you posts from people and pages you were most likely to interact with.

This changed the fundamental nature of the web. Instead of everyone seeing the same content, each person saw a personalized version tailored to them.

Cloud Computing and Serverless Architecture (2010s-present)

Amazon Web Services (AWS), launched in 2006, pioneered cloud computing… letting companies rent computing power instead of buying servers. This made it easy for startups to build and scale without huge infrastructure investments.

Later, serverless computing (Lambda, Cloud Functions) abstracted away even servers. You could write a function, upload it to the cloud, and it would run and scale automatically. The cloud had become the default way to deploy code.

The Modern Era (2020-present)

Today's web is incredibly sophisticated:

- **Progressive Web Apps** that work offline
- **WebAssembly** that lets you run compiled code in the browser at near-native speeds
- **GraphQL** for more efficient data querying
- **Real-time collaboration** with tools like Google Docs and Figma
- **Edge computing** that runs code closer to users for lower latency
- **AI and Machine Learning in the Browser** with TensorFlow.js

But the core principles remain the same: computers connected over networks, using HTTP to send data, displaying content in browsers using HTML/CSS/JavaScript.

2.10 You Can Build the Next Internet Giant

Here's the point of this entire chapter, and it's important:

Everything you see on the internet was built by people. Regular people. People who learned the technology, understood the problems, and built solutions.

Tim Berners-Lee didn't come from a famous tech company. He was a researcher at a physics lab who was frustrated with managing information, so he built a solution.

Larry Page and Sergey Brin were PhD students who noticed something interesting about how the web was structured and built a better search engine.

Mark Zuckerberg wrote Facebook in his Harvard dorm room.

Jack Dorsey and Evan Williams built Twitter because they thought the real-time web was cool.

You can do this too.

The barrier to entry is lower than ever:

- **Technology is free**: Open-source tools, free cloud computing tiers, free hosting services

- **Knowledge is free**: Everything you need to learn is available online
- **Distribution is free**: If you build something interesting, you can reach millions of people through social media and word of mouth
- **Capital requirements are low**: You don't need to build expensive infrastructure; you can use someone else's (AWS, Google Cloud, Azure)

What you do need:

1. **A problem you care about**: You notice something broken or inefficient
2. **Willingness to learn**: You have to be curious enough to figure out how to build the solution
3. **Persistence**: Building anything worthwhile is hard; you need to stick with it
4. **Basic engineering skills**: Which is exactly what this book is teaching you

Examples of Recent Internet Giants

Let's look at some recent examples of people who saw an opportunity and built something amazing:

Figma (2012-present): Dylan Field noticed that design tools were stuck in the past. Designers were using ancient software. He built Figma... a modern, web-based design tool where multiple people could collaborate in real-time. Today, Figma is valued at over $20 billion and is used by designers worldwide.

Stripe (2010-present): Patrick and John Collison noticed that accepting payments online was unnecessarily complicated. They built Stripe, making it easy for anyone to accept credit card payments. Today, Stripe processes hundreds of billions of dollars in payments and is one of the most valuable private companies in the world.

Notion (2016-present): Ivan Zhao built Notion as a tool to organize his thoughts and projects. He realized others had the same problem. Today, Notion is used by millions of people and companies for note-taking, databases, and project management.

TikTok (2016-present): Zhang Yiming noticed that short-form video was the future and built the algorithm and platform to deliver it. Today, TikTok is valued at over $100 billion and is the most downloaded app in the world.

All of these were built by people who:

1. Saw a problem
2. Had the technical skills (or learned them) to build a solution
3. Built something people wanted
4. Executed relentlessly

You have all the ingredients to do the same.

2.11 The Key Characteristics of People Who Build on the Internet

If you want to be the next person to build something that changes the world, what should you look like?

Curiosity: The most successful founders and engineers are endlessly curious. They ask "why" a lot. They read widely. They explore.

Stubbornness: When everyone said Facebook wouldn't replace Friendster, Zuckerberg kept building. When everyone said e-commerce was a bad business model, Bezos kept building Amazon.

Pragmatism: You don't need everything to be perfect. You build something that works, you ship it, you learn from users, you improve it. The idea that you need to get everything right before launching is paralyzing. The people who win ship early and often.

Empathy: You need to deeply understand your users. Not what you think they want, but what they actually want and need. The best products come from understanding users so well that you can anticipate their needs.

Willingness to look stupid: Asking basic questions ("How does email really work?" "Why is this so complicated?") can feel stupid in front of experts. But asking those questions is how you find the real problems to solve.

Community orientation: Many of the greatest things on the internet were built by communities, not individuals. The best engineers collaborate, share knowledge, and help others. The people who help others are the ones others help when they need it.

Key Takeaway

The Internet and Web were not invented by one company or one person. They emerged from thousands of people collaborating, competing, learning, and building. The web started as a simple solution to a specific problem (sharing documents at a physics lab) and evolved into a platform that changed the world.

The key lesson: **the future of the internet will be built by people like you**. The barrier to entry is incredibly low. The tools are free. The knowledge is available. The only thing you need is curiosity, the willingness to learn, and the persistence to build something worthwhile.

