JACOBS
UNIVERSITY

3-Tier Web Architectures

Ramakrishnan & Gehrke, Chapter 7

www.w3schools.com

www.webdesign.com

…

Overview

- Three-tier architectures
- Presentation tier
- Application tier

Components of Data-Intensive Systems

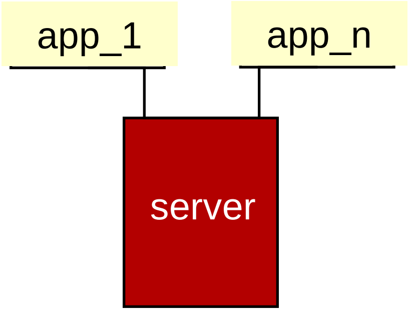
- Presentation
  - Primary interface to the user
  - Needs to adapt to different display devices (PC, PDA, cell phone, voice access, …)
- Application ("business") logic
  - Implements business logic (implements complex actions, maintains state between different steps of a workflow)
  - Accesses different data management systems
- Data management
  - One or more standard database management systems
- system architecture determines whether these three components reside on a single system ("tier) or are distributed across several tiers

Client-Server Architectures

- Work division: Thin client
  - Client implements only graphical user interface
  - Server implements business logic and data management
- Work division: Thick client
  - Client implements both graphical user interface and business logic
  - Server implements data management

Single-Tier Architectures

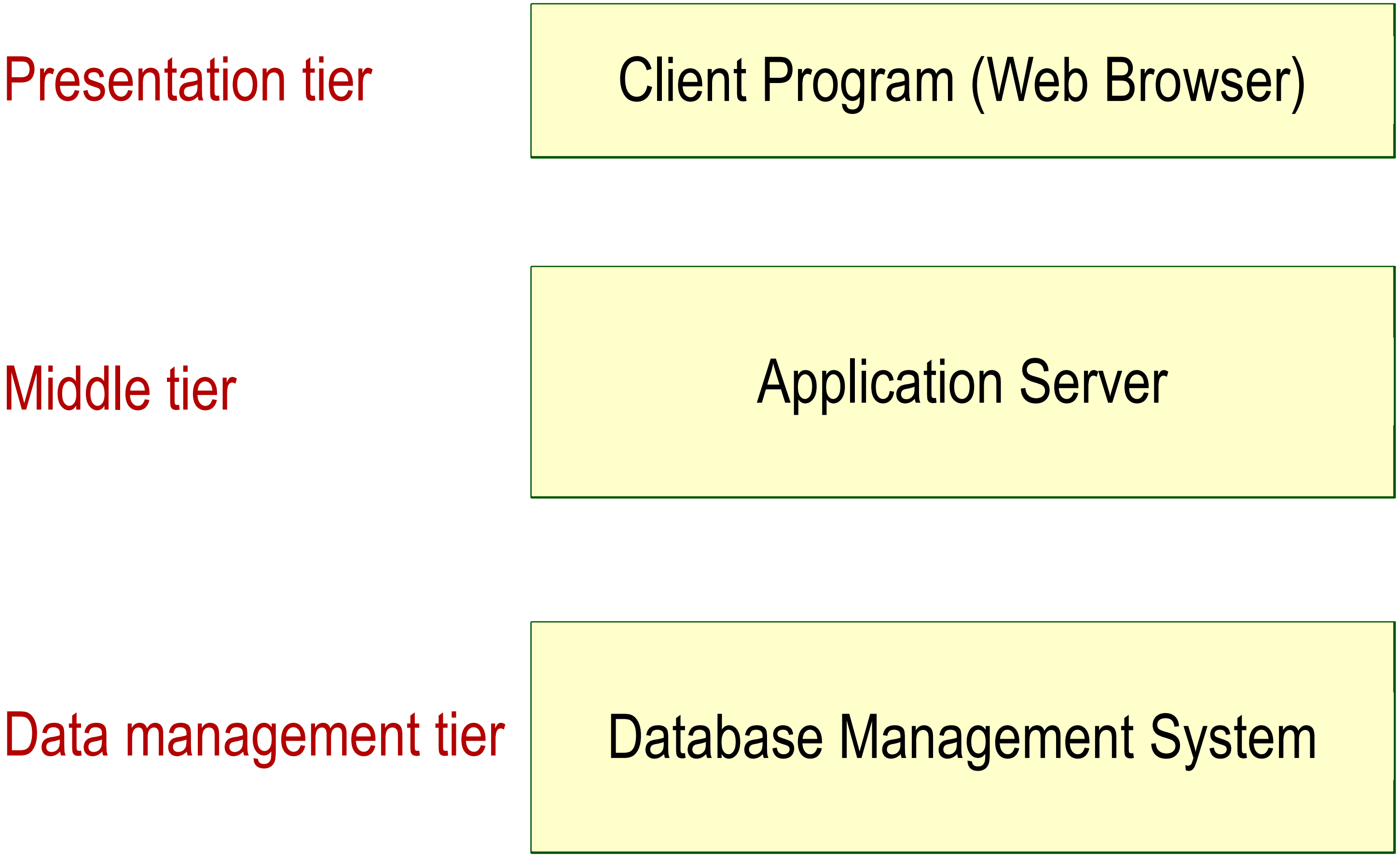
- All functionality combined into a single tier
  - usually on a mainframe
  - User access through dumb terminals
- Advantage
  - Easy maintenance and administration
- Disadvantages
  - users expect graphical user interfaces
  - Heavy load on central system



