Server Side Development

Exam Preparation

Revision

- Next slides present a high-level overview of the main topics covered during the lectures. Please note that this **is not an exhaustive list of all the topics and** their corresponding **subtopics** covered. That means, even though you will not see in the following slides a specific slide title from the lectures the particular information from that slide is still required for the exam!

Revision

- **Dynamic Websites:** Waterfall SW Development model; Attributes of the Web: loose coupling, statelessness, recovery; Client - Server model.
- **Object Oriented:** What is OO Programming? Class, Object, Variables, Inheritance, Encapsulation, Polymorphism.
- **Ruby:** Methods, Blocks, String interpolation “#{value}”, Arrays, Hashes.
- **Web Framework:** What is? Advantages? Disadvantages?
- **MVC concepts:** Model, View, Controller
- **Rails:** Philosophy, Framework, Commands, CRUD - Scaffolding.

Waterfall Model of Software Development

- Requirement analysis and definition: The system's services, constraints and goals are established by consultation with system users
- System and Software Design: Establishes an overall system architecture and describes the fundamental software system abstractions and their relationships.
- Implementation (&Unit Testing): The software design is realized as a set of programs or program units.
- Integration and system testing: The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met.
- Operation and Maintenance: Normally is the longest life-cycle phase. Maintenance involves correcting errors, which were not discovered in earlier stages of the life cycle.

Attributes of the Web

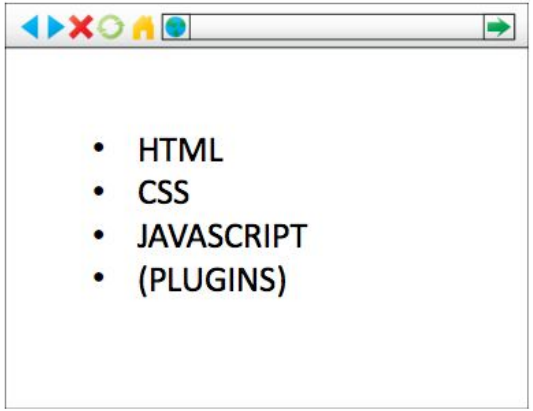
- Loose coupling: Growth of the Web in one place is not impacted by changes in other places. In other words the web can be expanded or added to and not have this addition directly impact on any other aspect of the internet. Perhaps a good way to explain this is by way of a simple example where adding one web page to an application will not cause a web page to be deleted from another application.

- Statelessness: All information required to process a request must be present in that request statelessness allows for easy replication of processes throughout an application. One Web server is replaceable with another supporting horizontal scaling or adding more devices to the application as opposed to more storage in a specific device.

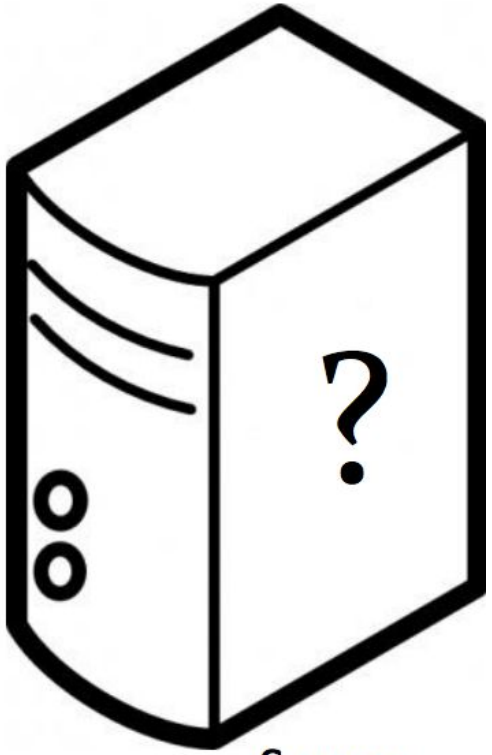
- Recoverable: Information retrieval is repeatable meaning that information can be requested again and again without an possibility of losing or damaging data. For example GET is a safe request such that if the request fails the request can simply be repeated to obtain the required result. This is also supported by HTTP status which can send information outlining the status of the response.

