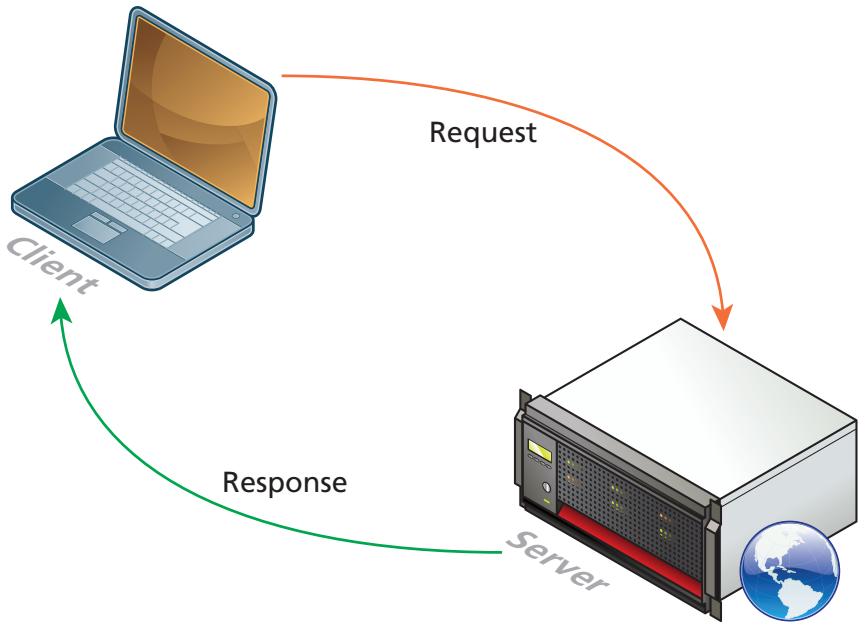
Topic 7: Intro to Spring

Part a: Web basics

Clients and Servers

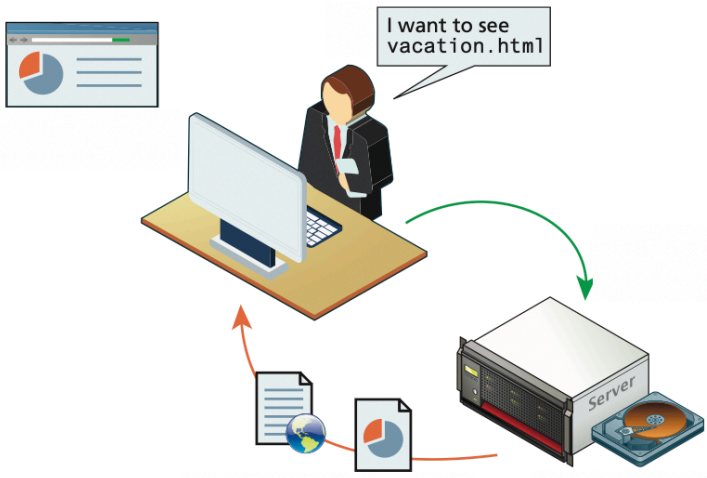
The Client-Server Model

The Request-Response Loop



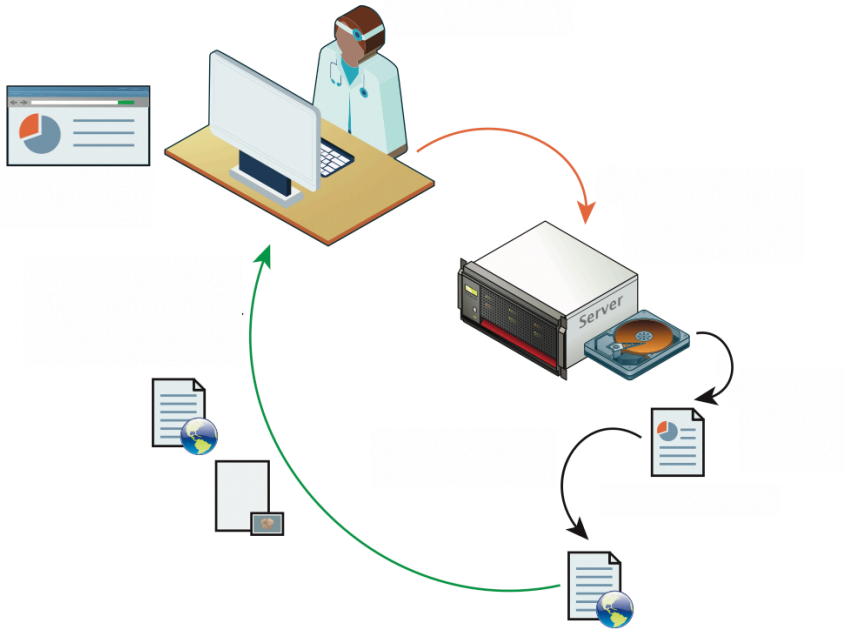
Definitions and History

Static Websites versus Dynamic Websites



Definitions and History

Static Websites versus Dynamic Websites



Internet Protocols

