Backend Web Frameworks

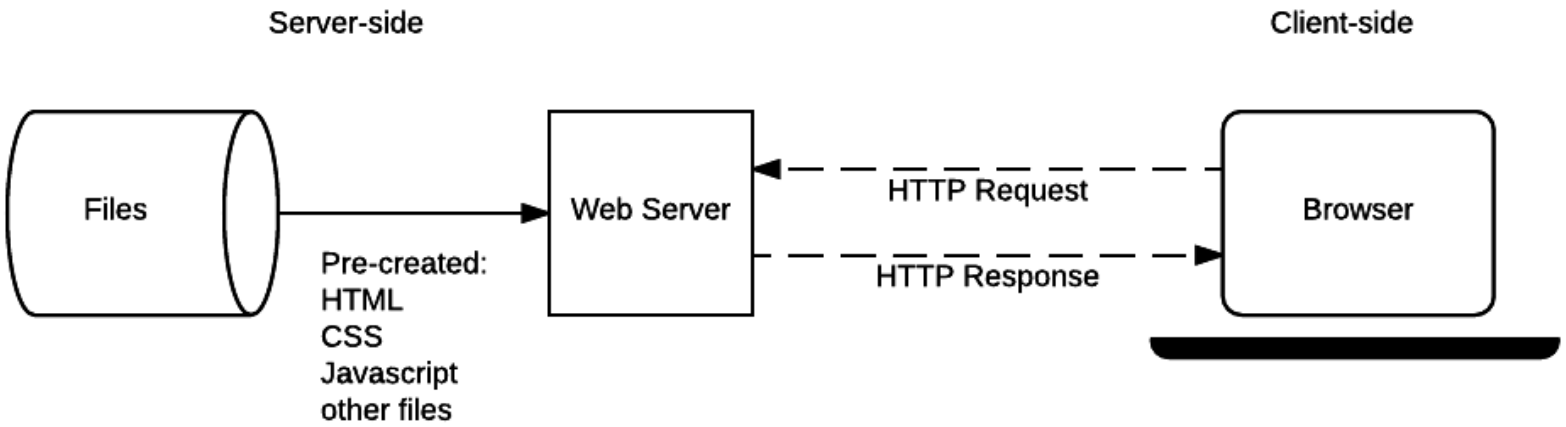
UA.DETI.IES - 2019/20

Main topics

- Web application
  - Static and dynamic
- Frontend development
  - HTML, CSS, JavaScript, JavaScript libraries
  - Frameworks
- Backend development
  - Frameworks
  - N-Tier
  - MVC

Static sites

- Returns the same hard-coded content from the server whenever a particular resource is requested.