Key elements

Client/Browser



- HTML
- CSS
- JAVASCRIPT
- (PLUGINS)



Server

Client

- AKA Client Side, Frontend
- Limited by what the browsers have
- HTML
  - Tags describing the content of a website
- CSS
  - Style Sheet(s) describing the “look” of a website
- JavaScript
  - Programming language allowing, for example, manipulation of the website page
- Plugins
  - Flash, Java Applets etc.
  - Becoming less common

Server

- AKA Server Side, Backend
- Not limited by what the user has, so an explosion of options!
- (Very!) broadly speaking, there are Web Servers and Application Servers
- Web Servers provide standard web services – serving up web pages over HTTP
- Application Servers extend this to allow processing based on a particular programming language

What is a variable?

- For the examples on the previous slide, we might want to find the type a variable is holding.
- Example
- age = 30
- puts(age.class)

- The output for age.class should be FixNum, a number variable

- Try this for a string example? What do we get as output?

Source: http://www.techotopia.com/index.php/Understanding_Ruby_Variables

Ruby: variable scope

- Scope defines where in a program a variable is accessible. Ruby has four types of variable scope, *local*, *global*, *instance* and *class*. In addition, Ruby has one constant type. Each variable type is declared by using a special character at the start of the variable name as outlined in the following table.

Name Begins With	Variable Scope
`$`	A global variable
`@`	An instance variable
`[a-z]` or `_`	A local variable
`[A-Z]`	A constant
`@@`	A class variable

Source: http://www.techotopia.com/index.php/Ruby_Variable_Scope

Global Variables

- Global variable: can be accessed from anywhere within an application.

```
def myMethod
  puts $x
end
$x = 10
myMethod
```

Instance Variables

- Instance / Object variables: can only be used within the same instance of an object. Example:

```
class Person
  def initialize(name)
    @name = name
  end

  def get_name
    @name
  end
end
```

Local Variables

- [a-z] or _
- A local variable is used in the code block in which it is declared. It is only available in this code block and does not visible anywhere else in the program.
- Example

```
class Person
  def initialize(name, age)
    @name = name
    @age = age
  end

  def getName
   labelname = “Fname:”
   return labelname + @name
  end
end
```

Constant Variables

- Ruby constants are values which, once assigned a value, should not be changed. It is possible to change a constant after it has been assigned, this will create a warning message
- Constants declared outside of a class or module are assigned global scope.

Source: http://www.techotopia.com/index.php/Ruby_Variable_Scope

Class Variables

- A class variable is a variable that is shared amongst all instances of a class. This means that only one variable value exists for all objects instantiated from this class. This means that if one object instance changes the value of the variable, that new value will essentially change for all other object instances.

```
class Person
  @@counter = 0
  def initialize(name, age)
    @name = name
    @age = age
    @@counter=@@counter + 1
  end

  def self.numberPeople
    @@ counter
  end
end
```