Internet Protocols

A Layered Architecture

TCP/IP.

These protocols have been implemented in every operating system, and make fast web development possible.

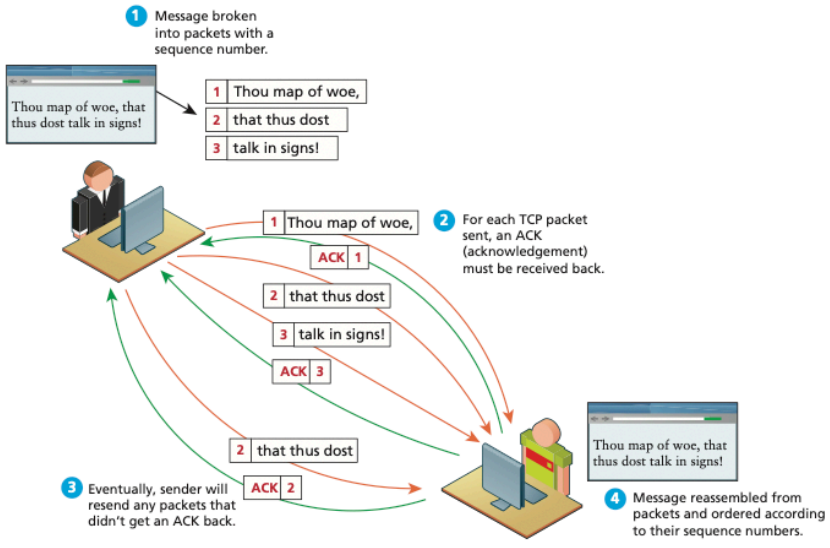
Networking is it's own entire discipline.

Web developer needs general awareness of what the suite of Internet protocols does

Internet Protocols

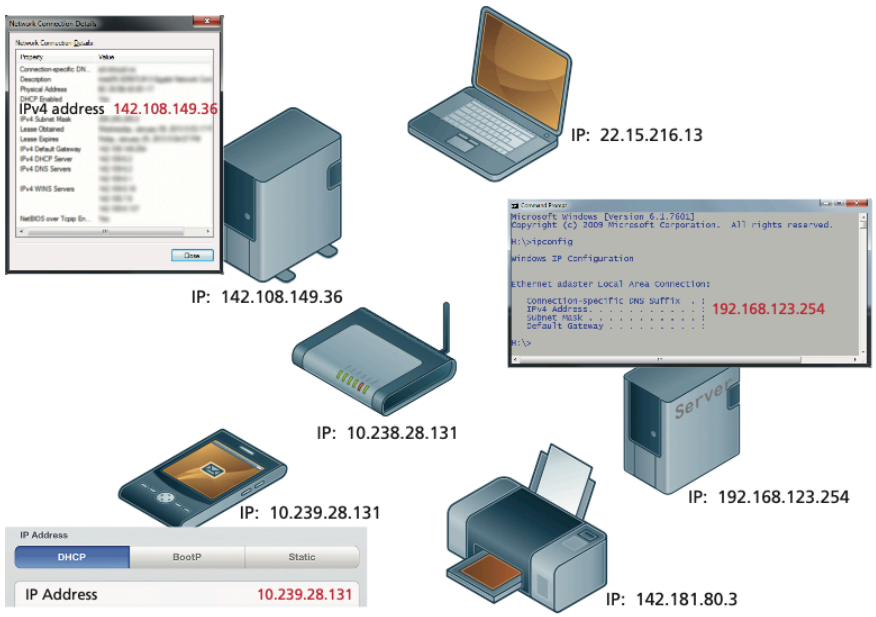
Transport Layer (TCP)