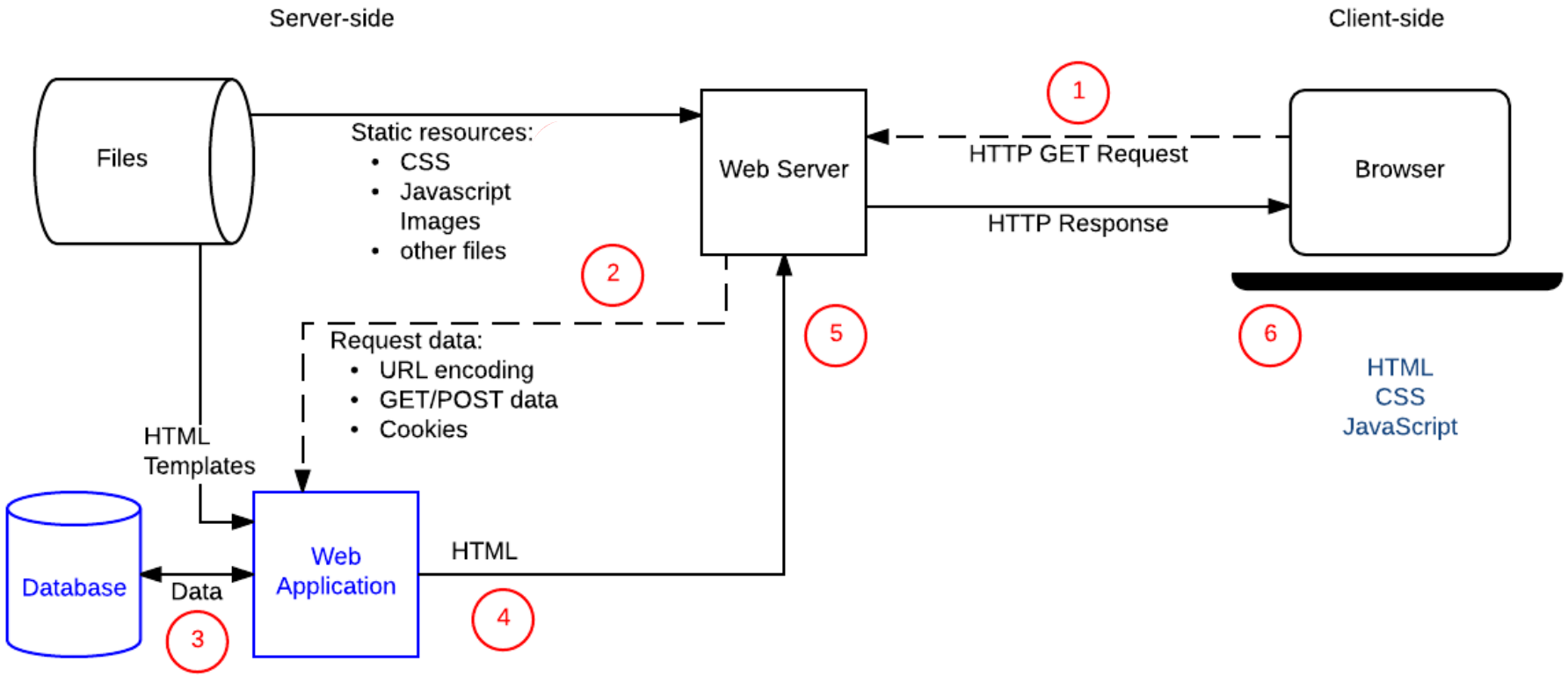


https://developer.mozilla.org/en-US/docs/Learn/Server-side

Dynamic sites

- The response content can be generated dynamically, only when needed.
  - HTML pages are normally created by inserting data from a database into placeholders in HTML templates
- A dynamic site can return different data for a URL
  - based on information provided by the user or stored preferences and can perform other operations as part of returning a response (e.g. sending notifications).
- Most of the code to support a dynamic website runs on a web server.
- Creating this code is known as **server-side** programming or **back-end** programming.

Dynamic sites



https://developer.mozilla.org/en-US/docs/Learn/Server-side

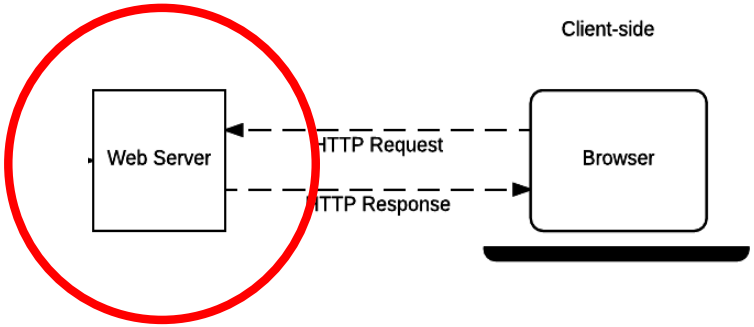
Why did we need Backend

- Programs execute binary code
  - Well, also text (interpreted languages)

```
Hello world!
```

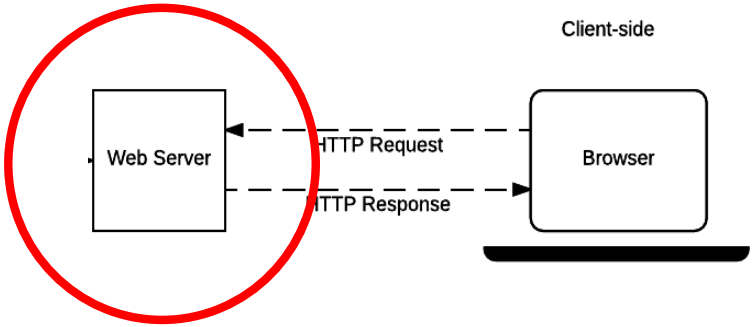
- But, a browser understands HTML, not binary code
  - Well, some other scripts (e.g. JavaScript)

```
<!DOCTYPE html>
<html><head>
<title>Page Title</title>
</head><body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```



Why did we need Backend

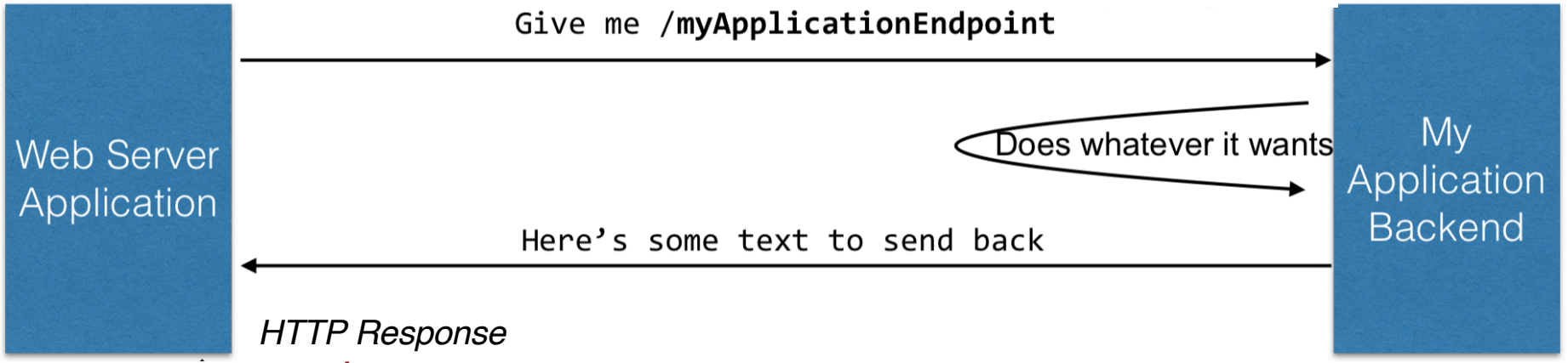
- Dynamic content
  - Adapt typical server-side applications to Web browsers
- Performance
  - Avoid duplicating computation (do it once and cache)
  - Do heavy computation on more powerful machines
  - Do data-intensive computation "nearer" to the data
- Security
  - Validation, security, etc. that we don't want to allow users to bypass



The “good” old days of backends

HTTP Request

```
GET /myApplicationEndpoint HTTP/1.1
Host: www.ua.pt
Accept: text/html
```



HTTP Response

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html><head>...
```

