Contents

Chapter 1

Making Web Requests with HTTP

The Hypertext Transfer Protocol (HTTP) is the protocol used for transmitting hypertext over the World Wide Web. It is the foundation of any data exchange on the web and it is a protocol used for transmitting hypertext requests from clients (like a user's browser) to servers, which respond with the requested resources.

1.1 HTTP Requests

An HTTP request is made up of several components:

- **Method**: The HTTP method, like GET (retrieve data), POST (send data), PUT (update data), DELETE (remove data), etc.
- **URL**: The URL of the resource to retrieve, send data to, update or delete.
- **Headers**: Additional information about the request or response, like the content type of the body.
- **Body**: The body of the request, used when sending data in POST or PUT requests.

1.2 Using HTTP with Web-Based APIs

Software developers often use HTTP to interact with web-based APIs. An Application Programming Interface (API) is a set of rules that allows programs to talk to each other. The developer creates the API on the server and allows the client to talk to it.

When a developer makes a request to an API endpoint, they're asking the server to either send them some data or receive some data from them. The response from the server will often be in a format like JSON or XML, which the developer can then use in their own application.

For instance, a developer might make a GET request to '`https://api.example.com/users`' to retrieve a list of all users. The server would respond with a list of users in a format like JSON.

CHAPTER 2

Using and Creating APIs

As a software engineer, you are likely familiar with building applications that interact with various external services and data sources. One of the most common methods for communication and integration is through HTTP APIs (Application Programming Interfaces). HTTP APIs provide a standardized way for applications to exchange data and functionality over the internet.

This chapter will introduce you to the world of HTTP APIs and explore how you can leverage them in your software development projects. We will cover the fundamental concepts, techniques, and best practices for effectively working with HTTP APIs.

An HTTP API allows two software systems to communicate and exchange information using the Hypertext Transfer Protocol (HTTP). It enables your application to make requests to an API server and receive responses in a structured format, such as JSON (JavaScript Object Notation) or XML (eXtensible Markup Language).

Using HTTP APIs offers a range of benefits for software engineers. It allows you to leverage external services and data sources, enabling your application to access functionality or retrieve valuable information from third-party systems. This opens up opportunities for integration with popular platforms, social media networks, payment gateways, geolocation services, and much more.

Throughout this chapter, we will explore various aspects of working with HTTP APIs, including:

- API endpoints and methods: Understanding how to interact with an API involves identifying the available endpoints and the supported methods, such as GET, POST, PUT, DELETE, etc. We will discuss how to construct API requests and handle different response formats.
- Authentication and authorization: Many APIs require authentication to ensure secure access and protect sensitive data. We will delve into different authentication mechanisms, including API keys, tokens, OAuth, and other authentication protocols commonly used in API integrations.
- Request parameters and payloads: APIs often accept additional parameters or payloads to customize the request or send data for processing. We will explore how to pass query parameters, request headers, and request bodies when interacting with APIs.
- Error handling and status codes: Learning how to handle errors and interpret status

codes returned by APIs is crucial for building robust and resilient applications. We will discuss common status codes and best practices for handling various scenarios gracefully.

- Rate limiting and throttling: Many APIs impose restrictions on the number of requests you can make within a given timeframe to prevent abuse and ensure fair usage. We will cover techniques for handling rate limiting and implementing efficient strategies to manage API quotas.
- API documentation and testing: Proper documentation is essential for understanding an API's capabilities, endpoints, and expected behavior. We will explore how to read and interpret API documentation, as well as techniques for testing and validating API integrations.

By mastering the art of using HTTP APIs, you will expand your development toolkit and gain the ability to seamlessly integrate your applications with external services, leverage their functionalities, and build powerful, interconnected systems.

So, let's dive into the world of HTTP APIs and uncover the endless possibilities they offer for enhancing your software engineering projects.

Chapter 3

Data Compression and Decompression

Data compression and decompression are fundamental techniques used in modern computing, enabling efficient storage and transmission of data. The concept of entropy, borrowed from the field of information theory, plays a crucial role in determining the compression rate.