- Ensures transmissions arrive in order and without error



Internet Protocols

Internet Layer (IP)



Internet Protocols

Application Layer

There are **many** application layer protocols. Web developers should be aware of :

- **HTTP.** The Hypertext Transfer Protocol is used for web communication.
- **SSH.** The Secure Shell Protocol allows remote command-line connections to servers.
- **FTP**. The File Transfer Protocol is used for transferring files between computers.
- **POP/IMAP/SMTP**. Email-related protocols for transferring and storing email.
- **DNS**. The Domain Name System protocol used for resolving domain names to IP addresses.

Hypertext Transfer Protocol

Uniform Resource Locators

- The **domain** identifies the server from which we are requesting resources.
  - Alternatively, an IP address can be used for the domain

- The optional port attribute allows us to specify connections to ports other than the defaults
  - Add a colon after the domain, then specify an integer port number.

Uniform Resource Locators

Protocol

Recall that we listed several application layer protocols on the TCP/IP stack. FTP, SSH, HTTP, POP, IMAP, DNS, …

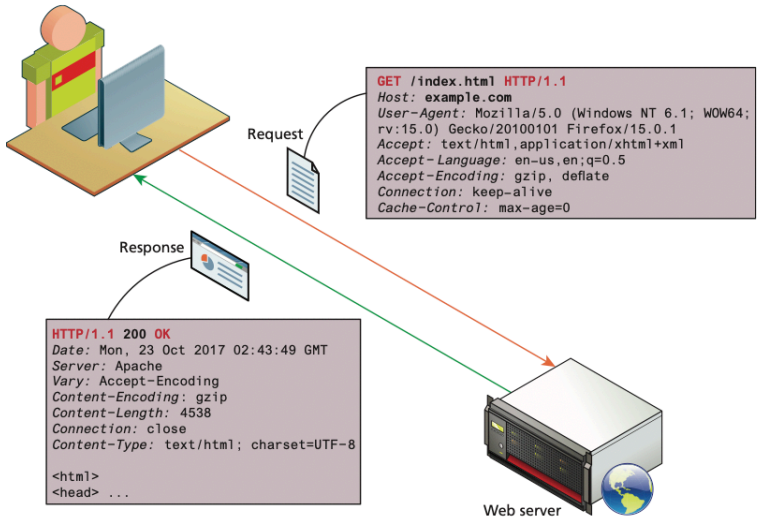
Requesting

- **ftp**://example.com/abc.txt → sends out an FTP request on port 21, while
- **http**://example.com/abc.txt → transmits an HTTP request on port 80.