In the next chapter, we'll dive into how web browsers work... the software that powers everything you see on the internet. Understanding browsers will give you insight into how the code you write gets executed, how the browser protects you, and why browsers are some of the most sophisticated pieces of software ever created.

The internet was built for you. Now let's learn how to build with it. 🚀

Hands-On Exploration: Connecting with Internet History

Before moving to the next chapter, spend some time exploring the history of the internet and the web. This will ground these concepts and give you a deeper appreciation for what you're about to learn.

Task 1: Visit The Internet Archive's Wayback Machine

The **Wayback Machine** (archive.org) allows you to see how websites looked in the past. It's like a time machine for the web.

1. Go to https://web.archive.org/
2. Search for these classic websites and see how they evolved:

    - **Yahoo.com**: Search for early versions. Notice how cluttered the homepage became as they tried to be a portal.
    - **Google.com**: Search for early versions. Notice how simple it's always been.
    - **Amazon.com**: See how the design has evolved.

- **Facebook.com**: Notice how social features evolved.
- **YouTube.com**: See how video streaming design changed.

This hands-on experience will be more valuable than reading about it. You'll see the evolution of design and understand why certain patterns emerged.

Task 2: Explore Your Browser's History

Think about the websites you visit most frequently. Most of them are less than 20 years old:

- **Google**: Founded 1998
- **YouTube**: Founded 2005
- **Facebook**: Founded 2004
- **Twitter**: Founded 2006
- **Instagram**: Founded 2010
- **TikTok**: Founded 2016
- **Discord**: Founded 2015
- **Slack**: Founded 2013

These are the products that shape how we communicate, access information, and interact. All of them exist because someone noticed a problem and built a solution.

Task 3: Research the Founders

Pick 3 of your favorite websites or apps, and research their founders:

1. **Who created it?**: Who are they? What's their background?
2. **Why did they create it?**: What problem were they solving?
3. **When did it start?**: How long have they been around?
4. **How many people use it?**: What's the scale?

Look for interviews with the founders on platforms like:

- **YouTube**: Search "[Founder Name] interview"
- **Podcasts**: Search on Spotify or Apple Podcasts

- **Articles**: Medium, TechCrunch, The Verge

You'll be surprised at how normal these founders are. Many started in their dorm rooms, garages, or small offices. Many were told their idea would never work. Many of the most successful people in tech started with no special advantage except curiosity and persistence.

Task 4: Understand Business Models

Now that you understand some history, think about how these companies make money:

CompanyBusiness Model**Google**Advertising**Facebook**Advertising**Amazon**E-commerce + Cloud Services**Netflix**Subscription**Stripe**Transaction fees**Figma**Subscription

Pick 3 companies you're interested in and understand their business model. How do they get paid? Who pays them? Is their model sustainable?

This understanding will help you when you think about building your own products. You need not just a good idea, but a business model that can sustain it.

Task 5: Understand the Technology Stack

Pick one modern website and think about what technologies might power it:

For example, **Instagram**:

- **Frontend**: Web interface built with JavaScript (probably React)
- **Mobile**: Native iOS and Android apps
- **Backend**: Servers handling user requests, storing data
- **Database**: Storing millions of photos, user profiles, connections
- **Storage**: Cloud storage for all those billions of photos
- **Infrastructure**: Distributed servers across the world for low latency
- **Real-time**: Push notifications, live feeds

Don't worry if you don't understand these technologies yet. Just start thinking about the layers:

1. **Client layer** (what you see in your browser)
2. **Server layer** (code running on servers processing your requests)
3. **Database layer** (storing data persistently)
4. **Infrastructure layer** (managing where code runs)

By the end of this book, you'll understand the client layer deeply. You'll have a basic understanding of how it connects to the server layer. And you'll appreciate the overall architecture.

Task 6: Look for Problems to Solve

This is the most important task. As you use the internet and use apps and websites, start noticing problems:

- Is there a task that's annoying to do?
- Is there information that's hard to find?
- Is there a process that's inefficient?
- Is there something you wish existed?

Write down 5 problems you notice. Don't judge whether they're "good" problems or "bad" problems. Just notice them.

Examples:

- "It's hard to coordinate plans with friends because everyone's in different group chats"
- "I can't easily see which of my online friends are actually available to talk"
- "It's hard to find the specific information I need within long documents"
- "Every time I open this website, I have to scroll past the same ads to get to the content"
- "I spend too much time searching for music recommendations I actually like"

These problems are where ideas come from. These problems are where the next billion-dollar company will come from.

Reflection Questions

As you finish this chapter, think about these questions:

1. **If you had been Yahoo's CEO in 2002, what would you have done differently?** Would you have bought Google? How would you have responded to Google's growth?
2. **What problem could you solve for the internet?** What's something that frustrates you when you use the web?
3. **Why do you think it's easier to start an internet company today than it was in 1995?** What tools and resources are available now that weren't available then?
4. **Which of the modern internet giants (Google, Facebook, Amazon, Apple) do you think will be most important in 10 years? Why?** Which might decline?
5. **What would you have needed to believe in 1998 to have invested in Google or Amazon when they had no profit?** What does that tell you about taking risks?

