Hospital Information Systems

Unit 1: Information Systems Architecture
Part 2 – Information Systems Architecture

Master in Biomedical Engineering

Rodrigo García Carmona

Universidad CEU San Pablo
Escuela Politécnica Superior
Departamento de Tecnologías de la Información

Table of Contents

1. Design patterns
2. Web applications
3. Model-View-Controller
4. HTTP
5. Frameworks and middleware

CEU

DESIGN PATTERNS

CEU

Information Systems Architecture

- The architecture of an information system is **the conceptual and logical design and disposition** of the components (both hardware and software) that comprise a system built using computers.
- It must take into account the **intended use** of the system.
- Software systems are not tied to the set of rules defined by the real world physics; it is necessary to define a new set of rules.
- Software systems cannot be “touched”, so a **set of models** is used instead.
- Information systems are incredibly **complex**.
- On top of that, they must adapt to **changes in requirements and technology**.
- However, the **replication and distribution costs** of software systems **are very low**. And hardware is starting to also get these advantages thanks to the cloud.

Information Systems Architecture

- Information systems architecture involves:
  - System organization
  - Hardware and software selection
  - Interfaces definition
  - Behavior of components and elements
  - Subsystems structure
- Information systems architecture also includes:
  - Features
  - Usability
  - Fault tolerance
  - Performance
  - Reusability
  - Technological and budgetary restrictions
  - Aesthetics

Abstraction

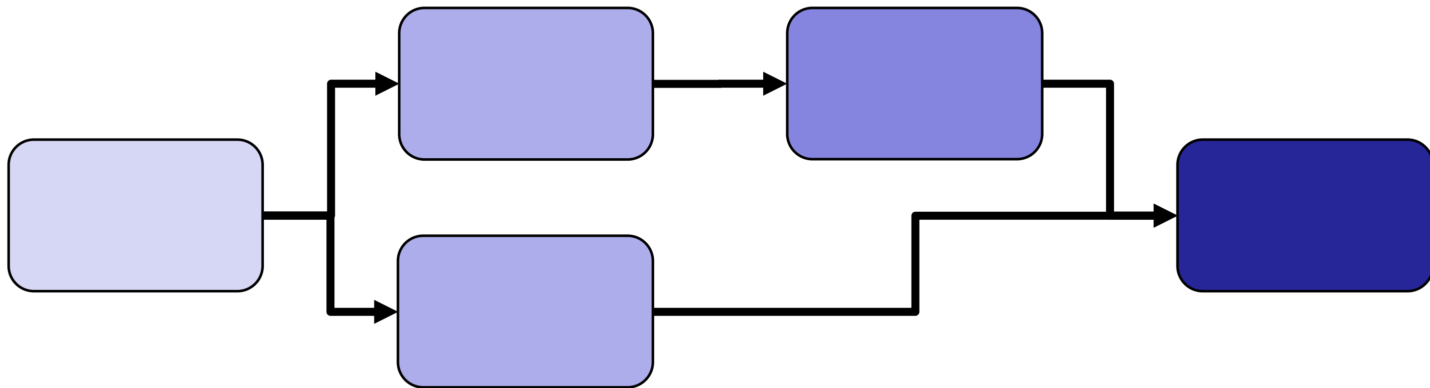
- IT engineers use abstraction to manage complexity.
- Abstraction involves representing the essential features a software design or component, without including the background details.
- Design patterns provide useful abstractions for building software systems.
- A pattern is a well-known solution to a common problem.
- All systems that are well structured use patterns.

Design Patterns

- A design pattern is a reusable solution to a design problem that involves a set of components that interact to solve a general design problem within a particular context.
  - How you should organize your components to solve a problem.
  - The specific circumstances under you apply the pattern.
  - Abstract templates that can be applied over and over again in many different contexts. They are **not** code.
  - Well known design patterns are often used, alone or in combination, to simplify a complex design.
  - Design patterns provide a way to communicate the parts of a design from one engineer to another.
- **Architectural patterns** are basic schemas that can be built upon. They define the structure of an information system:
  - Subsystems and their responsibilities.
  - Rules and techniques to build the relationships between them.

Design Pattern: Pipes and Filters

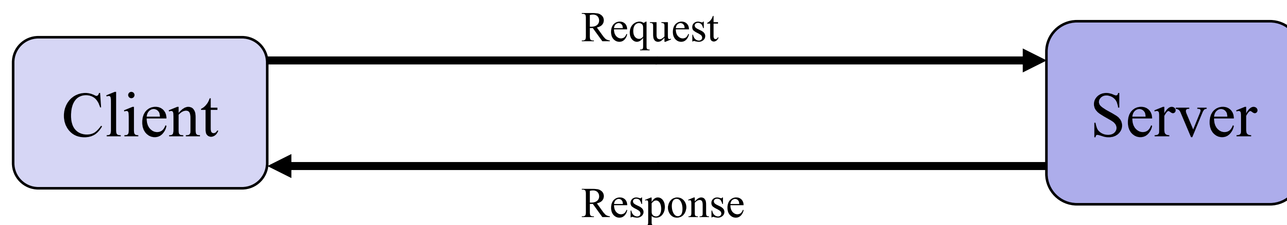
- For systems that process a stream of data.
- Each step is represented by a **filter**.
- Each **pipe** connect two filters.
- By combining filters we can create families of related systems.