History of Backend Development

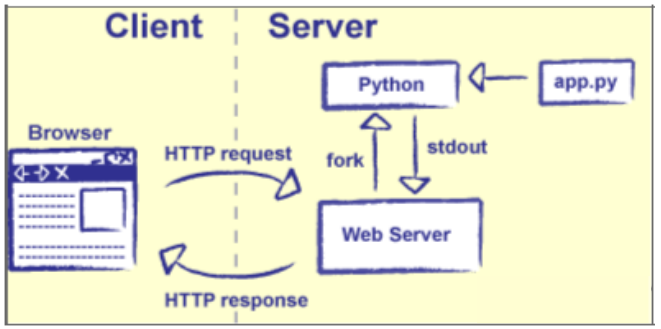
- The Common Gateway Interface (**CGI**) was the first technology that enable dynamic web applications.
  - To call a CGI application to get control, we specify the name of the application in the uniform resource locator (URL)

```
<FORM METHOD=POST ACTION=http://www.mybiz.com/cgi-bin/formprog.pl>
```

- The CGI protocol specifies:
  - How a web server passes information to a program
  - How that program passes information back to the web server
- CGI does not specify:
  - A particular language
    - You can use Fortran, the shell, C, Java, Perl, Python…
  - How the web server figures out what program to run
    - Each web server has its own rules

A simple example: Hello, CGI

- Simplest possible CGI pays no attention to query parameters or extra data
  - Just prints HTML to standard output, to be relayed to the client
  - Along with a Content-type header to tell the client to expect HTML…
    - …and a blank line to separate the headers from the data




```
1 #!/usr/bin/env python
2 # Headers and an extra blank line
3 print 'Content-type: text/html'
4 print
5 # Body
6 print '<html><body><p>Hello, CGI!</p></body></html>'
```

```
Content-type: text/html

<html><body><p>Hello, CGI!</p></body></html>
```

History of Backend Development

- A following alternative to CGI, was Microsoft's Active Server Page (**ASP**) and Java Server Pages (**JSP**)
  - a script embedded in a Web page is executed at the server before the page is sent.

```
<html>
<head><title>First JSP</title></head>
<body>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
      <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
  <%
    } else {
  %>
      <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
  <% } %>
  <a href="<%= request.getRequestURI() %>"><h3>Try Again</h3></a>
</body>
</html>
```

```
<html>
<h2>You'll have a luck day!</h2>
<p>(0.987)</p>
<a href="first.jsp"><h3>Try Again</h3></a>
</html>
```

The new days of backends

- In the middle of the 2000s, the first modern full-stack **server-side frameworks** for web development started to appear
  - e.g. ASP.NET, J2EE, Ruby on Rails, Symfony, Django.
  - … and multiple frontend libraries, e.g. jQuery, Mustache.
- More recently, full-stack **client-side frameworks** for web development emerged
  - e.g.,
    ReactJS,
    AngularJS,
    Vue.js,
    …

Frameworks

Definition 1 (structure)

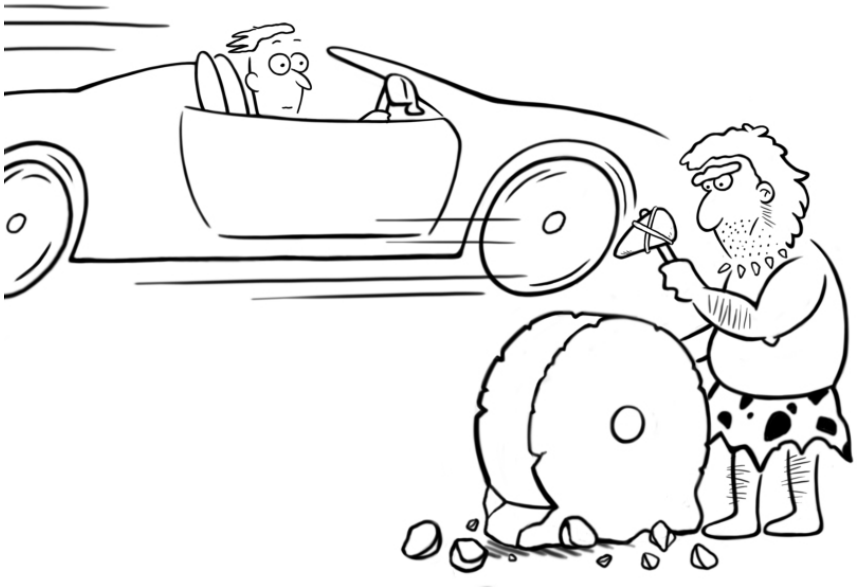
- A framework is a **reusable design** of all or part of a system that is represented by a set of abstract classes and the way their instances interact.

Definition 2 (purpose)

- A framework is the skeleton of an application that can be **customised** by an application developer.

