Web Services, REST, and Email

CPS353 Internet Programming

Simon Miner

Gordon College

Last Modified: 11/20/2013

Agenda

- Scripture (Colossians 3) and Prayer
- Check-in
- Web Services and REST
- Email
- Milestone 8
- Faith and Technology Discussion

Check-in

- Syllabus updates
  - Milestone X
- Homework 7
- Milestone 7

Web Services

- Characteristics
  - Application components
  - (Usually) communicate via open protocols (i.e. HTTP)
  - Self-contained and (often) self-describing
  - Can be used to connect existing software
- Two approaches
  - SOAP
    - Heavily used by and between corporations and institutions
    - Very descriptive, flexible and formal, but somewhat cumbersome (XML messages)
  - REST
    - Alternative that makes use of HTTP method verbs
    - Simpler and less formal

SOAP

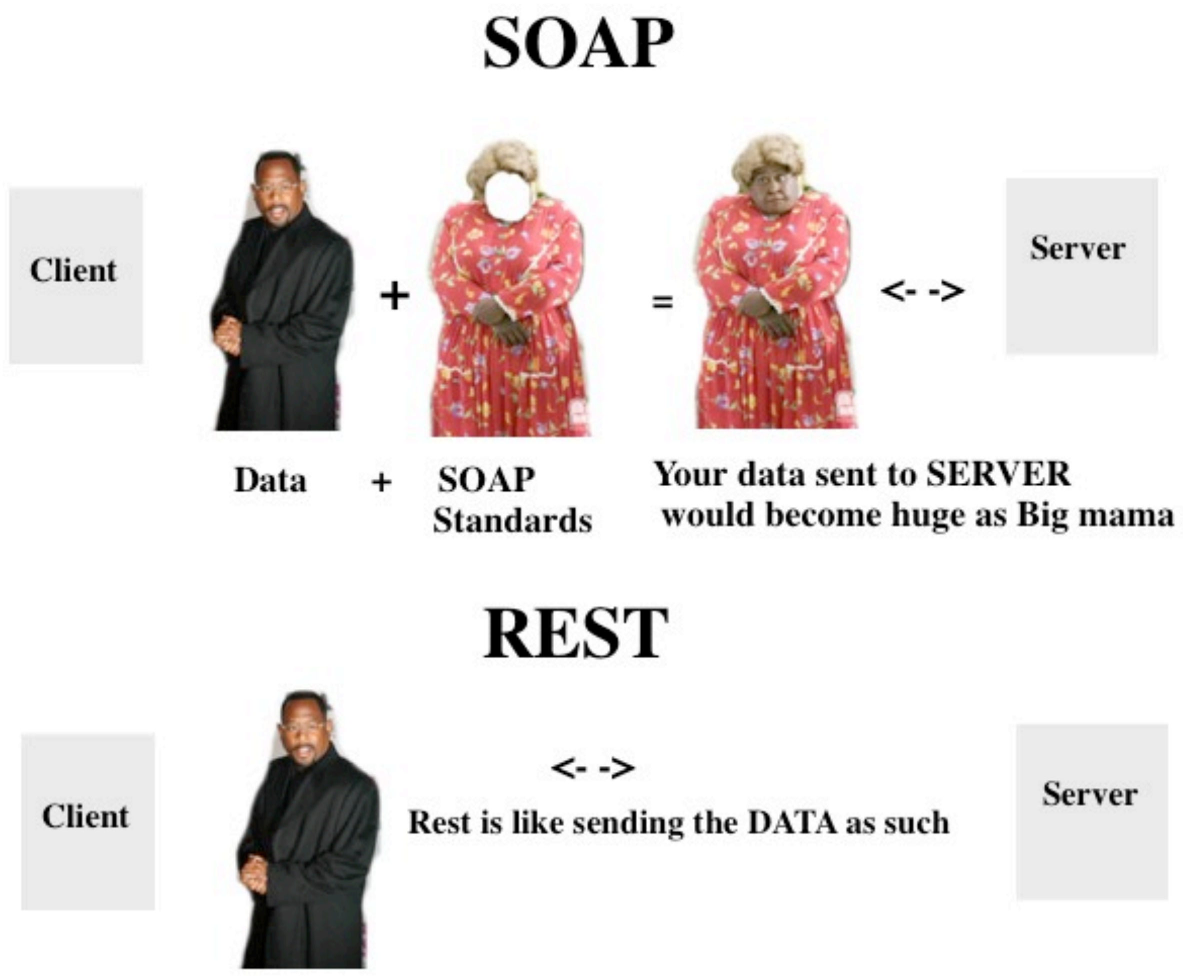
- Simple Object Access Protocol (SOAP)
  - Used to access and manipulate remote objects
  - Platform independent
  - Language independent
  - Uses XML (envelopes, messages, bindings…)
  - Service oriented
- Services are defined in Web Service Description Language (WSDL)
  - Provides location and details about how to access and use web services
    - Feed your program a WSDL and it will get a bunch of classes/functions to interact with the web service(s)
  - In XML
  - Can be registered in a Universal Description, Discovery, and Integration (UDDI) framework so others can find and use it