Source: http://www.techotopia.com/index.php/Ruby_Variable_Scope

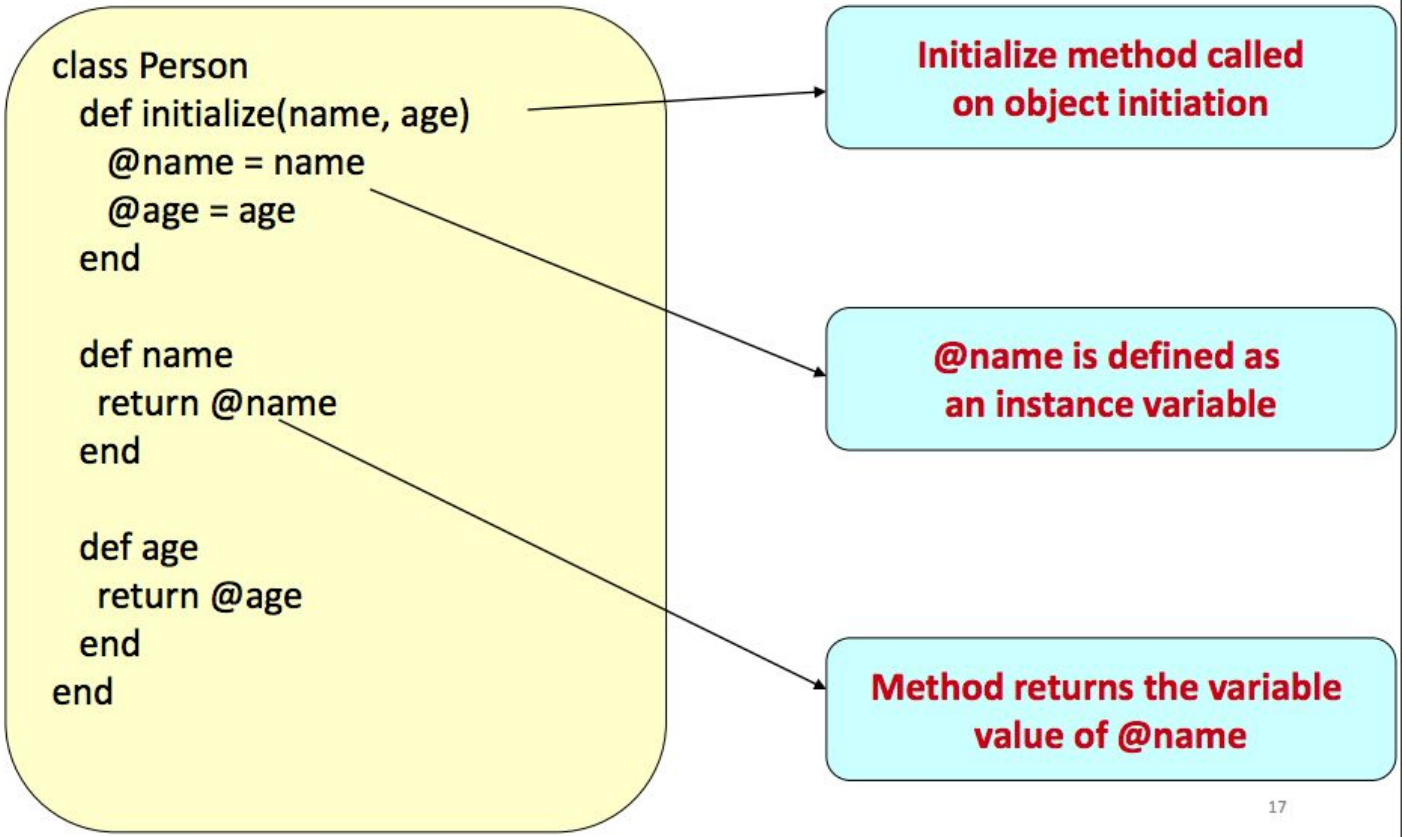
Objects

- Everything in Ruby is an Object.
- This means that we can call methods on all elements within a ruby class, including all data types (unlike Java, which has some primitive data types like int that are not objects)

Object.methodName

Periods are used between the Object and the method being called

Person Object Explained



Person Object Instantiated

```
class Person
  def initialize(name, age)
    @name = name
    @age = age
  end

  def name
   return @name
  end

  def age
   return @age
  end
end
```

Create a file named person.rb and add the following, first the Person class then the code on the right.

```
person1 = Person.new("John",21)
Person2 = Person.new("Jane",27)

puts person1.name
puts Person2.age
```

What output does this generate?

Inheritance

- Inheritance is a relation between two classes.
- *In Ruby, a class can only inherit from a single other class.*

```
class Mammal
  def breathe
    puts "inhale and exhale"
  end
end

class Cat < Mammal
  def speak
    puts "Meow"
  end
end

class Lion < Mammal
  def speak
    puts "Roar!!!"
  end
end

class Turkey < Lion
  def fly
    puts "Im flying"
  end
end
```

Create a file named inherit.rb and add the following, first the classes then the code on the right.

```
dodger = Cat.new
dodger.breathe
dodger.speak

lion1 = Lion.new
lion1.breathe
lion1.speak

turkey = Turkey.new
turkey.speak
turkey.fly
```