CEU

Design Pattern: Server-Client

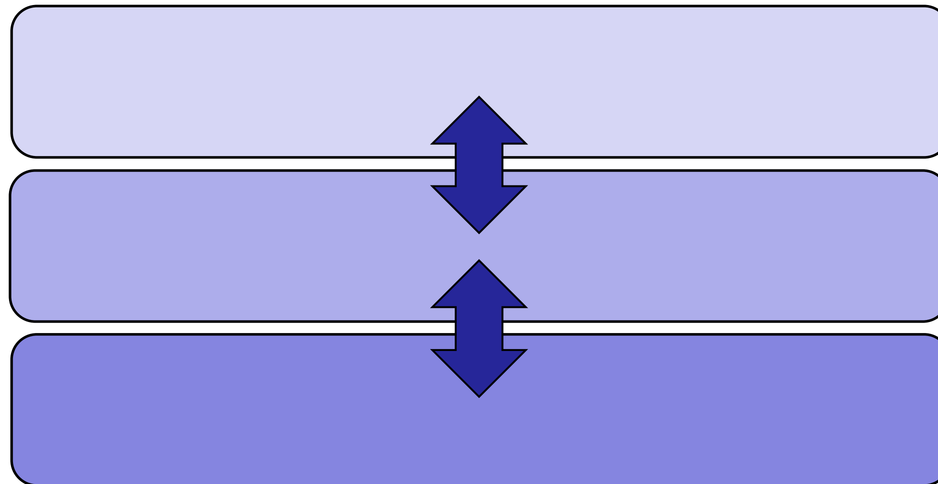
- Used to structure distributed systems. Allows us to distribute the components of an application between clients and servers.
- Decouples the components, which interact by invoking remote services.
- Processing is distributed between servers and clients.
- Client processes use resources provided by the servers.
- Client and servers could coexist in the same host or different machines connected using a network.



CEU

Design Pattern: Layered Architecture

- Components are organized in groups.
- Each group is in a different abstraction level.
- We only need to understand one level and the interface of another one at the same time.



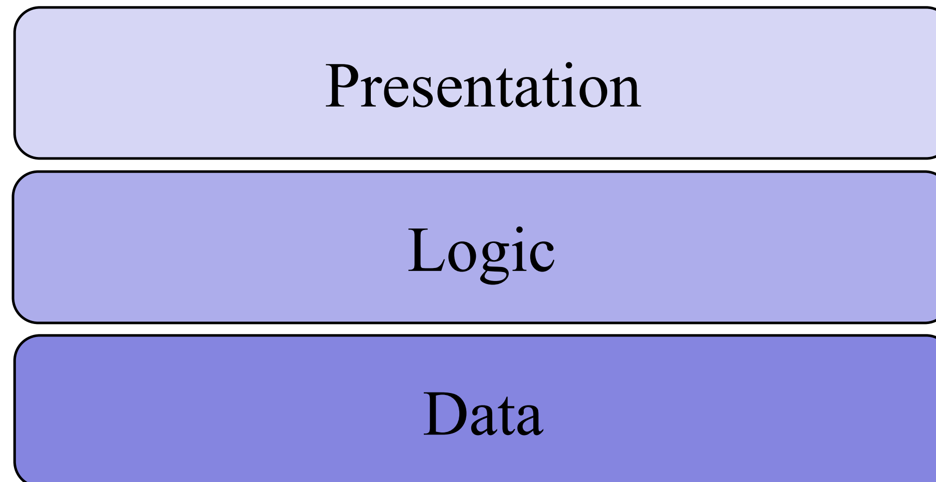
CEU

N-tier Architecture

- Combines the previous design patterns.
- A client-server architecture in which application functionality is further partitioned into separate tiers related to:
  - Presentation
  - Application processing
  - Data management
- Provides **separation of concerns**
  - Each tier addresses a separate “concern”, encapsulated within a well-defined interface.
  - This allows each tier to be developed, modified or replaced without affecting other tiers.
  - Encapsulation greatly simplifies development and maintenance of software systems.

3-tier Architecture

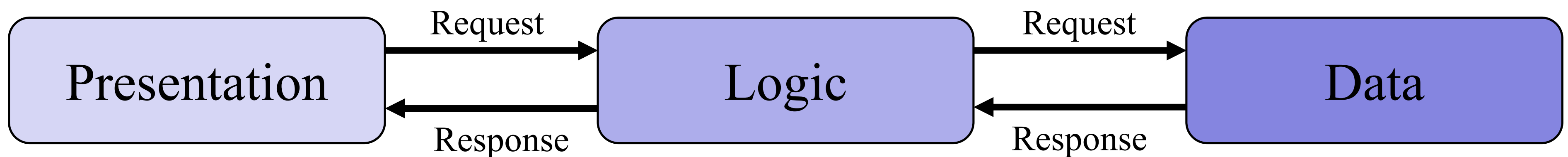
- Simplest approach, one tier per function
  - **Presentation tier:** user interface
  - **Application (logic) tier:** Retrieves, modifies and/or deletes data in the data tier, and sends the results to the presentation tier. Also responsible for processing the data itself.
  - **Data tier:** persistent storage of data associated with the application.



CEU

3-tier Architecture

- In the 3-tier architecture, each layer could be realized in a different physical (or virtual) host.
- The layers interact with each other using a client-server model.
- So we have client-servers inside a 3-tier architecture that, in turn, is used to implement a server in another client-server model.



CEU

WEB APPLICATIONS

What is a Web Application?

