Web Architecture and Frameworks

This lecture is an introduction to the technologies enabling mobile web apps and making programming easier for them. The lecture attempts to introduce important terms and give an overview of the structure of the technologies. It is not intended to be an exhaustive list of all the potential technologies you could use to create a web app or a comprehensive description of the technologies. I hope it is a good start for you and gives us all a common language to speak of the technologies.

- ## The Internet: Client-Server Model and HTTP

The internet is a network of machines that interact, assuming a client-server model. That model assumes that the client sends “requests” to the server, and the server sends “responses” back to the client. For our web apps, the client is the browser and the server is the Apache Tomcat running on the server machine. The machines in the network need a language to interact with (or speak with each other) and there are many languages or “protocols.” In fact, not only are there many different protocols, but there are layers of protocols, called the “internet protocol suite,” going from the link, to internet, transport and finally application layer. We are only concerned with the topmost protocol layer, the “application layer.” HTTP (meaning Hypertext Transfer Protocol) is one of the application protocols, the other protocol that machines can use to speak with each other are POP, SSH, Telnet, FTP, etc.

HTTP is the most pervasive protocol and is used for serving webpages. Because it is an application protocol, the request and response use ASCII text. A request consists of three parts: the “request line,” the “header,” and an optional “body.” The request line expresses the “request method,” the “request URL,” and the HTTP version. The header contains information, and since HTTP 1.1 must contain the host domain name. The response contains the “response line,” the “header,” and almost always the “body.” The response line contains the HTTP version, “status code,” and “status message.” The response header contains information, for example, the “content-type” and “connection.” The response body is typically the content that was requested, for example, the web page expressed as HTML.

- **Status Codes**

You should become familiar with the common status codes and messages that the server sends.

200s are success

- 200 OK

400s are client errors

- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 408 Request Timeout

500s are server errors

- 500 internal Sever Error
- 504 Gateway Timeout
- Methods

You should become familiar with some of the common methods.

- GET – should only retrieve information from the server. The method for static webpages, but can also be used for dynamic webpages that only read from the database. A GET request does not have a body.
- POST – request that the body be added to the resource, typically a database table. The map (meaning the data) from a web form is in the body of a post request. Typically, a new database entry would be a POST request.
- PUT – request that the body of the request replace the resource, typically a database. Typically, PUT requests are updates into the database.
- DELETE – request to remove an entry from the resource. A remove from the database is typically a DELETE request.

The use of proper request methods is only a guideline and only enforced by your server coding standards. Note that the Grails auto generated code adheres to the standards. GET is considered a safe request because it does not change anything on the server, while POST, PUT, and DELETE change data on the server. Also, PUT and DELETE are idempotent because a single call has the same effect as multiple calls.

- Representational State Transfer (REST)

REST is a web architecture for web services. It is only a standard for the design of the web services and can only be enforced by server coding practices. Some of the important properties of REST:

- Client-server – separation of concerns between the client and server. For example the client should not be concerned with data storage or the database that is provided by the server.
- Stateless – client-server communication does not depend on the state of the server or client. No client information is stored on the server. Session state is held by the client.
- Uniform Interface – contains "identification of resources" and "self-descriptive messages" besides others.

Specifically, to web services, the REST standards implies using a base URI, JSON or XML for data transfer, proper use for request methods (GET, PUT, POST, and DELETE), and hypertext links for representing the state of the client or requested resource. The advantages of using REST architecture are that it scales very well with the number of client requesting services and makes the client request codes very visible. As much as possible website should adhere to the REST architecture.

But it is not always possible to adhere to the REST architectural standards. For example, the typical practice is not to store all the session data on the client. Generally, only the session ID is stored on the client. For security, we would not want all the session data stored on the client and then transmitted for every request. For example, should the authentication happen on every webpage

Request. Also, if the web app is to work offline, then the client must be concerned with data storage.

- HTTP Cookies

Cookies enable the server to store a small amount of data on the client. Actually, the amount of storage space is not limited by most browsers, but it is considered bad technique to store more than a few hundred bytes in a cookie. Cookies are set in the header of the server response and stored by the browser and associated with the domain of the URL. On the next request made by the browser, the browser checks if there are any cookies associated with the domain of the URL in the request. If there are any cookies, the browser attaches them to the header of the request. Cookies contain a map of names and values.

The typical use of cookies is to store state information on the client, these cookies are called session cookies. For example, when a user logs in, the server creates a session ID and a file containing the session parameters on the server for the session. A file is used to store the session values because the session parameter are not long-lasting, but also a database can be used to store session values. The session ID is sent to the browser in a cookie. On the next request, the cookie is sent back to the server. The server can then find the file containing the session value for this request from the session ID.