Source: http://rubylearning.com/satishtalim/ruby_inheritance.html

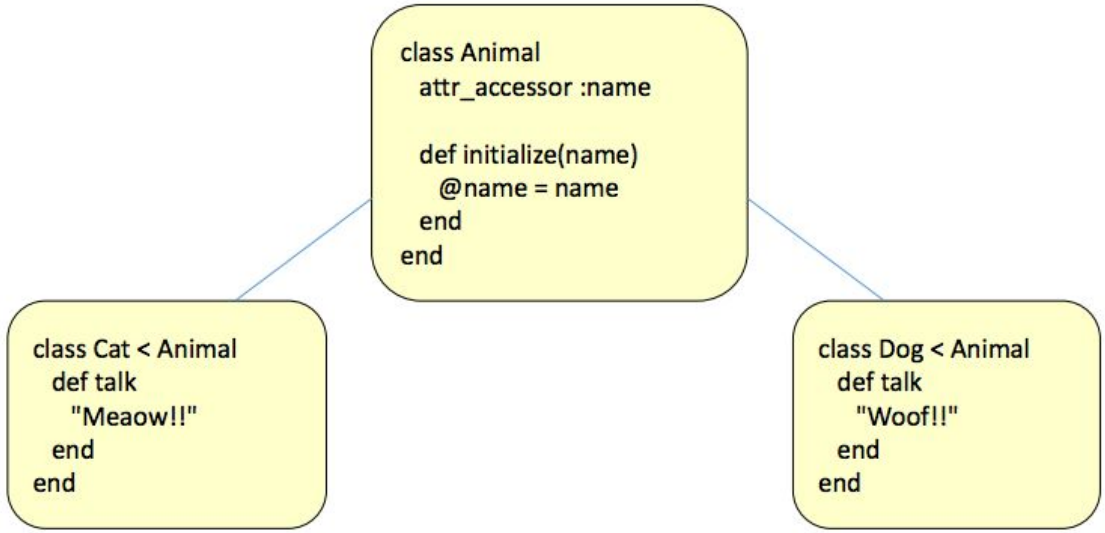
Ruby: Encapsulation

- Encapsulation allows Ruby to allow an object have certain methods and attributes available for public use and other components only available within the class. Example,

```
class Person
  def initialize(name)
set_name(name)
  end

  def name
    @first_name + '' + @last_name
  end

  def set_name(name)
    first_name,last_name = name.split(/\s+/)
    set_first_name(first_name)
    set_last_name(last_name)
  end

 def set_first_name(name)
    @first_name = name
  End

  def set_last_name(name)
    @last_name = name
  end
end
```

Ruby: Polymorphism

- Polymorphism is the concept of writing code that can work with objects of multiple types and classes. For Example, the + method works for adding numbers, joining strings together and adding arrays together. The method that runs depends on the implementation of the method.



Ruby: Polymorphism Example

- To test the above example, we create an array consisting of a Dog and a Cat. Loop through the array and implement the talk method.

```
animals = [Cat.new("Ginger"),Dog.new("Rex")]

animals.each do |animal|
  puts animal.talk
end
```

Ruby: Arrays and Lists

Arrays are used to store collections of objects in ruby. Like most programming languages arrays in ruby start from 0 -> n, where n is the size of the array. Example,

```
x = [1,2,3,4,5,6,7,8,9]
puts x[2]
```

```
x[2] = 20
puts x[2]
```

Arrays and Lists

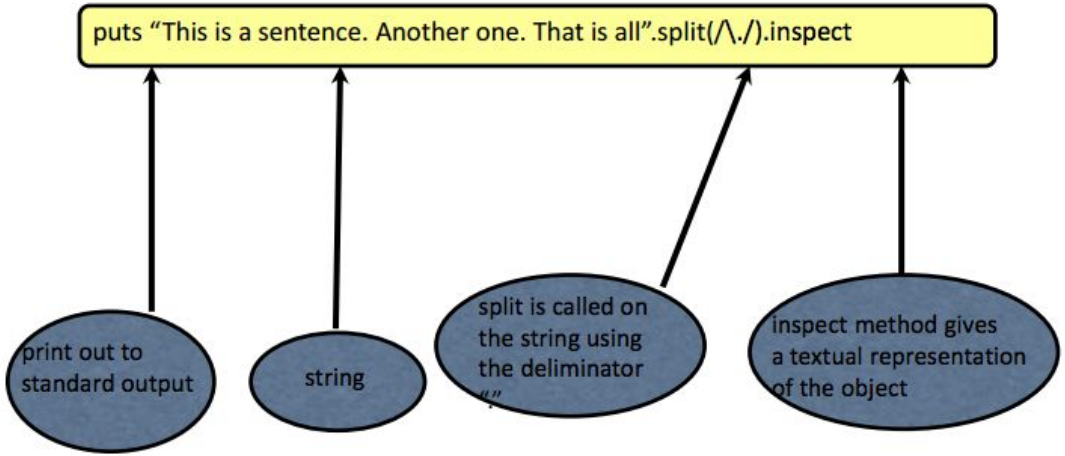
Ruby: Arrays and Lists

Arrays in ruby contain methods that act like a simple stack structure. For Example there exits a push and a pop method, which when implemented interacts like a first in, last out protocol. Example,

```
arr1 = []
arr1 << “Hello”
arr1 << “World”

puts arr1.join(“”)
puts arr1.length

puts arr1.pop
puts arr1.length
```