Disadvantages of Thick Clients

- No central place to update the business logic
- Security issues: Server needs to trust clients
  - Access control and authentication needs to be managed at the server
  - Clients need to leave server database in consistent state
  - One possibility: Encapsulate all database access into stored procedures
- Does not scale to more than several 100s of clients
  - high data transfer volume between server and client
  - More than one server creates a problem:
    x clients, y servers => x*y connections

The Three-Tier Architecture



Presentation tier

Client Program (Web Browser)

Middle tier

Application Server

Data management tier

Database Management System

Advantages of a 3-Tier Architecture

- Modularity
  - Tiers can be independently maintained, modified, replaced
- Scalability
  - Replication at middle tier permits scalability of business logic
- Thin clients
  - Only presentation layer at client (web browsers), no biz logic
- Integrated data access
  - Several database systems handled transparently at middle tier
  - Central management of connections
- Easier software development
  - Code for business logic is centralized, easier to maintain
  - well-defined APIs between tiers allow use of standard components → interoperability

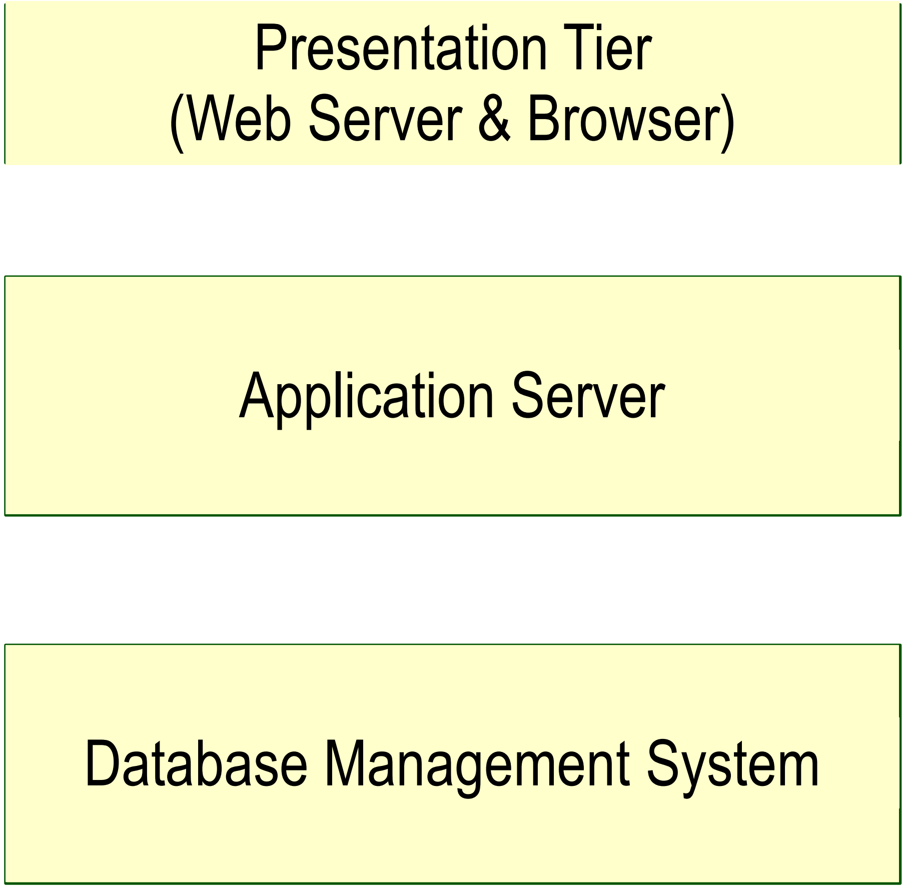
Overview of Technologies: Client-Side

- Contents presented by browser (static)
  - Text, HTML/CSS, XML/DTD/XSL, images, movies, audio, ...
- Contents interpreted by the browser
  - Dynamic HTML; Browser scripting: JavaScript, VBScript, ...
- Code executed by browser
  - in browser context: Java applets, ActiveX, …
  - Dedicated programs in browser context = plug-ins: flash, ...
  - External programs launched by browser = Helper applications
- *Security always an issue: keeping client (!) safe from intruders*

Overview of Technologies: **Server**-Side

- Static contents (eg, HTML) with executable code
  - SSI (Server-Side Includes), XSSI
  - Server-side Scripting (Livewire, ASP, PHP, JSP, ...)