3.1 Data Compression and Decompression

Data compression is the process of reducing the amount of data needed to represent a particular set of information. The two main types of data compression are lossless and lossy. Lossless compression ensures that the original data can be perfectly reconstructed from the compressed data, whereas lossy compression allows some loss of data for more significant compression rates.

Decompression is the reverse process of compression, reconstructing the original data from the compressed format.

3.2 Entropy

In information theory, entropy measures the unpredictability or randomness of information content. More specifically, it quantifies the expected value of the information contained in a message. Lower entropy implies less randomness and more repetitiveness, which in turn means the data can be compressed more.

3.3 Entropy and Compression

The role of entropy in data compression is fundamental. The entropy of a source of data is the minimum number of bits required, on average, to encode symbols drawn from the source. It serves as a lower bound on the best possible lossless compression rate.

For a source X with probability distribution $p(x)$, the entropy $H(X)$ is defined as:

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x) \tag{3.1}$$

If the entropy of the data is high (i.e., the data is random and unpredictable), the potential for compression is low. On the other hand, if the entropy is low (the data is predictable), the data can be compressed to a smaller size.

Chapter 4

Dealing with JSON and XML

Chapter 5

HTML

HTML, an abbreviation for Hypertext Markup Language, is the standard language for creating web pages and web applications. It's a cornerstone technology of the World Wide Web and forms the structure and layout of web content.

5.1 HTML Elements

An HTML document is composed of a series of elements, which are denoted by tags. Elements have an opening tag and a closing tag with content in between. Some elements, however, are self-closing and do not contain any content. For example, the paragraph tag '<p>' is used to denote a paragraph:

```
<p>This is a paragraph.</p>
```

5.2 HTML Document Structure

A typical HTML document has a specific structure, including the following elements:

- **DOCTYPE declaration**: It informs the browser about the version of HTML. For HTML5, it is '<!DOCTYPE html>'.
- **html**: This tag encloses the entire HTML document.
- **head**: This contains meta-information about the document, such as its title, meta tags, and links to scripts and stylesheets.
- **body**: This contains the content of the web page that is rendered in the browser.

Here is a basic example of an HTML document:

```
<!DOCTYPE html>
<html>
<head>
    <title>My First HTML Page</title>
</head>
<body>
    <h1>Welcome to My First HTML Page!</h1>
```

```
    <p>This is a paragraph.</p>
</body>
</html>
```

Chapter 6

Introduction to Text

In computer systems, text is represented in files as a sequence of characters, each of which corresponds to a specific number known as a character code. These character codes are then stored in the file as binary data.

6.1 Newlines and Carriage Returns

Two of the character codes that have special meanings are the newline (often represented as '\n') and the carriage return (often represented as '\r').

The newline character signifies the end of a line of text and the beginning of a new one. The carriage return character moves the cursor to the beginning of the line. The use of these characters can vary between operating systems. Unix-based systems (like Linux and MacOS) use the newline character to indicate the end of a line, while Windows systems use a combination of a carriage return and a newline ('\r\n').

6.2 ASCII

The American Standard Code for Information Interchange (ASCII) is one of the earliest character encodings. It uses 7 bits to represent each character, allowing it to define up to $2^7 = 128$ different characters. These include the English alphabet (in both lower and upper cases), digits, punctuation symbols, control characters (like newline and carriage return), and some other symbols.

6.3 UTF-8

UTF-8 (8-bit Unicode Transformation Format) is a variable-width character encoding that can represent every character in the Unicode standard, yet remains backward-compatible with ASCII. For the ASCII range (0-127), UTF-8 is identical to ASCII. But it can use additional bytes (up to 4 bytes in total) to represent characters that are not included in ASCII, such as characters from other languages, emojis, and many other symbols. This has made UTF-8 a widely used encoding in many modern systems.

Chapter 7

Stop Words

CHAPTER 8

Stemming and Lemmatization

Stemming and lemmatization are two fundamental techniques in natural language processing that are used to prepare text data. They help in reducing inflectional forms of a word to a common base form.