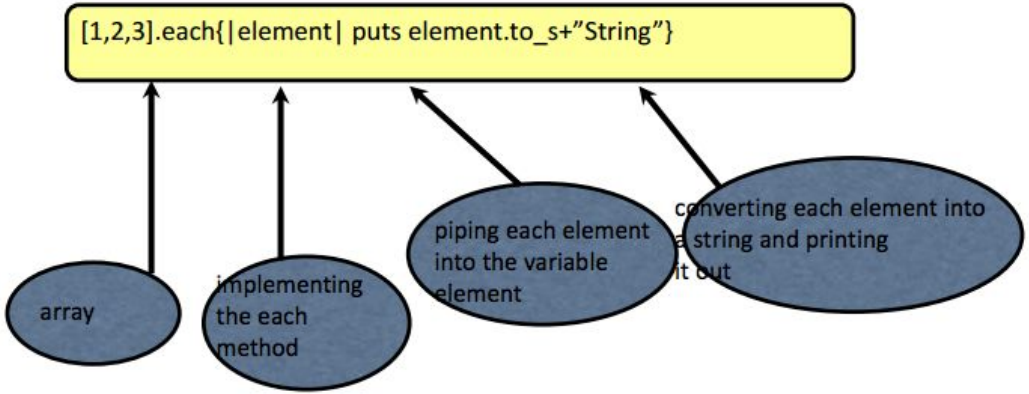
Ruby: Arrays and Lists

Using the *split* method in ruby you can brake a string into the components of an array. Example,



Ruby: Arrays Iteration

Iterating through an array is completed using the each method within Ruby.



Hashes

Ruby: Hashes

Hashes are collections of objects. Unlike, arrays hashes have an associated key with each element and are not always in order, instead they are usually viewed like a dictionary. Example,

```
dictionary = {‘cat’ => “fluffy the cat”, ‘dog’=> “Rex is a dog”}

puts dictionary.size
puts dictionary['cat']
```

Hashes use the *each* method as seen before with arrays for iterating through each element.

```
dictionary = {'cat' => “fluffy the cat”, 'dog'=> “Rex is a dog”}

dictionary.each{|key.value| puts “#{key} : #{value}”}
```

Ruby: Hashes - More Hash methods

Retrieving Keys:

```
p.dictionary.keys
```

Deleting Hash Elements:

```
dictionary.delete("cat")
```

Deleting Hash Elements conditionally:

```
x = {"a"=>20, "b"=>200, "c"=>120, "d"=>2}
puts x.size

x.delete_if {|key,value| value >100}
puts x.size
```

Frameworks

- Usually based on one particular programming language
- Most on based on the MVC design pattern
  - Model. View. Controller.
  - Much abused term, broadly means keep these three things separate from each other:
    - the presentation, GUI code (View)
    - data management code (Model)
    - Everything else code (Controller) separate from each other
  - Separating responsibilities increases flexibility in changing things later

Key technology elements of a framework

- Controlling
  - A programming environment capable of responding to requests
  - May have a build in web server or operate behind a separate web server
- View
  - Some mechanism for producing the actual HTML/CSS/Javascript pages
  - Usually involves templating, i.e. boilerplate web pages with tags for dynamic elements that are replaced with the real values at run time
- Model
  - Some way of storing off and accessing data
  - Might be as simple as a direct connection to a specific database
  - Or may be a full persistence layer that allows seamless swapping of database technology

Key technology elements of a framework

- Controlling
  - A programming environment capable of responding to requests
  - May have a build in web server or operate behind a separate web server
- View
  - Some mechanism for producing the actual HTML/CSS/Javascript pages
  - Usually involves templating, i.e. boilerplate web pages with tags for dynamic elements that are replaced with the real values at run time
- Model
  - Some way of storing off and accessing data
  - Might be as simple as a direct connection to a specific database
  - Or may be a full persistence layer that allows seamless swapping of database technology

What is Rails

- Rails is a web application development framework written in the Ruby language.
- Rails has a pre-defined configuration to help developers create applications in a rapid fashion.
- Rails is opinionated software. It makes the assumption that there is a “best” way to do things, and it’s designed to encourage that way – and in some cases to discourage alternatives.
- There are two ways of doing things, your way and “the Rails Way”. If you don’t follow the Rails Way, thinks will get more difficult!!