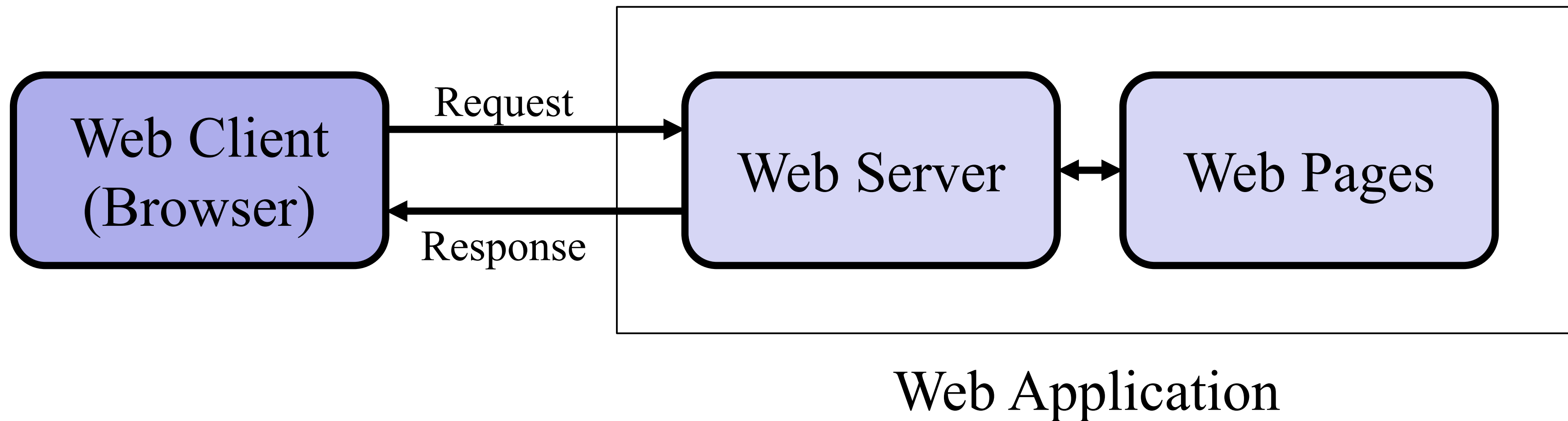
- Client-server architecture is the most basic model for describing the relationship between the cooperating programs in a networked software application.
- Traditionally, the client was a desktop application that connected to the server, another computer in the network, using a RPC (remote procedure call).
- Nowadays networked (and sometimes even purely local) applications are web applications.
- Web applications:
  - A web application is accessed using a browser as the client, and consists of a collection of client- and server-side scripts, HTML pages, and other resources that may be spread across multiple servers.
  - HTML pages contains hypermedia (text, images, video, …) and hyperlinks to other pages, giving the web applications its structure.

Evolution of Web Applications

- Web 1.0
  - Just web pages.
  - No separation between data and presentation. They must be separated to be able to create web apps that change its content dynamically.
  - Very simple browser.
  - Stateless protocol (each request is treated independently, no sessions)
- Web 1.0 Applications
  - The first web-based business models.
  - Richer applications needed more complicated server-side scripts that provoked maintenance issues.
  - Developers began creating applications that were more interactive, requiring a way of saving the state (cookies).
  - New technologies that improved performance: client-side scripts, cookies, faster web servers, web caching, CDNs…