Further Reading

If you want to dive deeper into internet history:

- **"The Innovators" by Walter Isaacson**: A comprehensive history of the digital revolution, including the internet and web
- **"Netscape Time" by Jim Clark**: The inside story of one of the first major internet companies
- **"The PayPal Wars" by Eric M. Jackson**: How payment systems drove e-commerce
- **"Shoe Dog" by Phil Knight**: Not about the internet, but a great example of founding and building a company
- **YouTube**: Search for "history of the internet" or "internet pioneers" for documentary-style videos
- **Tim Berners-Lee's TED talk**: He's given talks about the future of the web

Summary of Key Figures

PersonContributionKey Insight**Tim Berners-Lee**Invented the WebMade it free and open; this openness enabled it to succeed**Marc Andreessen**First graphical browser (Mosaic)Made the web accessible to non-technical people**Jerry Yang & David Filo**Founded YahooCreated the first useful way to find information**Larry Page & Sergey Brin**Founded GoogleSolved search through innovation in ranking algorithms**Marissa Mayer**Google engineer #20, later Yahoo CEOChampioned simplicity and user experience; showed how design choices matter**Steve Jobs**iPhoneChanged the paradigm from desktop to mobile**Mark Zuckerberg**Founded FacebookUnderstood that people want to connect socially**Evan Williams & Jack Dorsey**Founded TwitterRealized people want to share what's happening right now

You've now completed Chapter 2! You understand the history of the internet and web, the key players, the turning points, and the lessons about innovation and missed opportunities. You understand that the future will be built by people like you... people willing to learn, notice problems, and build solutions.

In Chapter 3, we'll dive into **how browsers work... **the software that brings the internet to life on your computer screen. You'll understand the browser wars, the challenges of supporting multiple browsers, and how the modern browser has become one of the most sophisticated pieces of software ever created.

Ready? Let's dive in...

Chapter 3: How Browsers Work

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Introduction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

3.1 What Is a Browser? A Bunny's Window to the Internet

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

3.2 The Browser's Journey: From URL to Rendered Page

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Step 1: You Type a URL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Step 2: DNS Lookup - Finding the Warren's Address

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Step 3: Establishing a Connection - TCP Handshake

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Step 4: Making the HTTP Request

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Step 5: Server Processing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Step 6: Parsing HTML - Building the DOM

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Step 7: Fetching Resources - CSS, JavaScript, Images

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Step 8: Building the Render Tree

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Step 9: Layout - Calculating Positions and Sizes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Step 10: Painting - Drawing to the Screen

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Step 11: Compositing - Combining Layers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Step 12: Display - You See the Page

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

3.3 The Critical Rendering Path and Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

3.4 The Browser Wars I: Netscape vs. Internet Explorer (1995-2002)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

The Rise of Netscape Navigator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Microsoft Enters: Internet Explorer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

The First Browser War

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

The "Browser Wars": Microsoft's Aggressive Tactics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

The Impact: Web Developer Hell

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Famous Browser Hacks for IE

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Netscape's Decline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

3.5 The Browser Wars II: Firefox, Safari, Chrome, and Standards (2002-Present)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Firefox's Rebellion (2004)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Apple's Safari (2003)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Google's Chrome (2008)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Internet Explorer's Decline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

The Web Standards Project

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

3.6 Internet Explorer: The Dark Ages for Web Developers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

IE's Problems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

IE 6: The Nightmare

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

IE's Legacy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

3.7 Modern Microsoft and Open Source: A Redemption Arc

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Satya Nadella's Transformation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

My Gratitude

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

3.8 Modern Browsers and Web Standards (2020-Present)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Browser Market Share (2026)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Modern Rendering Engines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

The W3C and Standards Bodies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Regular Release Cycles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Feature Parity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

3.9 The Browser as a Platform

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Progressive Web Apps (PWA)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Web APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

WebAssembly

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Web as a Universal Platform

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

3.10 The Browser as a Developer Tool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

The Inspector/Elements Tab

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

The Console

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

The Network Tab

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

The Performance Tab

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Other Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

3.11 Security and Privacy in Browsers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Same-Origin Policy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Content Security Policy (CSP)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

HTTPS and Encryption

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Password Storage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Sandboxing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Malware and Phishing Detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

3.12 Reflection: Why Understanding Browsers Matters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Making Good Decisions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

The Joy of Mastery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Key Takeaway

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Hands-On Exploration: Getting Familiar with Your Browser

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Task 1: Open Developer Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Task 2: Explore the Elements/Inspector Tab

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Task 3: Explore the Console Tab

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Task 4: Explore the Network Tab

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Task 5: Check a Slow Network

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Task 6: Inspect a Specific Website's Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Task 7: Look for JavaScript Errors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Task 8: Understand DNS

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Task 9: Trace a Network Request

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Task 10: Experiment with Changing Styles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Reflection Questions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Further Learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.

Summary of Key Concepts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/frontendforbunnies.