Frameworks

- Don't reinvent the wheel
- And don't run from it



https://boarsheadeastcheap.com/2018/04/04/not-reinventing-the-wheel-after-all/

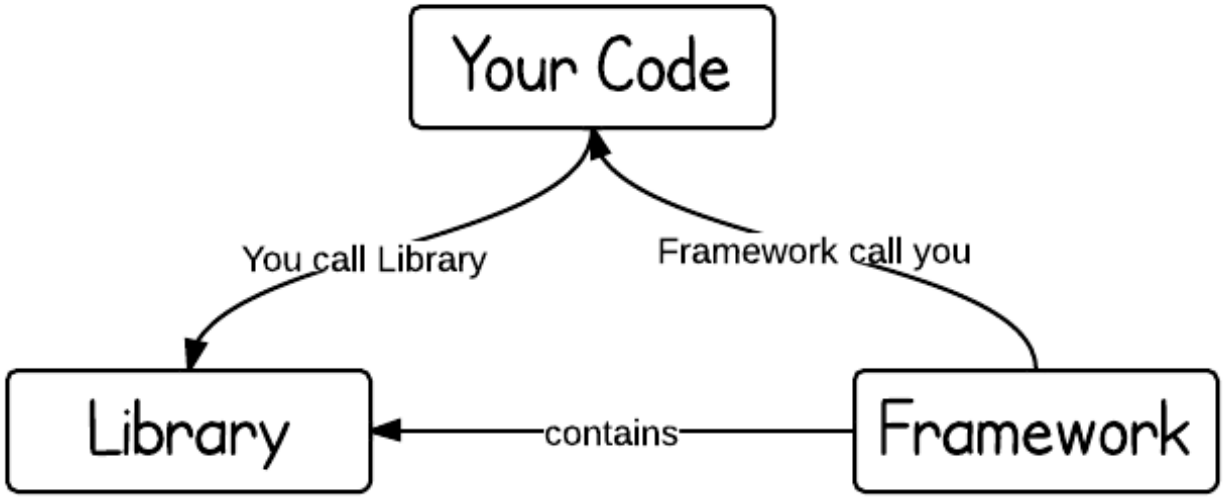
Software Frameworks

- A framework provide a generic software foundation over which custom application-specific code can be written.
  - They often include multiple libraries, in addition to tools and rules on how to structure and use these components.

- Frameworks differ from other class libraries by reusing high-level design
  - Libraries are used by the application-specific code to support specific features.
  - Frameworks control the application flow and call application-specific code.

Frameworks and Libraries

- Libraries
  - *call the functions in the API to do things for you*
- Frameworks
  - *don't call us, we'll call you*
  - Inversion of Control containers



https://www.programcreek.com/2011/09/what-is-the-difference-between-a-java-library-and-a-framework/

Software Frameworks

- Frameworks are normally **domain-specific**.
  - Time to market is increasingly important, and is the main reason many companies build frameworks.
- Frameworks are a form of **design reuse**.
  - Although they can be thought of as a more concrete form of a pattern, frameworks are more like techniques that *reuse both design and code*.
  - Design reuse has advantages over code reuse:
    - can be applied in more contexts
    - applied earlier in the development process, and so can have a larger impact.

Software Frameworks

Advantages

- Powerful and complex
- Implementation speed
- Tested, proven solutions
- Access to expertise and off-the-shelf solutions
- Maintenance (i.e. updates, patches)

Disadvantages

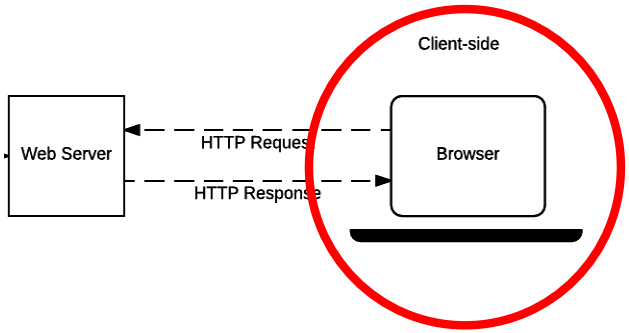
- Difficult to learn
- Lower performance
- Reduced independence
- Dependence on external entities
- Technological lock-in

Web Development Frameworks

https://www.scnsoft.com/blog/web-application-framework

Client-side frameworks (Frontend)

- AngularJS
- ReactJS
- Backbone.JS
- Vue.JS
- Ember.JS
- …



Client-side APIs

- Manipulate **documents** loaded into the browser
  - e.g., DOM (Document Object Model) allows to manipulate HTML and CSS.
- **Fetch data** to update small sections of a webpage
  - e.g., AJAX has a huge impact on the performance and behaviour of sites
- **Drawing** and manipulating graphics
  - e.g., Canvas and WebGL
- Client-side **storage**
  - e.g. IndexedDB API allow to save an app state between page loads, or when device is offline.

Server-side Frameworks (Backend)

- Frameworks can be grouped in three types:
  - **Micro** Frameworks — focused on routing HTTP request to a callback, commonly used to implement HTTP APIs.
  - Flask (Python), Express.js (Node.js), Spark (Java)
  - **Full-Stack** Frameworks — feature-full frameworks that includes routing, templating, data access and mapping, plus many more packages.
  - Django, ASP.NET, Spring, ..
  - **Component** Frameworks — collections of specialized and single-purpose libraries that can be used together to make a a micro- of full-stack framework.
  - Angular (google), React (facebook), Vue,..

Server-side Frameworks – examples

- Python
  - Django
- JavaScript
  - Express.js
- Ruby
  - Ruby on Rails
- Java
  - Spring
- PHP
  - Symfony
- C#
  - ASP.NET

Backend Frameworks	Pros	Cons
Django	1. Scalable and flexible 2. Great for MVPs 3. Secure 4. Great documentation	1. Not the fastest one 2. Monolithic
ExpressJS	1. Simple 2. Flexibility 3. Packages for API development	1. Many callbacks 2. Unhelpful error messages 3. Not suitable for heavy apps
Ruby on Rails	1. Many tools and libraries 2. Fast development 3. Good for prototyping 4. Test automation	1. Slow boot time 2. Not the best choice for heavy applications 3. Lack of proper documentation
Spring	1. Great for Java apps 2. Easy to cooperate with other programs 3. Flexible	1. Difficult to learn 2. Can be unstable
Symfony	1. Fast Development 2. Reusable code 3. Great documentation	1. Comparatively slow

https://gearheart.io/blog/top-10-web-development-frameworks-2019-2020/

Common Tasks in Web Development

- Manage interface components
- Access control management
- Session and authentication management
- Handle access requests
- Validating inputs
- Error handling
- Access and manipulate data
- …