Evolution of Web Applications

Web 1.0 Applications Structure



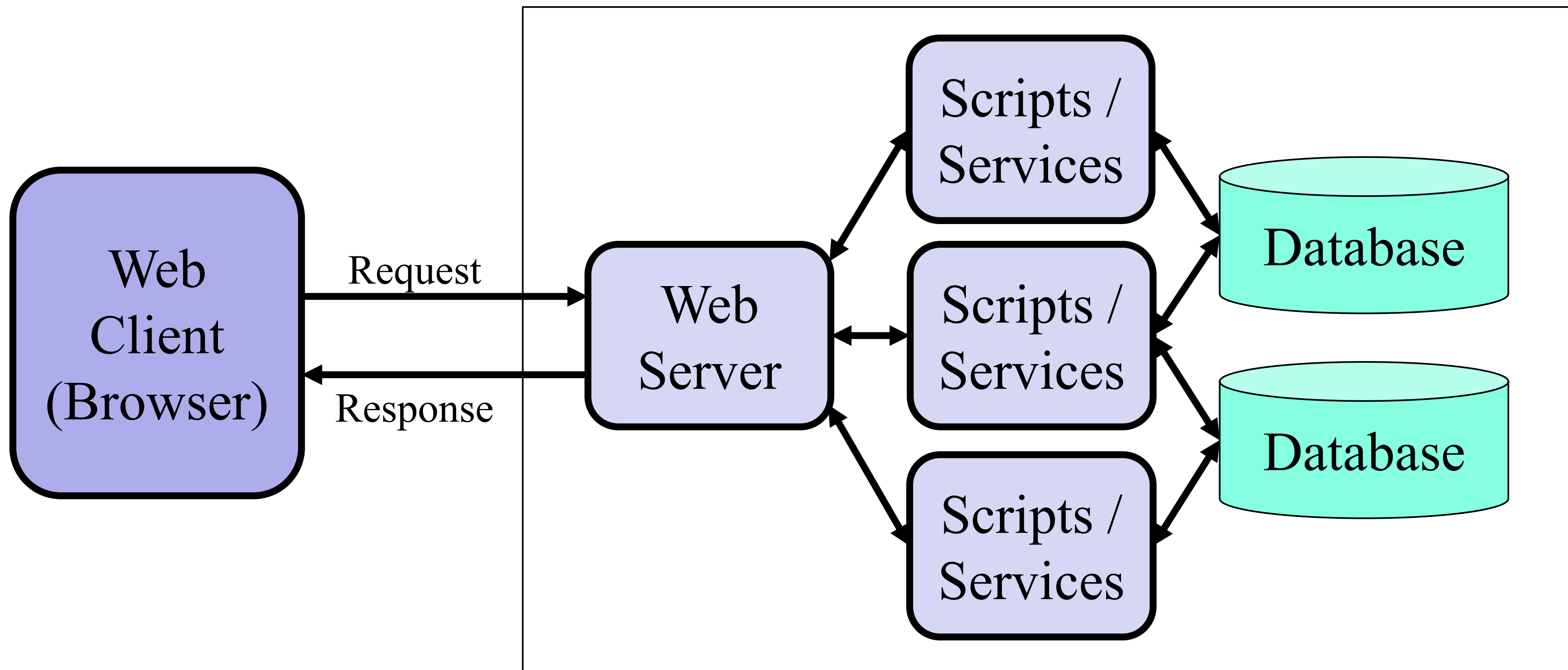
CEU

Evolution of Web Applications

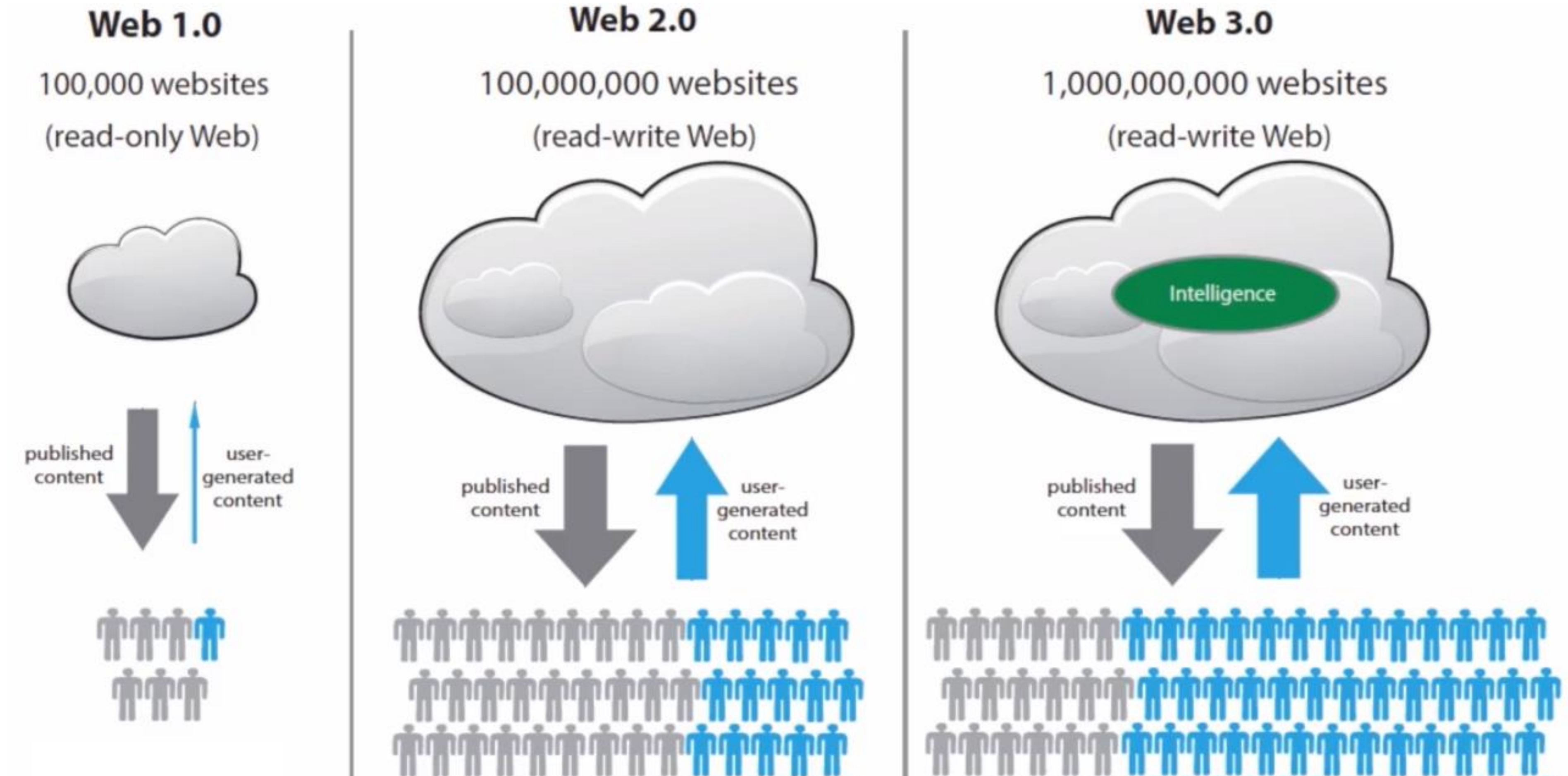
- Web 2.0 and 3.0 architectures
  - Web stack with better standards support (no more browser wars).
  - The browser is much more capable now.
  - With Web 3.0 the users are a third of the humanity.
  - Metadata and other semantic information.
- Web 2.0
  - Social networking, online commerce, wikis, lightweight collaboration.
  - Interactivity (Ajax).
- Web 3.0
  - Recommender systems, semantic web, mobile-friendly, Internet of Things (IoT).
  - Ubiquitous / Intelligent web.
- Web 2.0 and 3.0 enablers
  - JavaScript, XML, Jason, Ajax.
  - Web services interoperability (SOAP, REST).
  - Cloud computing.
  - Powerful mobile platforms and web-enabled devices.
  - Metadata, linked data, machine processing by intelligent agents.

Evolution of Web Applications

Web 2.0 and 3.0 Applications Structure



Evolution of Web Applications



CEU

Web Application Advantages and Disadvantages

- Web apps advantages:
  - Ubiquity and convenience of using a web browser as a client.
  - Inherent cross-platform compatibility.
  - Update and maintain web apps without distributing and installing software on potentially thousands of client computers.
  - Reduction in IT costs.
- Web apps disadvantages:
  - User experience compared to desktop apps used to be worse. Nowadays is not the case.
  - Privacy and security issues.
  - More difficult to develop and debug, there are a lot of interworking parts.