Rails

- Rails is a large, heavy web application framework that has lots of features included. Rails can be used for the development of both large and small web applications but is resource intensive when compared to other frameworks. A typical rails application will have lots of files and a tight structure built up of models, views and controllers.

What is Rails?

- An highly productive web-application framework.
- Written using the Ruby programming language.
- You could develop a web application at least ten times faster with Rails than you could with a typical Java framework.
- An open source Ruby framework for developing database-backed web applications.
- Your code and database schema are the configuration!
- No compilation phase required.

Source: http://www.tutorialspoint.com/ruby-on-rails/rails-introduction.htm

Rails - Full Stack Framework

- Includes everything needed to create a database-driven web application using the Model-View-Controller pattern.
- Being a full-stack framework means that all layers are built to work seamlessly together (Less Code).
- Requires fewer total lines of code than other frameworks spend setting up their XML configuration files.

Source: http://www.tutorialspoint.com/ruby-on-rails/rails-introduction.htm

Rails Strengths

- Rails is packed with features that make you more productive, with many of the following features building on one other.

- **Metaprogramming** - Other frameworks use extensive code generation from scratch. Metaprogramming techniques use programs to write programs. Ruby is one of the best languages for metaprogramming, and Rails uses this capability well. Rails also uses code generation but relies much more on metaprogramming for the heavy lifting.

- **Active Record** - Rails introduces the Active Record framework, which saves objects to the database. The Rails version of Active Record discovers the columns in a database schema and automatically attaches them to your domain objects using metaprogramming.

- **Convention over configuration** - Most web development frameworks for .NET or Java force you to write pages of configuration code. If you follow suggested naming conventions, Rails doesn't need much configuration.

Rails Strengths

- **Scaffolding** - You often create temporary code in the early stages of development to help get an application up quickly and see how major components work together. Rails automatically creates much of the scaffolding you'll need.

- **Built-in testing** - Rails creates simple automated tests you can then extend. Rails also provides supporting code called harnesses and fixtures that make test cases easier to write and run. Ruby can then execute all your automated tests with the rake utility.

- **Three environments** - Rails gives you three default environments: development, testing, and production. Each behaves slightly differently, making your entire software development cycle easier. For example, Rails creates a fresh copy of the Test database for each test run.

Rails Philosophy

- **DRY** – "Don't Repeat Yourself" – suggests that writing the same code over and over again is a bad thing.

- **Convention Over Configuration** – means that Rails makes assumptions about what you want to do and how you're going to do it, rather than requiring you to specify every little thing through endless configuration files.

- **REST** is the best pattern for web applications – organizing your application around resources and standard HTTP verbs is the fastest way to go.

Rails Architecture - MVC

- Keeping the logic (or good stuff) away from the user interface. Strict MVC should ensure that you adhere to DRY code and allow for easy maintenance.