- Generated contents
  - Separate process per call: CGI
  - Within server context: Fast-CGI, Servlets, ...
- Server extensions
  - Google APIs, NSAPI, IISAPI, Apache modules, ...
  - Database gateways/frontends
- Application servers
- *Security always an issue: keeping server (!) safe from intruders*

Technologies



Presentation Tier
(Web Server & Browser)

HTML, CSS, Javascript
Ajax
Cookies

Application Server

JSP, Servlets, CGI, …

Database Management System

Tables, XML, JSON, …
Stored Procedures

Lecture Overview

- Three-tier architectures
- Presentation tier
- Application tier

The Presentation Tier

- Recall: Functionality of the presentation tier
  - Primary interface to the user
  - Needs to adapt to different display devices (PC, PDA, cell phone, voice access?)
  - For efficiency, simple functionality (ex: input validity checking)
- Mechanisms:
  - HTML Forms
  - Dynamic HTML / JavaScript
  - CSS

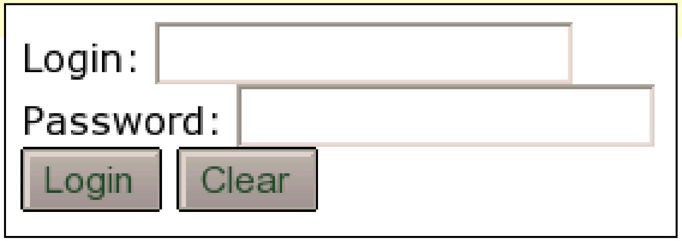
JavaScript

- Goal: Add functionality to the presentation tier
- Sample applications:
  - Detect browser type and load browser-specific page
  - Browser control: Open new windows, close existing windows (example: pop-ups)
  - Client-side interaction (conditional forms elements, validation, …)
- JavaScript optimal for Web browser because:
  - Built-in engine – always available, fast
  - Operates directly on “browser brain” = DOM

JavaScript: Example

- HTML Form:

```
<form method=”GET“ name=“LoginForm”
action="TableOfContents.jsp">
Login:
<input type="text" name="userid"/>
Password:
<input type="password“ name="password"/>
<input type="submit“ value="Login“
      name="submit” onClick=“testEmpty()”/>
<input type=“reset” value=“Clear”/>
</form>
```



- Associated JavaScript:

```
<script language="javascript">
function testEmpty()
{ result = false;
  loginForm = document.LoginForm;
  if ( (loginForm.userid.value == "") ||
       (loginForm.password.value == "") )
     alert( ‘Error: Empty userid or password.‘ );
  return result;
}
</script>
```

