Web Server Design

**Bauman, Daniel**

**Abstract**

Network connectivity is one of the principal components in the making of complex, useful, mobile apps. Whether it serves to provide information, such as up-to-date weather for a weather app, or to connect people such as a messaging app, network connectivity is the key to solving many of the problems that mobile apps proport to solve. In order to provide online services for apps, the developer needs to make a web server that services the users. In this paper, we examine the basic structure of a web server, the challenges posed by this task, and important principles to follow in web design.

**Introduction**

Network connectivity is an essential part of various mobile applications. It primarily serves to access dynamic information which could not be supplied during installation, to establish communication between users, and to backup information from mobile devices. In order to connect an application to the web, a developer requires a web service, provided by a web server.

The interaction between a web server and an application involves communication via a protocol, typically TCP or UDP. The connection between the application and the server follows a request-response model, typically using the HTTP protocol which involves a small number of request types with an infinitely flexible set of parameters. The web server than handles each request, usually asynchronously and in parallel with handling other requests, and responds appropriately with the information requested or with a detailed error code. The asynchrony of web servers is crucial to handling a potentially heavy volume of requests of varying complexity with relatively low waiting time. Since applications need to be scalable to support potentially millions of requests per minute, the web server would typically involve multiple machines running in parallel and servicing clients. The complexity of this problem in turn requires unique and clever solutions to make readable and reliable code. In this paper, we’ll go over principles such as encapsulation and parsimony, that will help guide a developer in making scalable web servers that can service mobile applications.

**The Basics of Web Servers**

**Network Protocols: TCP and UDP**

Transmission Control Protocol, or TCP, is used in applications in which accurate transfer of data is more important than speedy transfer of data. Typically, applications that require transferring small packets of information infrequently, such as browsing the web or uploading photos, would use TCP. This protocol verifies that data arrives in the order in which it was sent, and it utilizes a handshake in which the receiver of information acknowledges the information was received by sending a received notice back to the sender (Shafer, 2020). Because of this, TCP is reliable and generally preferable in situations in which speedy response time and large volume of data are not needed.

User Datagram Protocol, or UDP, is a protocol typically used in video or audio streaming and in online gaming applications, both requiring transmission of large amounts of data in a short span of time (Shafer, 2020). UDP verifies neither the correctness of data, nor the order in which it arrives; while this makes the quality of data less secure, it allows the data flow to arrive much faster so that applications requiring large amounts of data in a short amount of time could be supported with a reasonable cost of server hosting. Furthermore, UDP doesn’t perform a handshake between the server and the application to verify the connection is stable, and in avoiding this it allows for UDP to broadcast data across to multiple devices simultaneously, as it doesn’t need to wait for confirmation from each connection that it received the data.

**HTTP and The Request – Response Model**

Hypertext Transfer Protocol, or HTTP, is a request-response modeled network protocol and the most common form of communication for web services. The protocol usually involves a stateless web service, that is a server that does not keep track of which requests each connection has made, but rather treat each request independently of the connection on which it was made (Melnichuk, 2018). The requests are in the form of one of nine methods: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATH. Each method call can have a list of request parameters without a limit on number or type of parameter, making for great flexibility of server design for developers. The GET method retrieves data from the web server, without causing any significant other effect on the server (Melnichuk, 2018). It can be used, for example, to retrieve a recipe in a food application, or the user profile of a user on a social network. The HEAD method retrieves only the beginning of a response to another request, which contains information about the success of the request, formatted in the response codes set by the HTTP response standards, and the data type of the response. The POST method sends data to be saved by the server in some database, updating the data served by the web service; it can be used to upload a post to a social media feed or send a message to be delivered to another user (Melnichuk, 2018). The PUT method implements a narrower version of the request made with the POST request, requesting a piece of data to replace an existing database entry or make a new database entry; it can be used to fill in table entries or upload a file to a shard drive. The DELETE method is used to delete data from a database; web servers typically require permission and ownership to allow for the execution of this request; it can be used to delete a post or a user profile in a social media app. The TRACE method returns the received request from the server back to the application, and it is often used to check for additions or modifications to the request that could be made by intermediate parties between the client and server. The method is useful in checking for phishing attacks and middle-man software used to steal user data, but it can also be used for debugging purposes. The OPTIONS method is used to retrieve the possible requests that could be made to a particular domain in the server. It can be used to gather information to be used in making other HTTP requests. The CONNECT method creates a secure tunnel between the web server and the requesting application, typically using an SSL protocol. This method can be used to establish an SSH connection allowing remote access to the server computers from an admin user. The PATCH method is used to apply changes to data in a database or a file. Its parameters usually include the resource to be modified, and the changes to be made to it. It can be used for making realtime modification a file on a shared drive. The response from a web service using HTTP includes a status code, with a three-digit number in one of the formats: 1xx, 2xx, 3xx, 4xx, 5xx. The codes correspond to preset categories of messages. 1xx codes refer to informational requests, such as asking the client to continue making a request or providing an intermediate response such as “processing information”. 2xx codes refer to a successful response to a request, such as when a POST method has successfully added the data sent, or that a GET request successfully retrieved data. The 200 code is the basic code signaling a successful response. The 3xx codes refer to a redirection by the web server, such as when requested data has been moved to another address. The 4xx codes refer to a user error, such as requesting a non-existing resource or making a request without proper permissions. The 5xx codes refer to a service error, such as a server crash or limited resources. HTTP responds include a header that includes the status code, and information about the data type being transferred (Singh, 2020). The response data is typically in the type of text, HTML, or a JSON file, structured as a dictionary object with keys and values of various types.