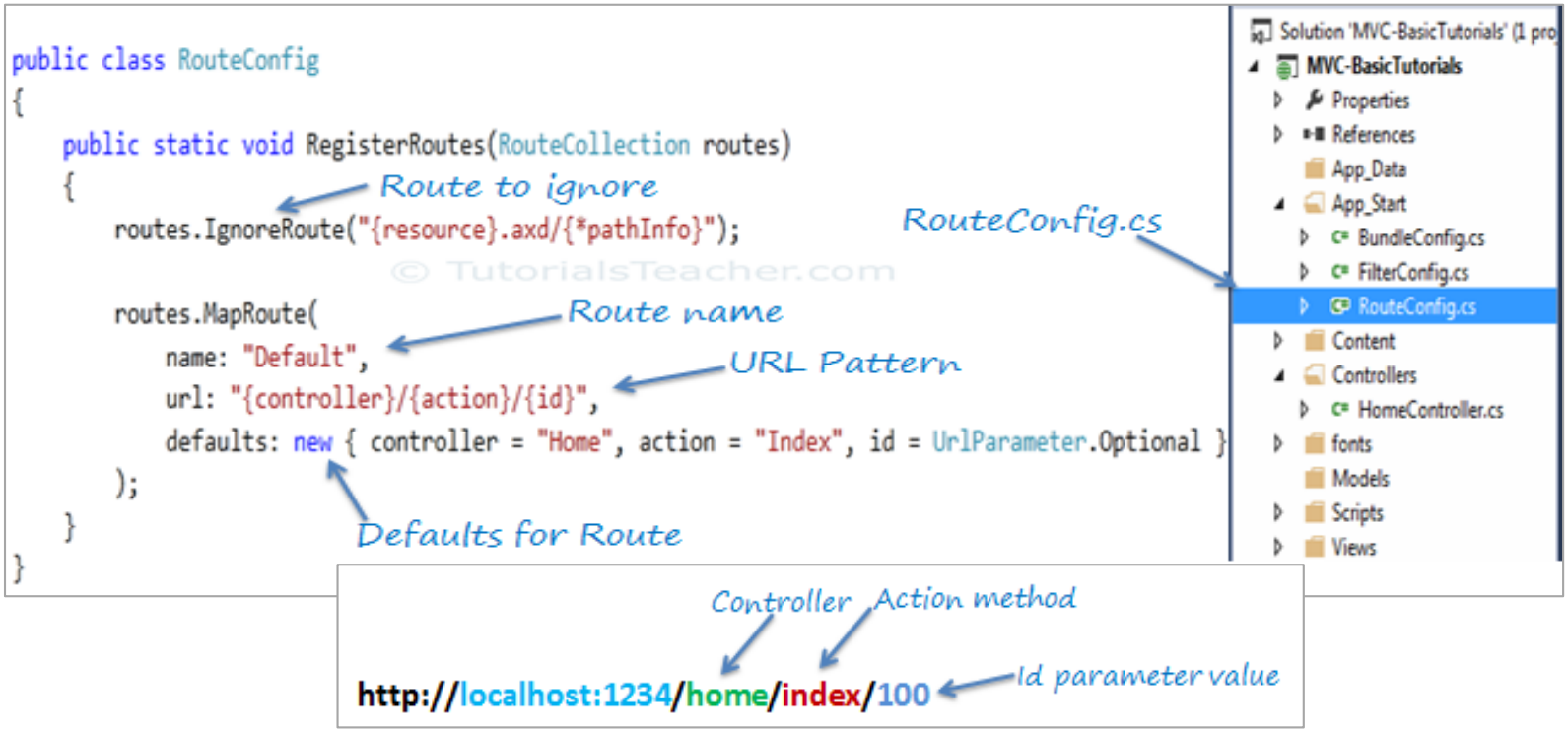
Framework Components

- Core components
  - **Request Routing** — Match incoming HTTP requests to code
  - **Template Engine** — Structure and separate presentation from logic
  - Data Access — Uniform data access, mapping and configuration
- Common components
  - Security — Protection against common web security attacks
  - Sessions — Session management and configuration
  - Error Handling — Capture and manage application-level errors
  - Scaffolding — Quickly generate CRUD interfaces based on data model