8.0.1 Stemming

Stemming is the process of reducing a word to its word stem, i.e., its basic form. For instance, the stem of the word 'jumps' would be 'jump'. A stemming algorithm reduces the words "jumping", "jumped", and "jumps" to the stem "jump".

It's important to note that stemming may not always lead to actual words. For example, the stem of the word "running" could be "runn" depending on the stemming algorithm used.

Stemming is generally simpler and faster than lemmatization, but it is also less precise.

8.0.2 Lemmatization

Lemmatization, on the other hand, reduces words to their base or root form, which is linguistically correct. For example, "running" and "runs" are both changed to "run".

Lemmatization uses a more complex approach to achieve this: it considers the morphological analysis of the words and requires detailed dictionaries which the algorithm can look through to link the form back to its lemma.

To summarise, both stemming and lemmatization help in text normalization and preprocessing, but while stemming can be faster and simpler, lemmatization is more accurate as it uses more informed analysis to create groups of words with similar meanings based on the context.

Chapter 9

Alphabets and Accents

In today's interconnected world, software developers often encounter text from diverse languages and cultures. As a developer, it is crucial to have a solid understanding of alphabets and accents to effectively handle and process this multilingual text. Alphabets, the building blocks of written language, vary widely across different nations and regions. Meanwhile, accents, diacritical marks, and other phonetic notations play a crucial role in conveying the correct pronunciation and meaning of words.

This guide aims to provide software developers with a fundamental understanding of alphabets and accents to navigate the complexities of handling text from different nations. By familiarizing yourself with these concepts, you will be better equipped to develop robust applications, support multiple languages, and ensure accurate representation and interpretation of text data.

Alphabets are sets of letters or symbols used to represent the sounds of a language. While the Latin alphabet is widely used in many Western languages, numerous other alphabets exist, such as Cyrillic, Greek, Arabic, Devanagari, and Chinese characters. Each alphabet has its own unique set of letters, often organized in a specific order, and may include uppercase and lowercase variations.

Accents and diacritical marks are additional symbols added to letters to modify their pronunciation or provide additional phonetic information. Accents can appear above, below, or beside a letter, and they can change the sound, stress, or intonation of a word. For example, in French, the acute accent (’e) changes the pronunciation of the letter "e" from /□/ to /e/.

When working with multilingual text, it is essential to consider various factors:

1. Character encoding: Different alphabets require specific character encodings to represent their letters digitally. Commonly used character encodings include ASCII, Unicode, and UTF-8. Understanding the appropriate encoding for each language is crucial to ensure proper text rendering and avoid data corruption.
2. Text input and validation: Building applications that handle user input requires robust text validation. Account for the diverse set of characters and possible accents that may appear in user-generated content. Implement proper validation and sanitization mechanisms to handle text input securely.
3. Sorting and collation: Sorting text from different languages involves considering the specific rules and conventions of each alphabet. Some languages may have unique

sorting orders, while others ignore accents or diacritics when determining the order of words. Take into account the appropriate sorting and collation algorithms to ensure consistent and accurate results.

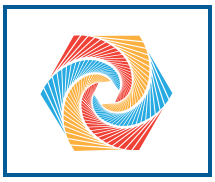
4. Search and indexing: Efficient search and indexing systems must be capable of handling multilingual text. Consider appropriate text normalization techniques to account for different character representations (e.g., case-insensitive matching, ignoring accents), enabling users to find relevant content across languages and variations in spelling or diacritics.

By grasping the concepts of alphabets and accents, software developers can build robust, inclusive applications that handle multilingual text effectively. Understanding character encodings, implementing proper text validation, considering sorting and collation rules, and enabling efficient search capabilities are crucial steps toward supporting diverse linguistic communities and providing a seamless user experience across different languages.

Now, let's delve deeper into specific alphabets and accents commonly encountered in software development, exploring their unique characteristics and considerations for handling text from different nations.

Appendix A

Answers to Exercises



Index