**Web Design Challenges**

**Scalability**

Unlike most applications, web servers have an incredibly wide range of possible workload. For example, a developer can know with reasonable assurance the amount of clicks a mobile app could need to process per minute, but a web developer can never know ahead of time how many users may need to access the service simultaneously in the future. Because of this, web servers need to not only be able to handle a large load of simultaneous requests, but an unknowingly large workload. The property needed to solve this problem is scalability, the ease with which the server could be made to support more users, and the limit of how many users could be at a time without sacrificing response time (Roe, 2002). Other useful sub-properties of this property are encapsulation and modularity.

**Complexity of Requests**

Because of the complexity involved in the operation of most web services, and the variation of the types of requests that a web service supports, a web server needs to be plan responses appropriately so that a simple request that should receive a response easily and quickly is not delayed by a complex request requiring long and complex operations on the server side. For example, if a web server receives first a request to upload a large file to a database, and then another request to retrieve simple information not requiring access to that database, the server should be able to respond to the second request before responding to the first. The properties needed to solve this problem are asynchrony and concurrency. Asynchrony involves operations working in a non-sequential way (Roe, 2002). This means if one request could be completed before another, it would, regardless of the order in which they arrived. Concurrency means operations can be run simultaneously without blocking each other; for example, if a server sends a request to retrieve data from a database for a response, it can respond to other requests while waiting for a response from the database, rather than being blocked while waiting.

**The Principles of Web Design**

**Asynchrony**

The asynchrony principle entails a system in which operations that can be done in parallel don’t block each other from being run. By allowing operations to run asynchronously, a server can maintain low response time, especially for small and simple requests. To explain why this is important, we consider an example of a retail store service made with an asynchronous server and a database. While the database may not allow multiple writing requests to be completed in parallel to prevent an override error, it could allow for multiple reading requests to be run in parallel. Because of this, if a retail store attempts to upload a new catalog of items, which may take a long period of time if the items contain a lot of data, the server would still be able to service simple item finding requests made by users while adding the catalog and prevent the inconvenience of a long waiting period that would’ve occurred with a synchronous server. The principle of asynchrony is crucial for a web server, as it deals with potentially millions of requests at a time and needs to respond to each request efficiently and quickly, without letting the complicated requests block the resources.

**Scalability**

The scalability of a web server is the ease with which it could be made to support more users. While this problem is difficult to solve, there are techniques that can help make servers more scalable. One important property for a scalable web server is encapsulation, the division of the server functions to independent components (Roe, 2002). For example, a common approach when constructing scalable web servers is splitting the server program from the database, not only so that they run in parallel, but in order to allow for multiple instances of both the server program and the database program to be run on multiple devices in a complex system. This property is crucial as web servers handling millions of requests are rarely run on one device and splitting as much functionality as possible to be run independently allows for ease of expansion of the server onto more hardware. Another name for this property is modularity, as the server is built as independent or semi-independent modules executing different functions.

**Conclusions**

In conclusion, a web service is a necessary component in a variety of mobile applications. In order to construct a successful web service, a developer needs to create a web server, typically a TCP or UDP based server, that handles requests by the mobile app. The requests are usually made with the HTTP protocol, using methods such as GET, POST, DELETE, or PUT. In order for the web server to be successful, it needs to be scalable, asynchronous, and concurrent.

Works Cited

Roe, Colleen et al (2020). *Server-Side Design Principles for Scalable Internet Systems.*

<http://eds.a.ebscohost.com.umasslowell.idm.oclc.org/eds/detail/detail?vid=25&sid=de3e5219-2d30-4014-91a0-9b6038de395f%40sessionmgr4007&bdata=JnNpdGU9ZWRzLWxpdmU%3d#AN=6538097&db=bth>

Melnichuk, M. et al (2018). *Web-service. RESTful architecture.*

<http://eds.a.ebscohost.com.umasslowell.idm.oclc.org/eds/detail/detail?vid=46&sid=de3e5219-2d30-4014-91a0-9b6038de395f%40sessionmgr4007&bdata=JnNpdGU9ZWRzLWxpdmU%3d#AN=130507428&db=edb>

Singh, Mrigendra (2020). *How do Web Servers work?*

<https://www.geeksforgeeks.org/web-servers-work>

Shafer, Amber (2020). *UDP vs. TCP and Which One to Use for Video Streaming*

<https://www.wowza.com/blog/udp-vs-tcp>

Chi-Jen, Wu et al (2013). *A Scalable Server Architecture for Mobile Presence Services in Social Network Applications*

<http://eds.a.ebscohost.com.umasslowell.idm.oclc.org/eds/detail/detail?vid=1&sid=a991ce8d-e140-4fb2-a4f6-0a3213268bf3%40sdc-v-sessmgr03&bdata=JnNpdGU9ZWRzLWxpdmU%3d#AN=edseee.6104055&db=edseee>