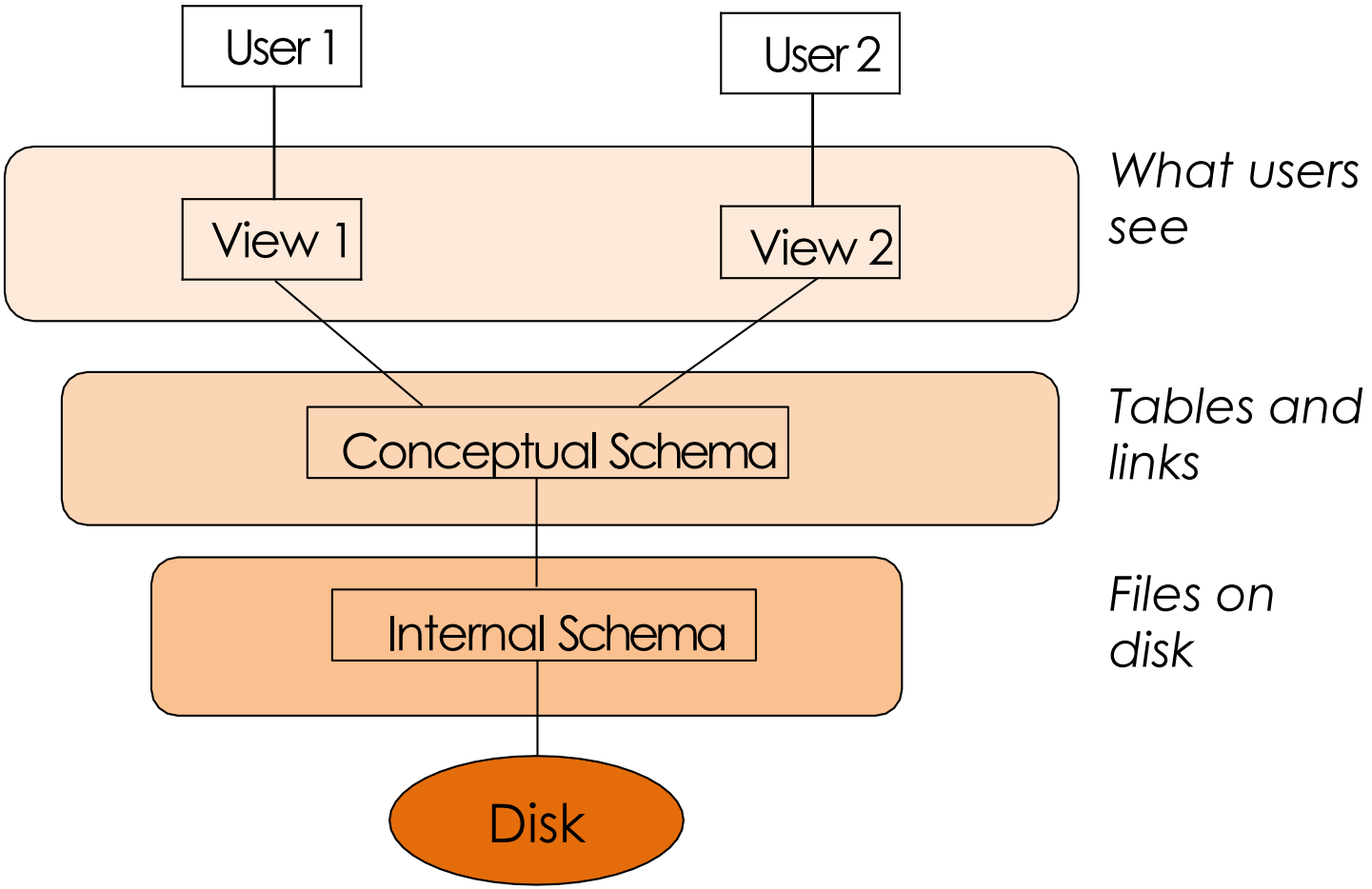
Request Routing

- Request routing maps HTTP access requests to specific functions
  - URL design is handled independently from application code using request routing

ASP.NET example:



Architecture With Views



Data Independence

- **Logical Independence**: The ability to change the logical schema without changing the external schema or application programs
  - Can add new fields, new tables without changing views
  - Can change structure of tables without changing view
- **Physical Independence**: The ability to change the physical schema without changing the logical schema
  - Storage space can change
  - Type of some data can change for reasons of optimization

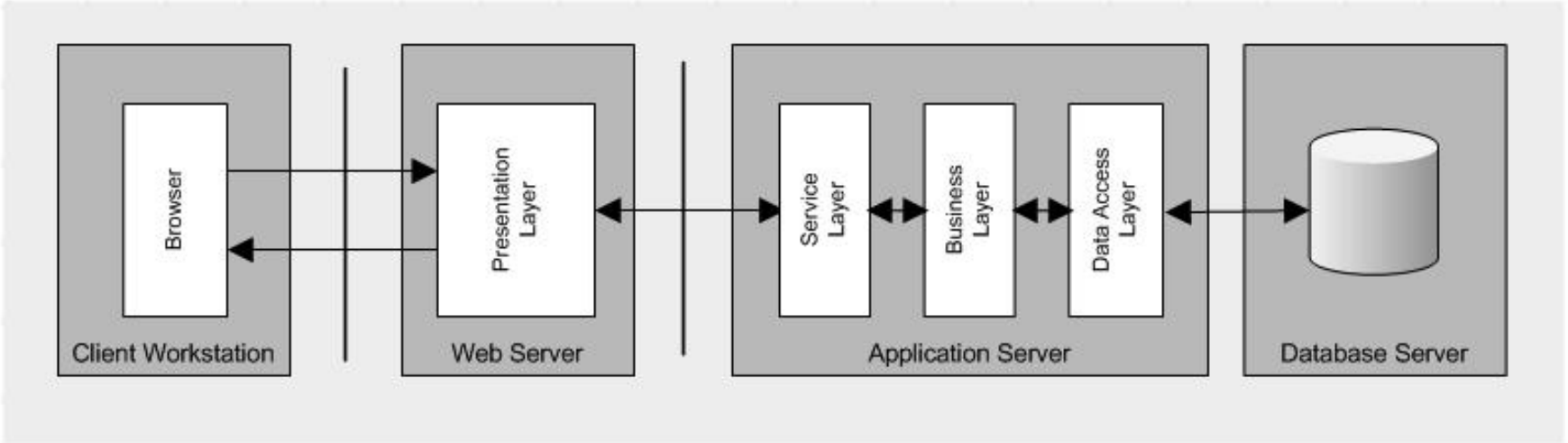
- Solution: Keep the VIEW (what the user sees) independent of the MODEL (domain knowledge)

N-Tier Architectures

- N-tier architectures have the same components
  - Presentation
  - Business/Logic
  - Data
- N-tier architectures try to separate the components into different tiers/layers
  - Tier: physical separation
  - Layer: logical separation