3-tier Architecture

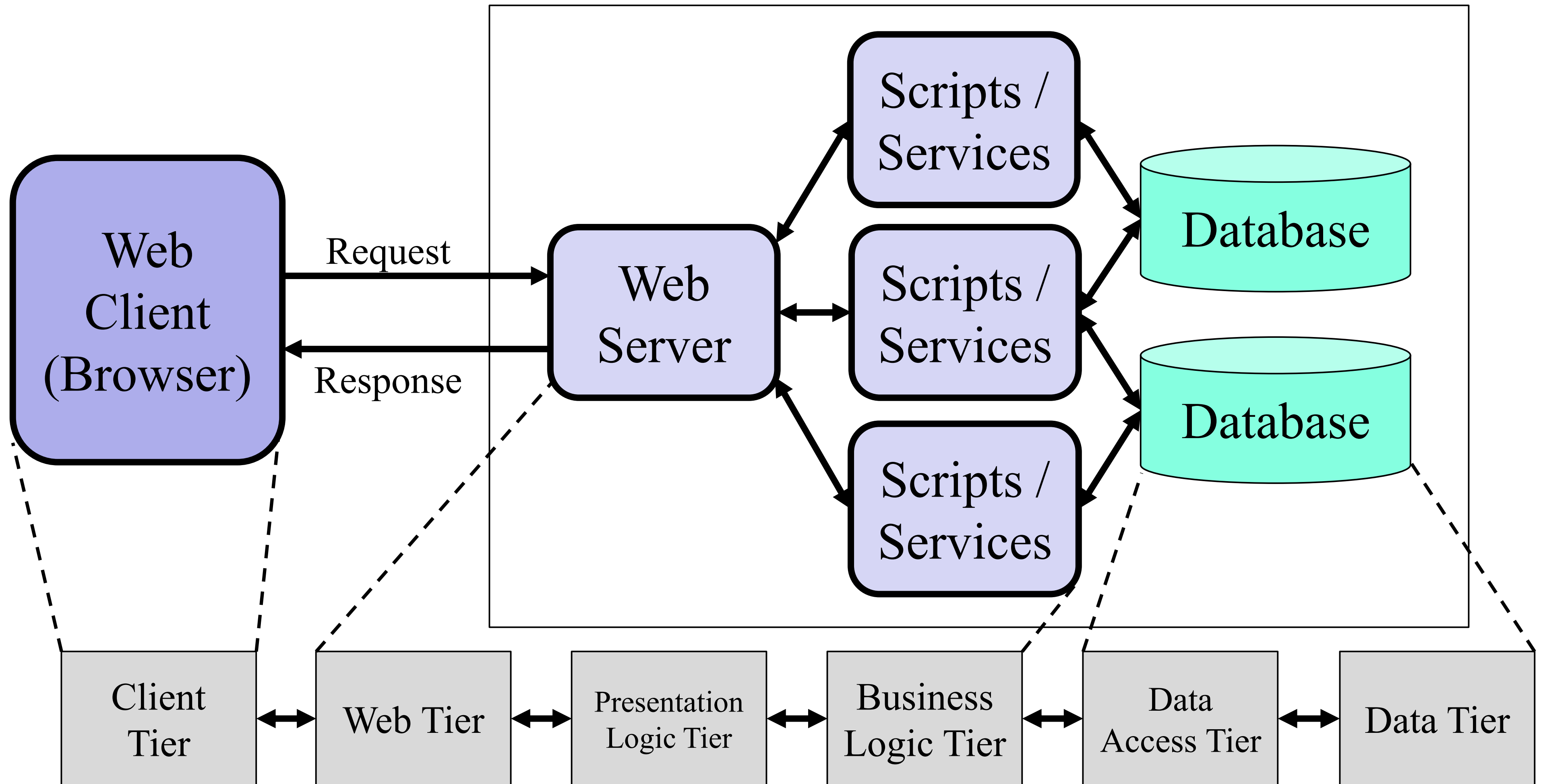
- Web applications work as 3-tier architecture:
  - **Presentation tier:** user's web browser.
  - **Application (logic) tier:** the web server and logic associated with generating dynamic web content.
  - **Data tier:** a database, relational or of other type.
- But we can further subdivide these three layers.

CEU

6-tier Architecture

- Presentation tier is subdivided in:
  - **Client tier:** client-side user interface components.
  - **Presentation logic tier:** server-side scripts for generating webpages.
- Application tier is subdivided in:
  - **Business logic tier:** models the business objects associated with the application.
  - **Web tier:** the web server.
- Data tier is subdivided in:
  - **Data tier:** the data used by the application, a persistent data store of some type.
  - **Data access tier:** responsible for accessing data from the data tier and passing it to the business logic tier.

6-tier Architecture



CEU

MODEL-VIEW-CONTROLLER

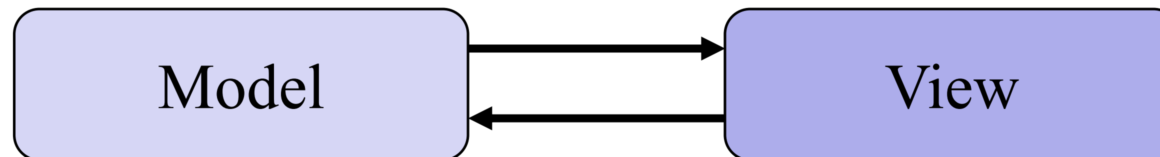
CEU

Model-View-Controller Design Pattern

- Most web applications use the Model-View-Controller pattern.
- Web application architectures:
  - Early web application architectures did not have much organization on the server side.
    - Almost always just fetching static web pages.
    - No separation of data from its presentation.
    - As applications became richer, server-side scripts became more complicated, and the applications became very difficult to maintain.
  - The Model-View-Controller (MVC) design pattern provides a means for dealing with this complexity:
    - Decouples data (model) and presentation (view).
    - A controller handles requests, and coordinates between the model and the view.
    - More robust applications, easier to maintain. Separation of concerns.
    - Very similar to the model for desktop applications.