Hypertext Transfer Protocol

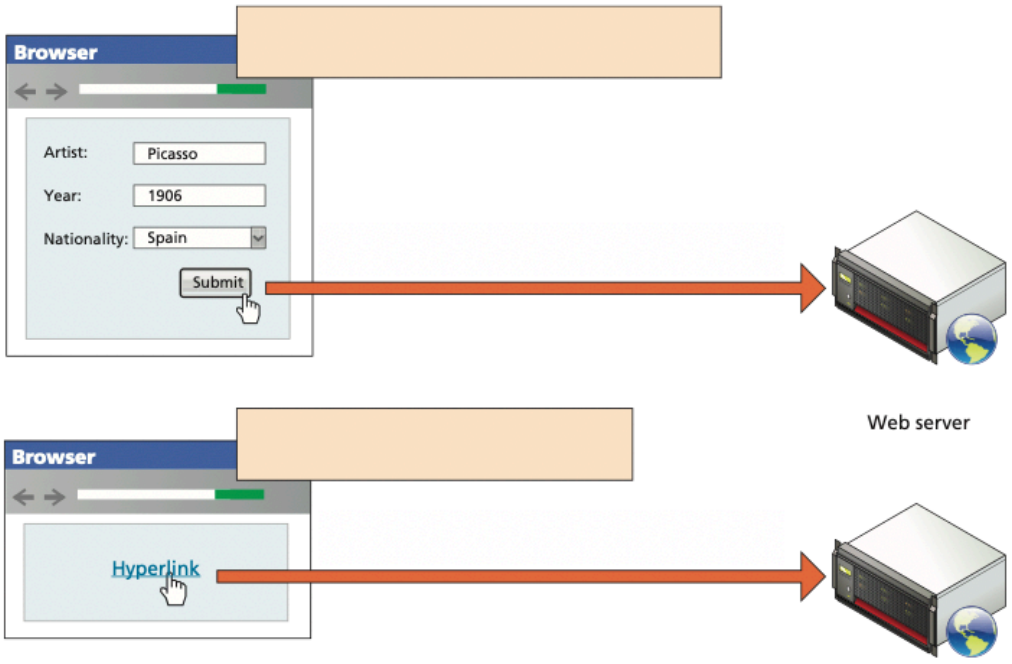
Headers

- **Request headers** include data about the client machine.
- **Response headers** have information about the server answering the request and the data being sent



Hypertext Transfer Protocol

Request Methods



Hypertext Transfer Protocol

Response Codes

- 2## codes are for successful responses,
- 3## are for redirection-related responses,
- 4## codes are **client** errors,
- 5## codes are **server** errors.

200: OK
301: Moved Permanently
304: Not Modified
307: Temporary redirect
400: Bad Request
401: Unauthorized
404: Not found
414: Request URI too long
500: Internal server error

REST and CRUD

What is REST?

- REST – **RE**presentational **S**tate **T**ransfer
- REST is a design pattern.
- It is a certain approach to creating Web Services.
- REST is a term coined by Roy Fielding to describe an architecture style of networked systems. REST is an acronym standing for Representational State Transfer.

Example: Course Registration Service

- Suppose that we want to write a course registration system for students and instructors.
- Instructors should get immediate service, and students get regular service.
- There are two main approaches to implementing the system ...

Characteristics of a REST based network

- Client-Server: a pull-based interaction style(Client request data from servers as and when needed).
- Stateless: each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.
- Cache: to improve network efficiency, responses must be capable of being labeled as cacheable or non-cacheable.
- Uniform interface: all resources are accessed with a generic interface (e.g., HTTP: GET, POST, PUT, DELETE).
- Named resources - the system is comprised of resources which are named using a URL.
- Interconnected resource representations - the representations of the resources are interconnected using URLs, thereby enabling a client to progress from one state to another.

Principles of REST web service design

1. Identify all the conceptual entities that we wish to expose as services. (Examples we saw include resources such as : parts list, detailed part data, purchase order)
2. Create a URL to each resource.
3. Categorize our resources according to whether clients can just receive a representation of the resource (using an HTTP GET), or whether clients can modify (add to) the resource using HTTP POST, PUT, and/or DELETE).
4. All resources accessible via HTTP GET should be side-effect free. That is, the resource should just return a representation of the resource. Invoking the resource should not result in modifying the resource.
5. Specify the format of response data using a schema. For those services that require a POST or PUT to it, also provide a schema to specify the format of the response.

"Approach 2" Advantages

- The different URLs are discoverable by search engines.
- It's easy to understand what each service does simply by examining the URL
- There is no need to introduce rules. Priorities are elevated to the level of a URL.
- It's easy to implement high priority
- There is no bottleneck. There is no central point of failure.

CRUD

Routes describe a resource (type of media) and what actions can be taken

- Actions:
  - GET
  - POST
  - DELETE
- CRUD
  - **C**reate
  - **R**ead
  - **U**pdate
  - **D**elete

Request and Response

Request

```
POST /user
Accept: application/json
Content-Type: application/json

{
    "name": "Bob", "country": "Canada"
}
```

Response

```
201 Created
Content-Type: application/json

{
    "user": {
        "id": 3, "name": "Bob", "country": "Canada"
    }
}
```