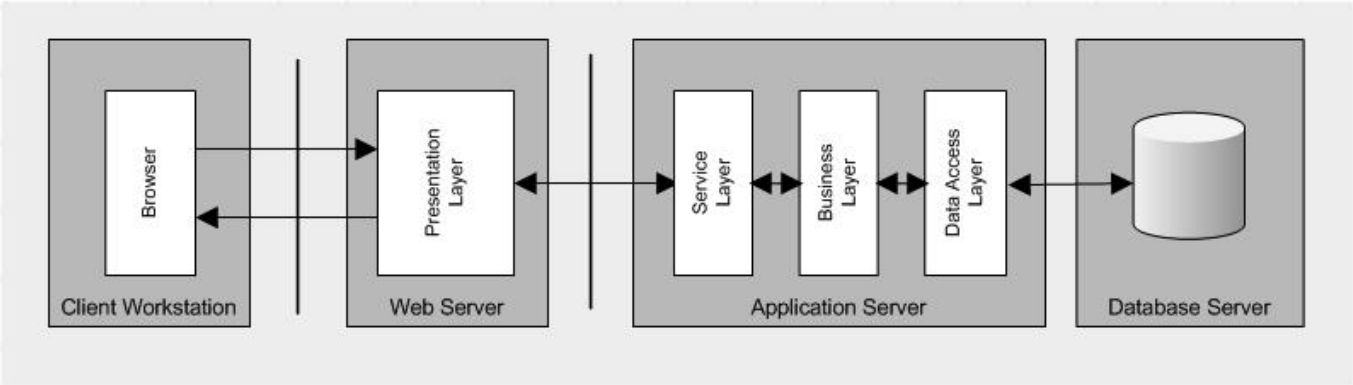
3-Tier Architecture

- Presentation, logic and data layer are disconnected
- Each layer can potentially run on a different machine



http://arun-architect.blogspot.com/2010/12/n-tier-architecture.html

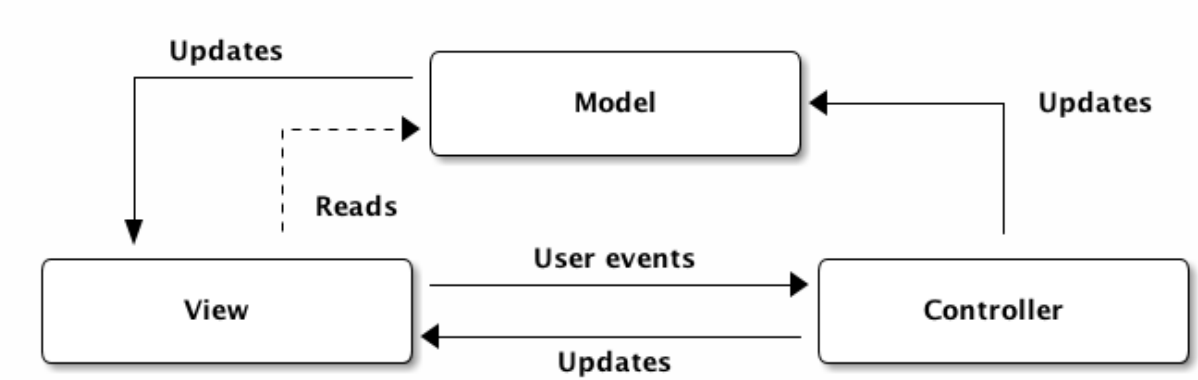
Design Problem



- Need to ?
  - change the look-and-feel
    - without changing the core/logic
  - present data under different contexts
    - e.g., powerful desktop, web, mobile device
  - interact with/access data under different contexts
    - e.g., touch screen on a mobile device, keyboard on a computer
  - maintain multiple views of the same data
    - list, thumbnails, detailed, etc.

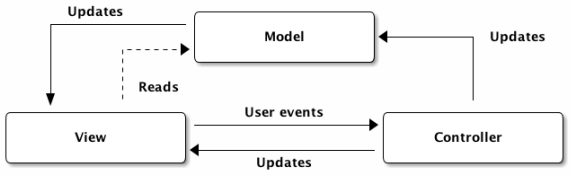
Model-View-Controller (MVC)

- Many Backend Frameworks follow or are an evolution of a design pattern called Model-View-Controller, or MVC.
  - Separates core functionality from the presentation and control logic that uses this functionality
  - Allow multiple views to share the same data model
  - Make supporting multiple clients easier to implement, test, and maintain



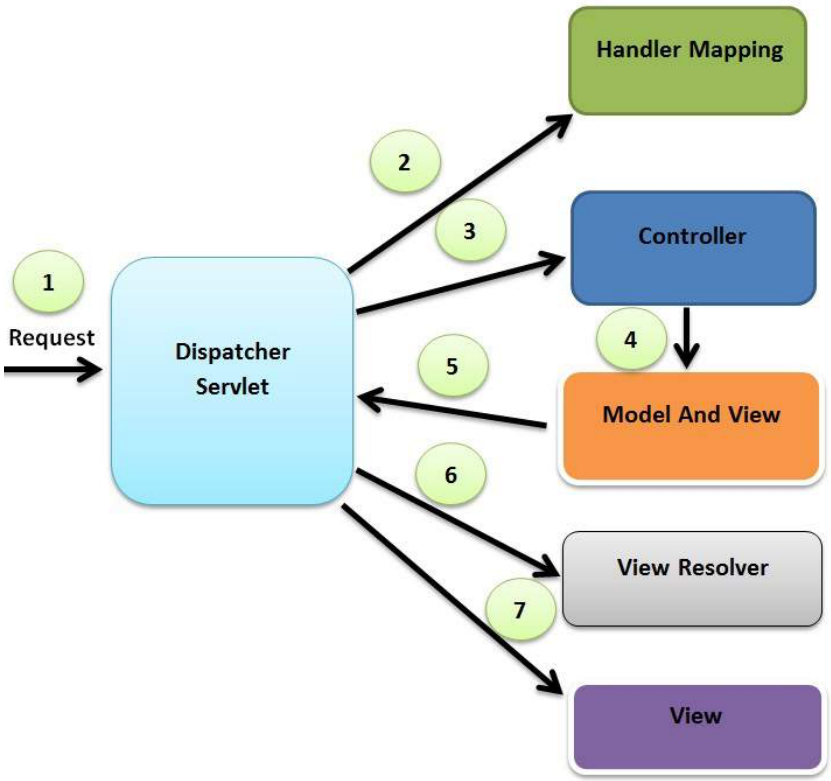
Model-View-Controller (MVC)