Model-View Structure

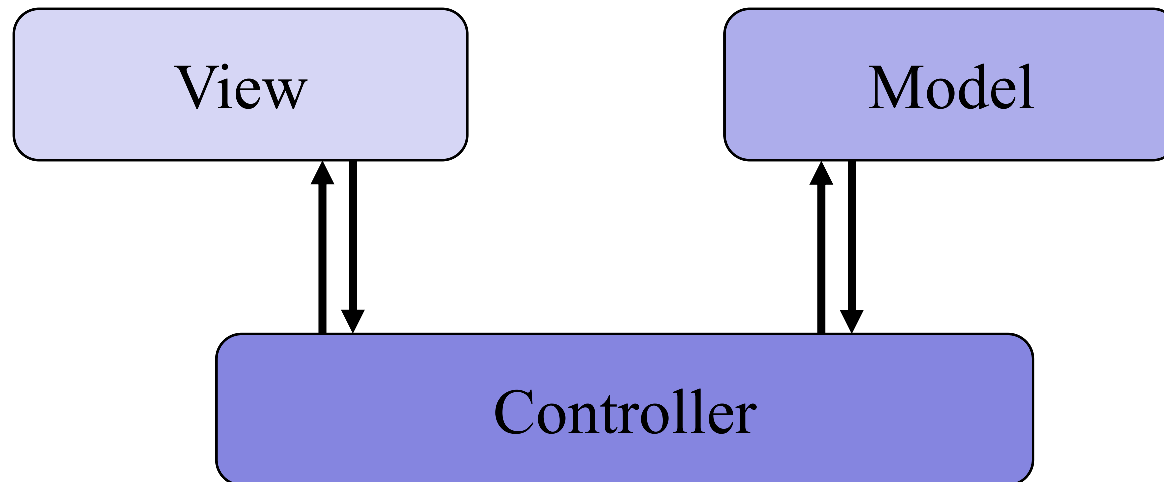
- **Model:** Domain representation of the data with which the application operates and some domain specific meaning that add knowledge to the data.
- **View:** Render the information associated with the model. It is the user interface. Might have multiple views for the same data.
- In the early days, model and view communicated to each other directly.



CEU

Model-View-Controller Structure

- The third piece, the **controller**, was designed to mediate between the model and the view.
  - When the model changes, the controller updates the view.
  - The controller makes changes to the model following orders dictated by the view.

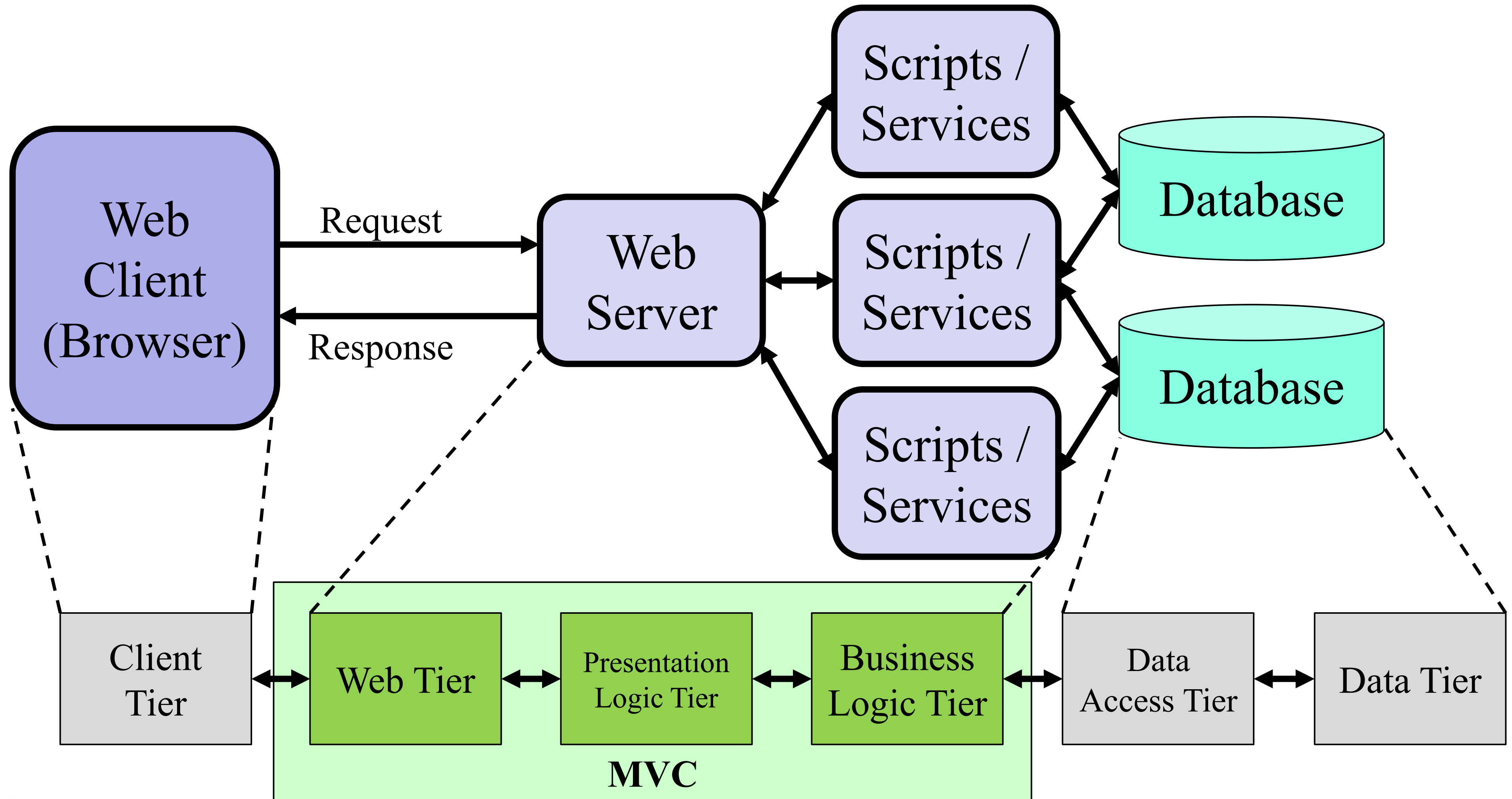


CEU

MVC Control Flow

1. View (user interface) awaits user input.
2. User provides some input (e.g.: clicks a button).
3. Controller handles the input event (an action understandable by the model).
4. Controller notifies the model, possibly changing the model state.
5. Controller notifies the view if it needs to be updated.
6. Back to step 1.