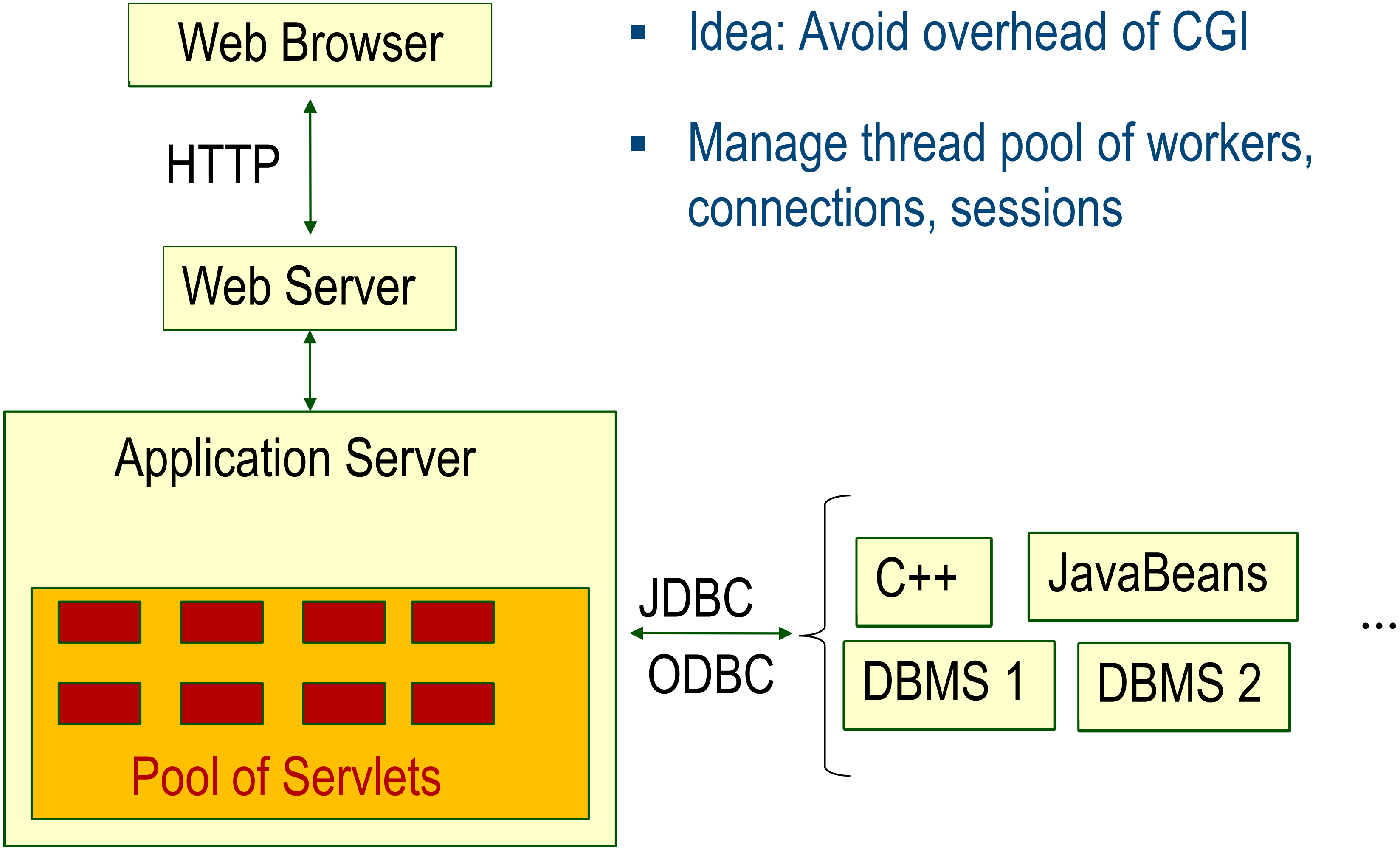
Lecture Overview

- Three-tier architectures
- Presentation tier
- Application tier

The Middle (Application) Tier

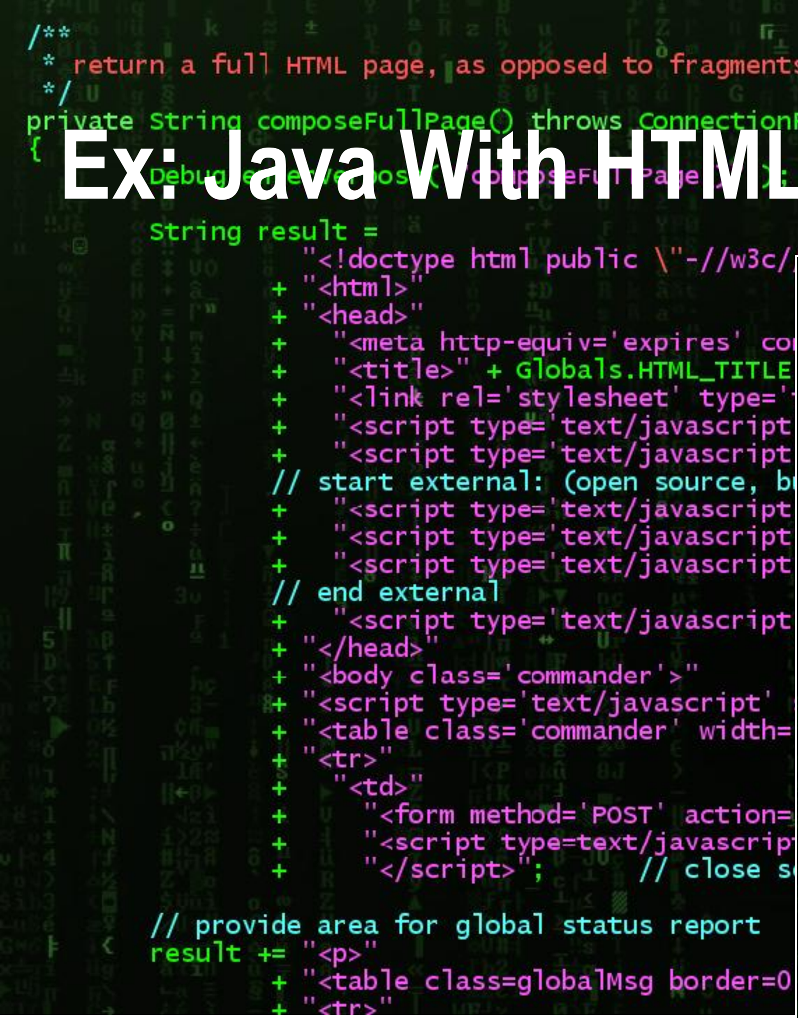
- Recall: Functionality of the middle tier
  - Encodes business logic
  - Connects to database system(s)
  - Accepts form input from the presentation tier
  - Generates output for the presentation tier
- Mechanisms:
  - CGI: Protocol for passing arguments to programs running at the middle tier
  - Application servers: Runtime environment at the middle tier
  - Servlets: Java programs at the middle tier
  - PHP: Program parts in schematic documents (see earlier)
  - How to maintain state at the middle tier

Application Server: Process Structure



- Idea: Avoid overhead of CGI
- Manage thread pool of workers, connections, sessions