- **Model**: The program representation of the object/database
  - e.g. Users, Products, Cars



- **View**: The visual representation of your data. This is what the web app user sees
  - i.e. HTML, CSS, and JavaScript
- **Controller**: The go-between Model and View
  - One way: takes input from the View and gives it to the Model to act upon.
  - Other way: Gathers data from Model(s) and gives it to the View to incorporate in the visual representation.
  - Examples: New User signup, listing all products that are less than some dollar amount, redirecting a user.

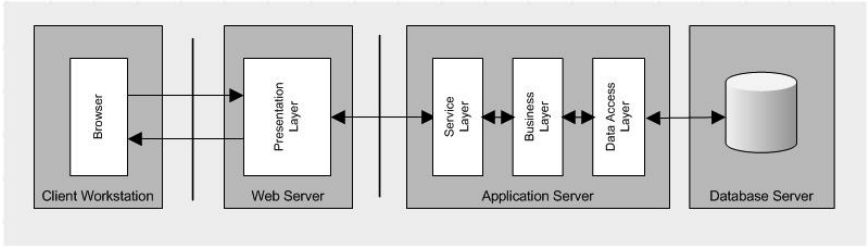
Spring MVC Architecture

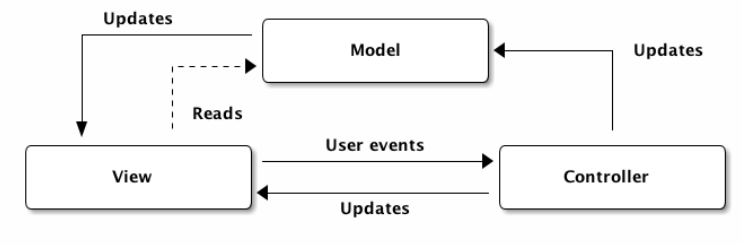


1. Based on the Servlet Mappings which we provide in our web.xml, the request will be routed by the Servlet Container to our **DispatcherServlet**
2. Once the request is received, the DispatcherServlet will take the help of **HandlerMapping** which has been added in the Spring Configuration file and get to know the Controller class to be called for the request received.
3. Now the request will get transferred to the **Controller**, the Controller then executes the appropriate methods …
4. and returns the corresponding **ModelAndView** object …
5. to the **DispatcherServlet**.
6. The DispatcherServlet will send the Model received to the **ViewResolver** to get the view page.
7. Finally, the DispatcherServlet will pass the Model to the **View** page and the page will be rendered to the user

https://javainterviewpoint.com/spring-mvc-flow-diagram/

3-tier vs. MVC Architecture





- **Communication**
  - 3-tier: The presentation layer never communicates directly with the data layer-only through the logic layer (linear topology)
  - MVC: All layers communicate directly (triangle topology)
- **Usage**
  - 3-tier: Mainly used in web applications where the client, middleware and data tiers ran on physically separate platforms
  - MVC: Historically used on applications that run on a single workstation

Template Engine

- Template engines take in tokenized strings and produce rendered strings with values in place of the tokens as output.
  - Templates are typically used as an intermediate format written by developers to programmatically produce one or more desired output formats, commonly HTML, XML or PDF.
- Examples
  - Java Server Pages
  - Thymeleaf
  - FreeMarker
  - Groovy
  - Jade4j
  - JMustache
  - Pebble
  - ...

JSP Templates

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<html>
    <head>
        <meta http-equiv="Content-Type"
          content="text/html; charset=ISO-8859-1">
        <title>User Registration</title>
    </head>
    <body>
        <form:form method="POST" modelAttribute="user">
            <form:label path="email">Email: </form:label>
            <form:input path="email" type="text"/>
            <form:label path="password">Password: </form:label>
            <form:input path="password" type="password" />
            <input type="submit" value="Submit" />
        </form:form>
    </body>
</html>
```

Thymeleaf Templates

```
<html>
    <head>
        <meta charset="ISO-8859-1" />
        <title>User Registration</title>
    </head>
    <body>
        <form action="#" th:action="@{/register}"
          th:object="${user}" method="post">
            Email:<input type="text" th:field="*{email}" />
            Password:<input type="password" th:field="*{password}" />
            <input type="submit" value="Submit" />
        </form>
    </body>
</html>
```

Jade4j Templates

```
doctype html
html
  head
    title User Registration
  body
    form(action="register" method="post" )
      label(for="email") Email:
      input(type="text" name="email")
      label(for="password") Password:
      input(type="password" name="password")
      input(type="submit" value="Submit")
```

Web_frameworks comparison

- https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks
- https://www.pixelcrayons.com/blog/web/best-web-development-frameworks-comparison/
- https://en.wikipedia.org/wiki/Comparison_of_web_frameworks