## I CASE STUDY

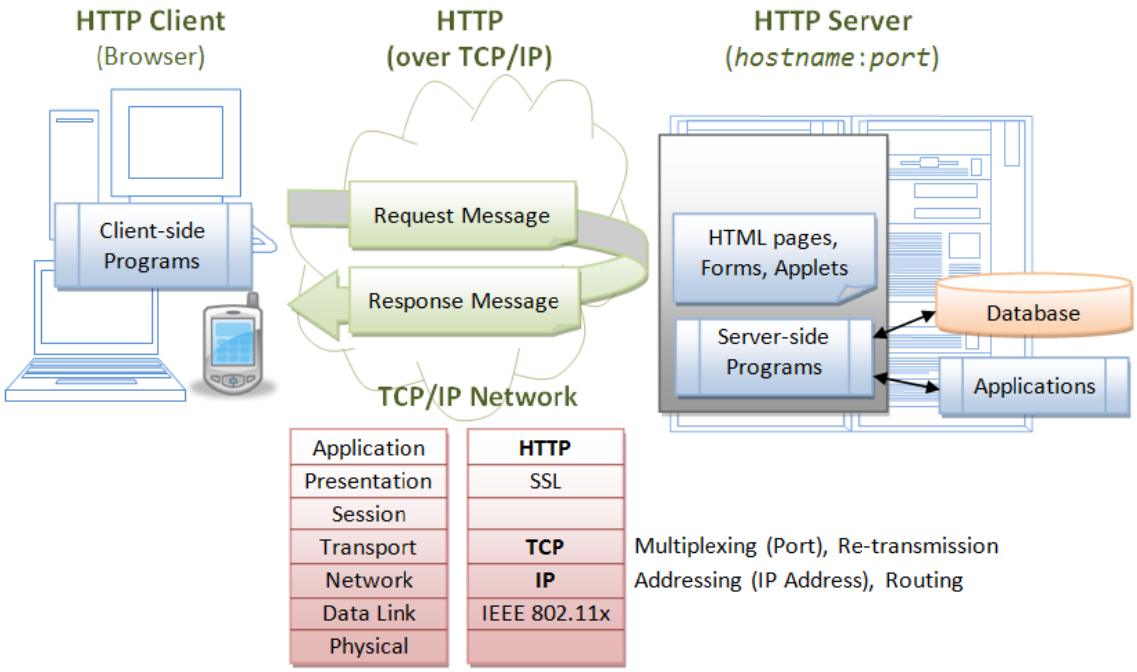
Care Well Hospital is a leading super specialty hospital in the city. They have various departments such as ENT, Cardiology etc. In every department doctors are available for OP consultation. Patients need to take appointment for consulting any doctor. First time a patient visits the hospital patient details will be recorded and an OP ID will be provided. For taking an appointment patient needs to provide OP ID to the clerk. Write a program to
a.Register new patient details
b.Book an appointment for a specific doctor
c.Book an appointment for any available doctor in a department

http://techforum4u.com/entry.php/1508-Java-Case-Study-Hospital-Management-System

## II Case study A Java Servlet E-Shop Case Study

1. Introduction

In this case study, we shall develop an "e-shop" based on the Java Servlet Technology. This e-shop is a typical Internet *business-to-consumer* (B2C) *3-tier client/server database application*, as illustrated below.




A typical *3-tier client/server web database application* consists of 5 components:

A HTTP Server (or commonly known as Web Server), such as Apache HTTP Server, Apache Tomcat Server, Microsoft Internet Information Server (IIS), nginx, or Google Web Server (GWS).

A HTTP Client, typically a Web Browser, such as FireFox, Chrome, IE or Safari.

A Relational Database, such as MySQL, Oracle, IBM DB2, MS SQL Server, MS Access, SAP SyBase.

Client-side programs, running inside the browser, which send requests to the server and process server's response. Client-side programs can be written in many technologies, e.g., HTML form, JavaScript, VBScript, Java Applet, Flash, ActiveX Control, and others.

Server-side programs, running inside the HTTP server, which process clients' request. The server-side programs extract the query parameters submitted by the client-side programs and query the database. Server-side programs can also be written in many ways, e.g., CGI Peal, Java Servlet/JSP/JSF, ASP, PHP, and many others.