MVC in the 6-tier Architecture



CEU

HTTP

HTTP Protocol Overview

- The Hyper Text Transport Protocol (HTTP) is the foundation of any communication in the web.
- It is an application layer protocol.
- Is built upon **requests** and **responses**.
- It is used to deliver **resources** in distributed hypermedia information systems.
- In order to build and debug web applications, it is vital to have a good understanding of how HTTP works.

HTTP – Resources

- **Hypertext:** Text, marked up using HyperText Markup Language (HTML), possibly styled with CSS, and containing references (hyperlinks) to other resources.
- **Hypermedia:** The logical extension of hypertext to graphics, audio and video.
- **Hyperlinks:** Define a structure over the Web.
- **Scripts:** Code that can be executed on the client side.

HTTP – Background

- Initially, with HTTP/0.9, a client could only issue GET requests, asking a server for a resource. The server then returns the contents of the requested file (the response was required to be HTML).
- The HTTP/1.0 protocol, introduced in 1996, extended HTTP/0.9 to include request headers along with additional request methods.
- HTTP/1.1 included the following improvements:
  - Faster response, by allowing multiple transactions to take place over a single persistent connection.
  - Faster response and bandwidth savings, by adding cache support.
  - Better support for dynamically-generated content through chunked encoding, which allows a response to be sent before its total length is known.
  - More efficient use of IP addresses, multiple domains can be served from a single IP address.
  - Support for proxies.
  - Support for content negotiation.
- HTTP/2, the second major update of the protocol, was published in 2015.
  - Data compression of HTTP headers.
  - Server push technologies.
  - Pipelining of requests.
  - Multiplexing multiple requests over a single TCP connection.

HTTP – Basics

- HTTP has always been a **stateless protocol**.
  - Server not required to retain information related to previous client requests.
  - Each client request is executed independently, without any knowledge of the client requests that preceded it.
  - Difficult for web applications to respond intelligently to user input, that is, to create the interactivity that users expect.
  - Cookies, sessions, URL encoded parameters and a few other technologies have been introduced to address this issue.

HTTP – Request

- An **HTTP request** is a message from the client to the server, with the former asking the latter to do something.
- An HTTP request message is made up of three parts:
  - Request line
  - Header
  - Message body

HTTP – Request – Request Line

- The request line identifies the **resource** and the desired **action** ("request", "verb" or "method") that should be applied to it.
  - The resource is typically identified by a Universal Resource Identifier (URI). A URL (Uniform Resource Locator) is a specific type of URI.
  - There are nine request types that can be specified. The most important five are:
    - **HEAD:** The response the resource would supply to a GET request, but without the response body.
    - **GET:** The response is a representation of a resource.
    - **POST:** Submits data to the resource and the result may be the creation of a new resource, or the update of an existing one.
    - **PUT:** Submits a representation of a resource.
    - **DELETE:** Deletes the resource.
  - The other request types are TRACE, OPTIONS, CONNECT and PATCH.

HTTP – Request – Request Line

- HEAD, GET, OPTIONS and TRACE are referred to as **safe methods**. They should not produce side effects on the server.
  - If a GET method is implemented in a safe way, a browser can make arbitrary GET requests without modifying the state of a web application. This means that these requests can be cached.
- The POST, PUT and DELETE methods may cause side effects on the server, they are **not considered safe**.
- Furthermore, the PUT and DELETE methods should be **idempotent**, multiple identical requests should have the same effect as a single request.
- All safe methods are idempotent, since they don't change the state of the server.

HTTP – Request – Header

- The message header is the primary part of an HTTP request.
- Header fields syntax:
  - *Filed_name: field_value*
  - Example:

```
Accept: text/plain
Accept-Language: en-US
```

- Fields may be any application-specific strings, but a core set of fields is standardized by the Internet Engineering Task Force (IETF).
- An HTTP message header must be separated from the message body by a blank line.

CEU

HTTP – Request – Body

- The message body in an HTTP request is optional.
- It is typically included if there is user-entered data or there are files being uploaded.
- If an HTTP request includes a body, there are usually header fields that describe the body.
  - Example:

```
Content-Type: text/html
Content-Length: 3495
```

CEU

HTTP – Response

- An **HTTP response** is the answer that the server sends to a petition received by a client.
- An HTTP response message is similar to a request message, it also consists of three parts:
  - Status line (also called response line)
  - Header
  - Message body
- Since HTTP/1.1 the server no longer closes the connection after sending a response.

HTTP – Response – Status Line

- The status line is the first line of the response provided by the server.
- It is made up of three parts:
  - The HTTP version, in the same format as in the message request.
  - A response status code that provides the result of the request.
  - A reason phrase in English that describes the status code.
- Example:

```
HTTP/1.1 200 OK
```

- The status codes associated with the status line belong to one of five categories:
  - **1xx (Provisional Response):** A provisional response that requires the client to take additional actions to continue.
  - **2XX (Successful):** The server successfully processed the request.
  - **3XX (Redirected):** Further action (usually a redirection) is needed to fulfill the request.
  - **4XX (Request Error):** There was likely an error in the request which prevented the server from being able to process it.
  - **5XX (Server Error):** The server had an internal error.