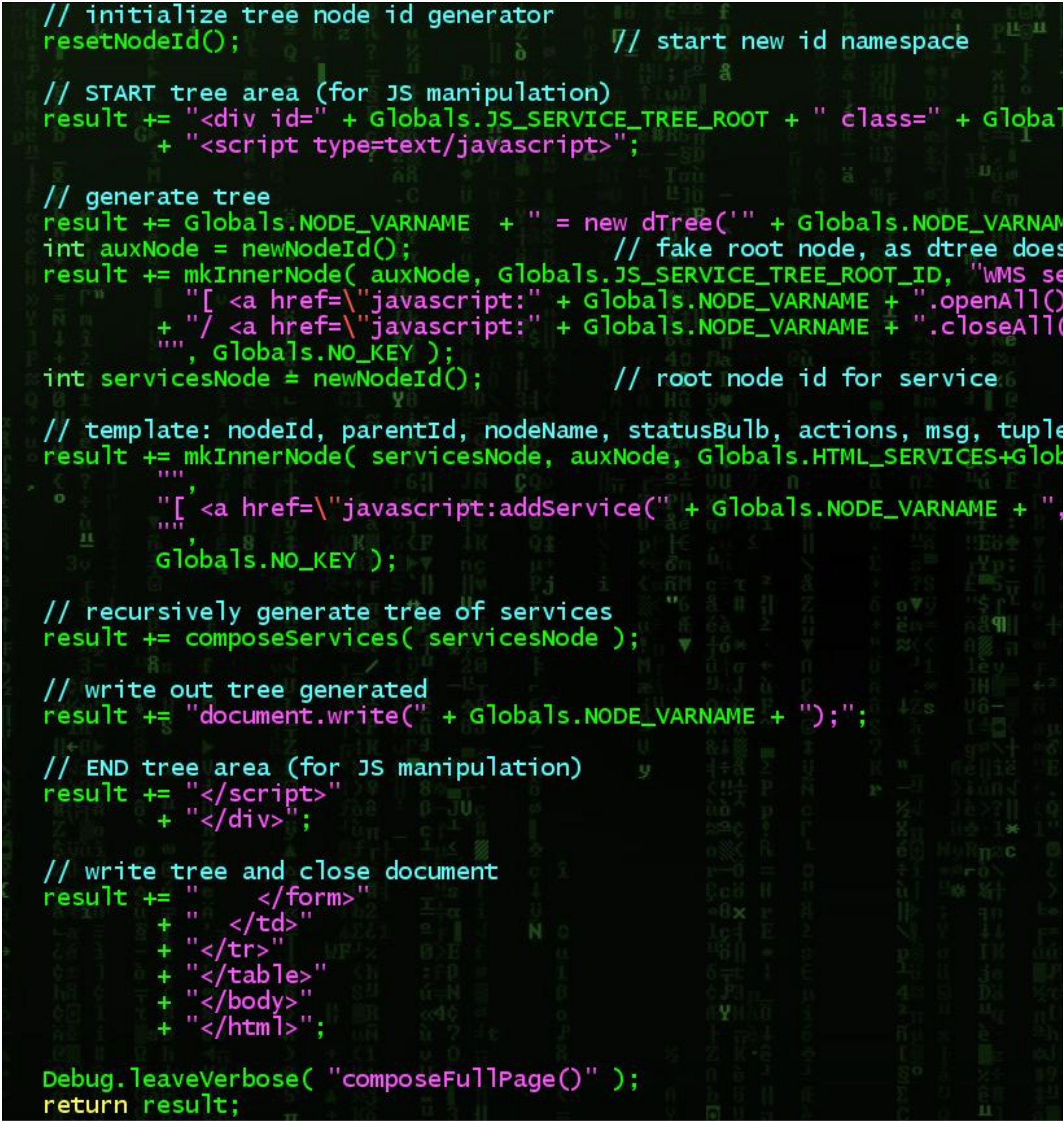
Ex: Java With HTML Inside



```
/**
 * return a full HTML page, as opposed to fragments
 */
private String composeFullPage() throws ConnectionFailedException, ConfigurationExcep
{
        Debug.enterVerbose( "composeFullPage()" );

        String result =
                  "<!doctype html public \"-//w3c/
                + "<html>"
                + "<head>"
                +   "<meta http-equiv='expires' co
                +   "<title>" + Globals.HTML_TITLE
                +   "<link rel='stylesheet' type='
                +   "<script type='text/javascript
                +   "<script type='text/javascript
                // start external: (open source, bu
                +   "<script type='text/javascript
                +   "<script type='text/javascript
                +   "<script type='text/javascript
                // end external
                +   "<script type='text/javascript
                + "</head>"
                + "<body class='commander'>"
                + "<script type='text/javascript'
                + "<table class='commander' width=
                + "<tr>"
                +   "<td>"
                +     "<form method='POST' action=
                +     "<script type=text/javascrip
                +     "</script>";        // close s

        // provide area for global status report
        result += "<p>"
                + "<table class=globalMsg border=0
                + "<tr>"
```



```
// initialize tree node id generator
resetNodeId();                              // start new id namespace

// START tree area (for JS manipulation)
result += "<div id=" + Globals.JS_SERVICE_TREE_ROOT + " class=" + Global
        + "<script type=text/javascript>";

// generate tree
result += Globals.NODE_VARNAME  + " = new dTree('" + Globals.NODE_VARNAM
int auxNode = newNodeId();                  // fake root node, as dtree does
result += mkInnerNode( auxNode, Globals.JS_SERVICE_TREE_ROOT_ID, "WMS se
          "[ <a href=\"javascript:" + Globals.NODE_VARNAME + ".openAll()
        + "/ <a href=\"javascript:" + Globals.NODE_VARNAME + ".closeAll(
        "", Globals.NO_KEY );
int servicesNode = newNodeId();             // root node id for service

// template: nodeId, parentId, nodeName, statusBulb, actions, msg, tuple
result += mkInnerNode( servicesNode, auxNode, Globals.HTML_SERVICES+Glob
        "",
        "[ <a href=\"javascript:addService(" + Globals.NODE_VARNAME + ",
        "",
        Globals.NO_KEY );

// recursively generate tree of services
result += composeServices( servicesNode );

// write out tree generated
result += "document.write(" + Globals.NODE_VARNAME + ");";

// END tree area (for JS manipulation)
result += "</script>"
        + "</div>";

// write tree and close document
result += "    </form>"
        + "  </td>"
        + "</tr>"
        + "</table>"
        + "</body>"
        + "</html>";

Debug.leaveVerbose( "composeFullPage()" );
return result;
```