The client and server interact with each other by exchanging messages using a protocol called HTTP (HyperText Transfer Protocol). HTTP is an asymmetric request-response protocol. A client sends a request message to the server. The server processes the request and returns a response message. In other words, in HTTP, the client pulls information from the server, instead of serverpushes information to the client. An HTTP server typically runs over TCP/IP, with a server IP address and on a TCP port number.

A typical sequence of operations for a webapp is as follows:

A client requests and downloads an HTML page containing an HTML form (or other client-side programs).

The client enters information into the form (such as search criteria), and submits these query parameters back to a server-side program for processing.

The server-side program extracts the query parameters, performs the database query, and returns the query results back to the requesting client.

The client displays the query results, and repeats the above steps for further request-response exchange.

Since this course is about Java, we shall build our webapp in Java. We shall write our server-side programs in Java servlets. We shall write our client-side programs in HTML forms and Java Applet.

http://www.ntu.edu.sg/home/ehchua/programming/java/JavaServletCaseStudy.html

## III Case Study : The Airport Announcer system

The Savanna Airport Company has decided to computerize its announcements, rather than using an operator. Announcements come from two sources: airlines and the company itself. Airlines send out announcements as required regarding check-in, flight delays and so on. The airport is responsible for security messages which tend to be sent at regular intervals, automatically. We are therefore looking at a system with an initial model design as in Figure 14.10. The monitor handles the distribution of the messages, via loudspeakers as well as on the TV screens around the airport. To keep our example simple, we shall use a single monitor, operating as a scrolling line of text.

Figure 14.10. Model for the Airport Announcer system.

## IV Case study using applets

**Cellphone applet.** Convert the cellphone GUI that you developed in Problem 10.3 into an applet. Add event handlers so that typing the keys produces a message on the Java console window.

**Changing prices**. The prices for fruit and vegetables change often. In the place where the Close button used to be on the Till program, add a button called Reset which will bring up a new window, listing the products and their current prices, and allow the user to type in new prices for any product that changes. The price change should be effective immediately after the window has been closed. Put password protection on the use of this window.

**Letter value update.** Create a separate applet with a password that can be selected by a menu or button from the Competition HTML page and will enable one of the newspaper employees to change the letter values interactively.

## V Case study Using connections and sockets

**Listing via HTML.** Create a Lister program as an applet, which accepts the name of a file coming in from a text field added to the applet's interface. Put the applet in an HTML page and have the HTML give a list of suggested files on a particular topic that can be fetched.

**Chatting clients.** The Chatter program (**Case Study 10**) provides a server only. Using the Ports program in**Example 14.3** as a basis, create a Java program that can be started up on the client side and provide similar facilities to telnet.

**Chatting on the web.** Even after Problem 14.2, the user interface to the Chatter is rather basic. Try making the server into an applet and embody it in an HTML page that provides instructions, displays the output, and has a separate line for typing in input.

**Knock knock!** Using the ATM server as a basis (**Example 14.4**) write a server to play Knock! Knock! The game goes like this:

| **Client** | **Server** |
|---|---|
| Knock knock | |
| | Who's there? |
| Amos | |
| | Amos who? |
| Amos Quito. | |

Use telnet for the client again.

**Cellphone messages.** Create a system to handle cellphone messages. There is a central server which keeps track of registered cellphones. Each cellphone is a client which collects a message and a phone number sends the message to the server, which relays it to the correct phone. This system is very similar to the chat system except that the messages go out to a specific person rather than to everyone. You can represent the cellphones using the command line, or you can use a GUI interface

**Voicemail.** If a phone is switched off, then a message should be stored at the server. As soon as the phone is switched on, a system message is sent out saying that there are so many stored messages. The user can then retrieve them one by one. Implement such a system for the cellphone

***For database connectivity***

**Pets database.** convert the Veterinary Tags to run on a database. The client applet should allow queries, and submission to a separate manager thread which handles updates. The manager can presumably then first checks that the updates are valid before adding them to the database.

**House search.** An estate agent keeps a record of available houses for sale on a database. When a client comes along, a list of suitable properties by price range and/or by area can be printed out. Set up such a system. Start by defining a house object.

**Coffee shop database.** Nelson would like to investigate whether a database would be better than serialized linked lists for having a truly permanent record of his inventory which can be queried by clients only, and updated by himself in his shop. Program a database for the coffee shop and write a report comparing the two approaches.

**Room bookings.** Going back to the Room Bookings system posed in the problems of investigates putting the system on a database. All queries and updates should be available to everyone.