CEU

HTTP – Response – Header

- The header allows the server to pass additional information that cannot be placed in the status line.
- Header fields give information about the server and how to access the resource identified by the request URI.
- It has the same syntax as the request headers. Some field names are:
  - Accept-Ranges – Allows the server to indicate its acceptance of range requests for a resource.
  - Age – Estimate of the amount of time since the response was generated at the origin server.
  - Location – Redirects to a location other than the request URI for request completion or identification of a new resource.
  - Proxy-Authenticate – Allows the client to identify itself (or its user) to a proxy that requires authentication.

HTTP – Response – Body

- The message body must be preceded by a blank line.
- The response to a HEAD request does not include a message body. All other response do, although it might be a zero length message body.
- The requested resource, that is, the actual web page, is included in the message body of the response.

HTTP – Sessions and Cookies

- HTTP is stateless, but we need to keep track of a **session**.
- A session is the time span between the moment a user starts to use an application and finishes doing so.
- The HTTP client (usually a browser) establishes a TCP connection with a server (typically port 80) and initiates a request.
- The HTTP server, listening on the port, receives the request, processes it, and sends back a response. Since HTTP is stateless, the response includes a **session ID**, and the server may store user information related to this session.
- When the browser makes another request, and includes the session ID, the server can reference previously stored information and respond accordingly.
- Usually, there is a timeout that closes a session if no new requests from it are received.

HTTP – Cookies

1. The session ID is typically a long randomly generated string sent back to the browser as a **cookie**.
2. The cookie is stored in the browser and sent back to the server with **every request**.
3. Additional data, related to the session itself, can be stored in the cookie.
4. If not managed properly, this information can lead to security vulnerabilities.
5. It's common in web applications to provide the session ID in the cookie, but to keep other information in a **session store** on the server side.
6. A cookie can only store about 4KB of data, so only a little amount of information can be stored.
7. Secret information (like passwords) should **never** be passed through cookies, since they can be easily snooped.
8. The session store uses the session ID to keep track of data.

WEB APPLICATION FRAMEWORKS

Framework

- Whole environment aimed at programming for a specific domain.
- Software application framework:
  - Universal, reusable software environment that provides particular functionality as part of a larger software platform to facilitate development of software applications, products and solution.
  - One size fits all solution for a domain of problems that the programmer can customize to solve a particular problem within that domain.
- Application framework parts:
  - **Frozen spots:** The parts that you generally don’t change in any instantiation of the framework. Define the overall architecture, the basic components and relationships between them, the application infrastructure (“plumbing”).
  - **Hot spots:** The parts where the programmers are supposed to add their own code. The means through which you extend the behavior of the framework. This is what gives the final application its specific functionality.

Application Frameworks

- Benefits:
  - Programmers get a lot of functionality they don’t have to think about (frozen spots).
  - Programmers can focus their creativity on the unique requirements of the application.
  - Takes more or less the same time to build a particular type of application from scratch than learning a framework for doing so. But you only need to learn once.
  - A good framework can be extended and improved.
  - A framework servers to enforce good programming and design patterns.
  - **Don’t fight the framework!**

Web Application Frameworks

- Frameworks designed to support the development of web applications that generally includes:
  - Database support.
    - Connect to the database backend.
    - Update the schema when the web app changes.
  - Templating for generating dynamic web content.
    - Consistent look and feel for the application.
    - Populating the web page dynamically facilitates keeping what's shown consistent with the database.

Web Application Frameworks

- Frameworks designed to support the development of web applications that generally includes:
  - HTTP session management and a web server.
    - Tracking different users through a session-id transmitted with every request.
    - Default web server and other software that works out of the box.
  - Built-in testing framework.
    - Testing is **hugely** important.
    - Frameworks should support continuous testing.
- Most of them also include:
  - Support for internationalization.
  - Scaffolding for creating a MVC structure.
  - Support for the development of REST APIs. We will see what REST is in the services unit.

Web Application Frameworks

"The point is that the cost per request is plummeting, but the cost of programming is not. Thus, we have to find ways to trade efficiency in the runtime for efficiency in the 'thought time' in order to make the development of applications cheaper. I believed we've long since entered an age where simplicity of development and maintenance is where the real value lies."

David Heinemeier Hansson, creator of Ruby on Rails.

Popular Web Application Frameworks

- All of these are open source:
  - **Play (Java and Scala)**
  - Ruby on Rails (Ruby)
  - ASP.NET MVC (Microsoft)
  - Django (Python)
  - Sinatra (Ruby)
  - Symfony (PHP)
  - Sails.js (Node.js, JavaScript)
  - Many other frameworks built on top of Node.js
- WordPress, Drupal and Joomla! are content management systems (CMS), not web application frameworks.

Middleware

- Middleware is the software component that sits between two other software components.
- The term was used first in distributed systems. A web application is a special case of a distributed system.
- In a web application, the middleware is a set of high level abstractions used to bind together the functionality that receives and responds to requests.
- The MVC pattern is implemented over the middleware. It is the "software glue" that binds together the services that allow the client and the server to interact, or the "dash" in the term "client-server".
- Provides services to applications beyond those available from underlying operating systems.
- Allows multiple processes running on different machines to interact, where natively they would not be able to. You don't need to dig into the intricacies of connecting several computers, just understand the middleware interfaces.

Middleware Frameworks

- Middleware frameworks have been created for connecting web application frameworks to application servers.
- Application developers (you) generally do not modify the services provided by the middleware framework. Middleware framework APIs are mainly used by the developers of web application frameworks, and are usually not exposed by the framework.
- A web application framework typically uses several middleware solutions, bound together to provide the combined set of features.
- A framework by itself can be considered a middleware.

Middleware in Web Applications

- The use of middleware makes it easier for software developers to build web applications. Provides services that you don't want to program, like the connection to a server, caching of pages, HTTP connections, etc.
- Typical middleware components in web applications:
  - Web servers
  - Application servers
  - Session management services
  - Proxy services
  - Load balancing services
  - Application frameworks