Vice versa, ie: HTML with PHP inside?
See earlier example & your project!

Maintaining Client State

- http is stateless – but there is information that needs to persist
  - Old customer orders
  - “Click trails” of a user’s movement through a site
  - Permanent choices a user makes
- Advantages
  - Easy to use: don’t need anything
  - Great for static-information applications
  - Requires no extra memory space
- Disadvantage: No record of previous requests means:
  - No shopping baskets, no user logins
  - No custom or dynamic content
  - Security is more difficult to implement

Where to Keep Application State?

- Client-side state
  - Information is stored on the client’s computer in the form of a cookie
- Hidden state
  - Information is hidden within dynamically created web pages
- Server-side state
  - Information is stored in a database, or in the application layer’s local memory

Server-Side State

- Various types of server-side state, such as:
- 1. Store information in a database
  - Data will be safe in the database
  - BUT: requires a database access to query or update the information
- 2. Use application layer's local memory
  - Can map the user's IP address to some state
  - BUT: this information is volatile and takes up lots of server main memory

Client-side State: Cookies

- Cookie = (Name, Value) pair
- Text stored on client, passed to the application with every HTTP request
  - Lifetime can be preset (eg, 1 hour)
  - Can be disabled by client
  - wrongfully perceived as "dangerous", therefore will scare away potential site visitors if asked to enable cookies

- Advantages
  - Easy to use in Java Servlets / PHP
  - simple way to persist non-essential data on client even when browser has closed
- Disadvantages
  - Limit of 4 kilobytes
  - Users can (and often will) disable them
- Usage: store interactive state
  - current user's login information
  - current shopping basket
  - Any non-permanent choices user has made

Hidden State

- overcome cookie disabling
- Can “hide” data in two places:
  - Hidden fields within a form
  - path information
- Requires no client or server “storage” of information
  - state information passed inside of each web page – “on the wire”

Hidden State: Hidden Fields

- Declare hidden fields within a form:
  - <input type='hidden' name='user' value='username'/>
- Advantages
  - Users will not see information unless they view HTML source
- Disadvantages
  - If used prolifically, it's a performance killer
    – EVERY page must be contained within a form
  - Works only in presence of forms

Hidden State: KVP Information

- Information stored in URL GET request:
  - http://server.com/index.htm?user=jeffd
  - http://server.com/index.htm?user=jeffd&preference=pepsi
- Parsing field in Java:
  - javax.servlet.http.HttpUtils.parserQueryString()
- Advantages
  - Independent from forms
- Disadvantages
  - Limited to URL size (some kB)

Multiple state methods

- Typically all methods of state maintenance are used:
  - User logs in and this information is stored in a cookie
  - User issues a query which is stored in the URL information
  - User places an item in a shopping basket cookie
  - User purchases items and credit-card information is stored/retrieved from a database
  - User leaves a click-stream which is kept in a log on the web server (which can later be analyzed)

Some Web Service Security Hints

- Never use anything blindly that comes from client side
  - don't assume that JavaScript code has been executed
  - double check cookies on server
  - don't trust hidden fields contents
- never assume anything!
  - set defaults (define in a central place!)
- Clear state after request response
- as with any API: clean, defensive programming
  - perform standard plausi checks:
    admissible number ranges, empty strings, max string lengths!
- *Be paranoid !!!*

Summary: 3-Tier Architectures

- Web services commonly architected as having 3 components
  - Presentation / application / data management tier
- Application tier needs most implementation flexibility
  - Rich choice of platforms (Java servlets, PHP, ...), each with tool support
- To maintain state, use:
  - Hidden form fields, hidden paths, cookies, server store, …
- *For every aspect & component, security is an issue!*