MVC architecture

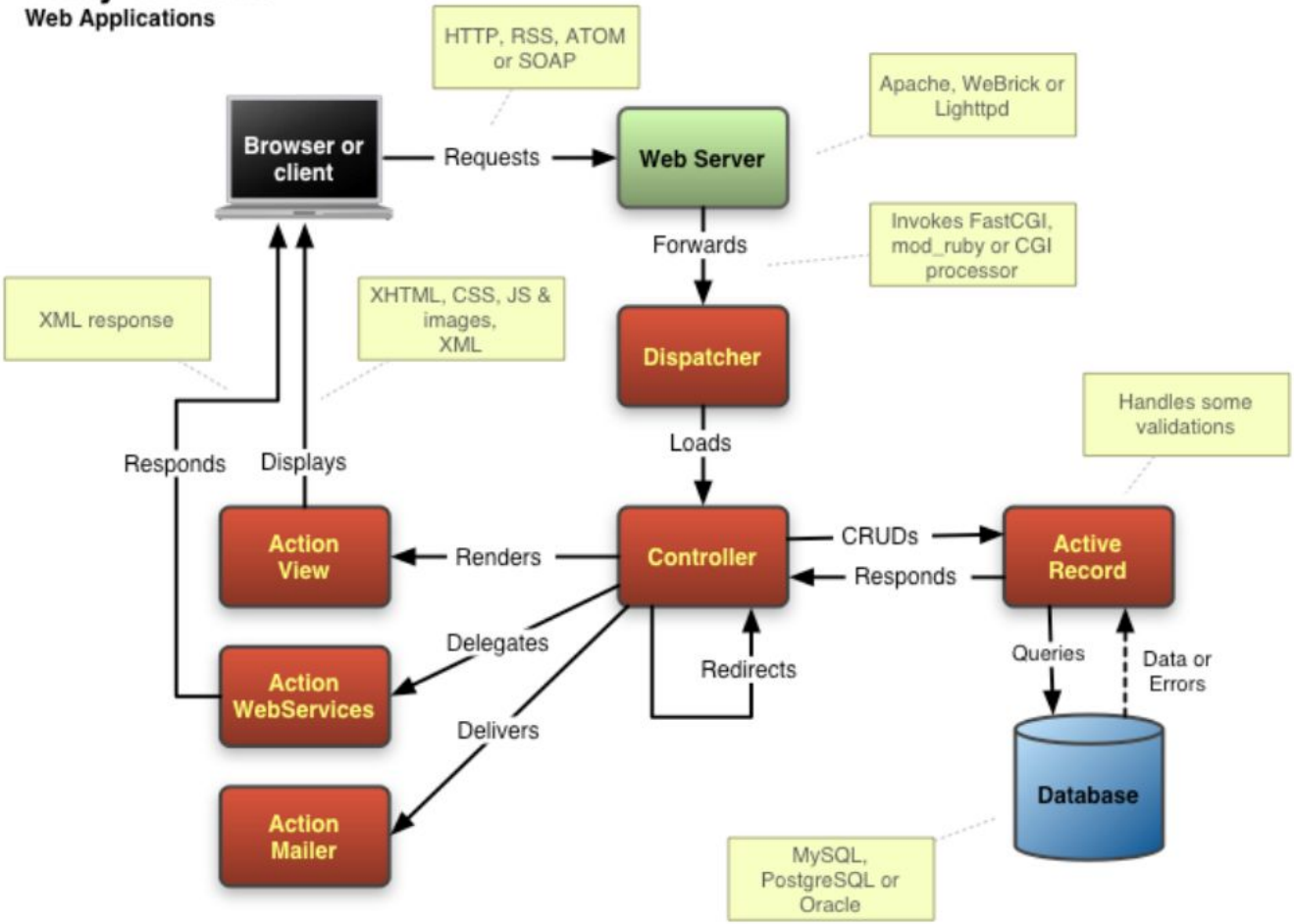
- A **model** represents the information (data) of the application and the rules to manipulate that data. In the case of Rails, models are primarily used for managing the rules of interaction with a corresponding database table. In most cases, one table in your database will correspond to one model in your application. The bulk of your application's business logic will be concentrated in the models.

- **Views** represent the user interface of your application. In Rails, views are often HTML files with embedded Ruby code that perform tasks related solely to the presentation of the data. Views handle the job of providing data to the web browser or other tool that is used to make requests from your application.

- **Controllers** provide the "glue" between models and views. In Rails, controllers are responsible for processing the incoming requests from the web browser, interrogating the models for data, and passing that data on to the views for presentation.

Ruby on Rails

Web Applications



Rails Components

- **Action Mailer**
- Action Mailer is a framework for building e-mail services. You can use Action Mailer to receive and process incoming email and send simple plain text or complex multipart emails based on flexible templates.

- **Active Model**
- Active Model provides a defined interface between the Action Pack gem services and Object Relationship Mapping gems such as Active Record. Active Model allows Rails to utilize other ORM frameworks in place of Active Record if your application needs this.

- **Active Record**
- Active Record is the base for the models in a Rails application. It provides database independence, basic CRUD functionality, advanced finding capabilities, and the ability to relate models to one another, among other services.

Rails Components

- **Active Resource**
- Active Resource provides a framework for managing the connection between business objects and RESTful web services. It implements a way to map web-based resources to local objects with CRUD semantics.

- **Active Support**
- Active Support is an extensive collection of utility classes and standard Ruby library extensions that are used in Rails, both by the core code and by your applications.

- **Railties**
- Railties is the core Rails code that builds new Rails applications and glues the various frameworks and plugins together in any Rails application.