REST

- Representational State Transfer (REST)
  - Used to access and interact with resources
  - Via HTTP method verbs
    - GET to retrieve a resource from the server
    - POST/PUT to put a resource on the server
    - DELETE to remove a resource from the server
  - Can transmit data in arbitrary formats (text, HTML, binary)
    - JSON – JavaScript Object Notation – the data is a set of (nested) JavaScript objects
  - Resource oriented
- Only requires an HTTP library
  - But client and server must agree on resources and their locations

SOAP/REST Comparison

Consider "Martin Lawrence" as your data



Your data sent to SERVER would become huge as Big mama

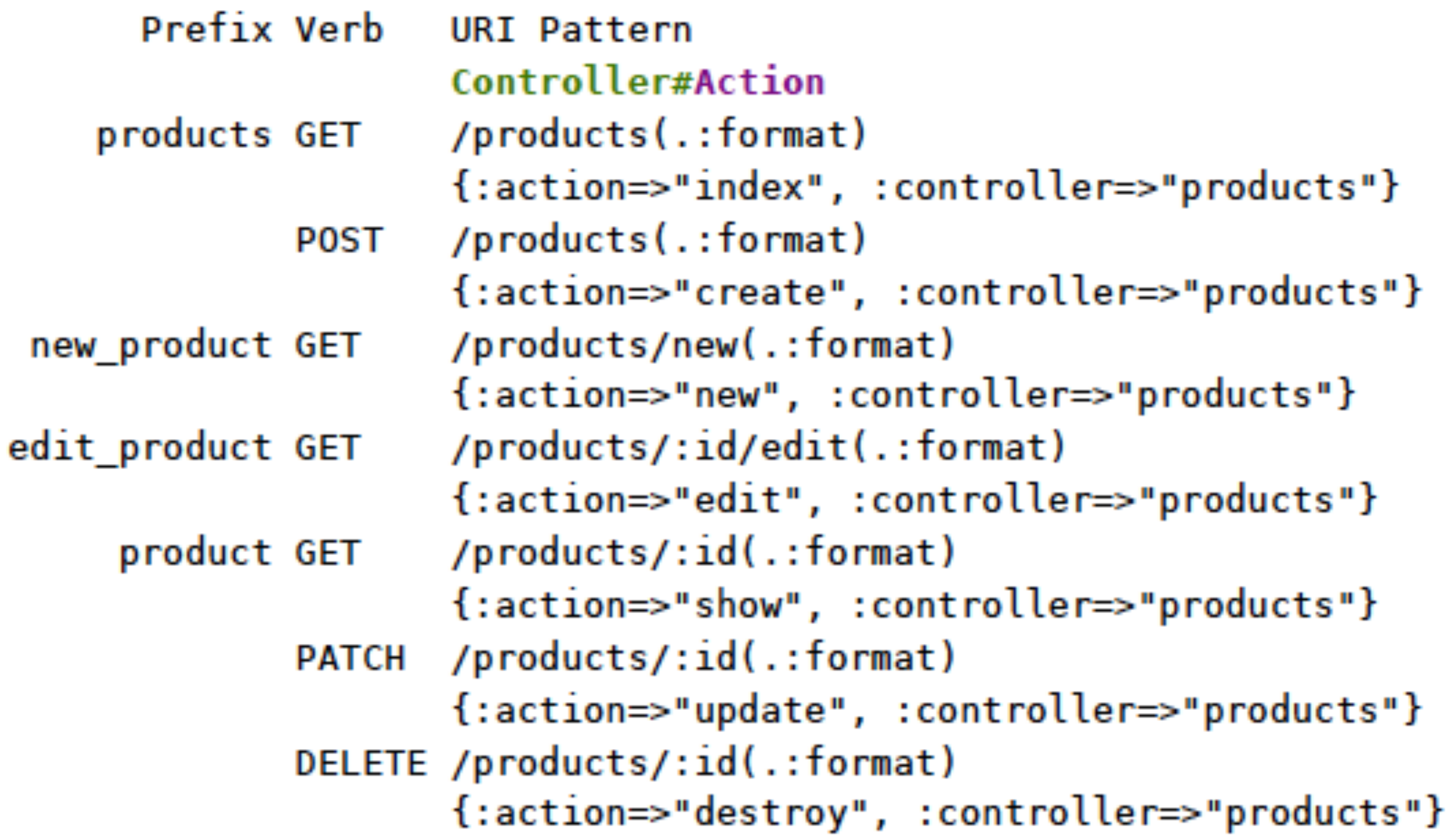
Rest is like sending the DATA as such

REST Benefits

- Common set of verbs (HTTP methods) mean that disparate applications can invoke a predefined set of actions
  - Just need to know the nouns (resources)
  - Makes applications easier to write, read, and maintain
- GET requests are inherently cacheable
  - Excellent for scalability
- Using a plain HTTP transport layer makes REST available on virtually every platform
- Decouples data from its representation

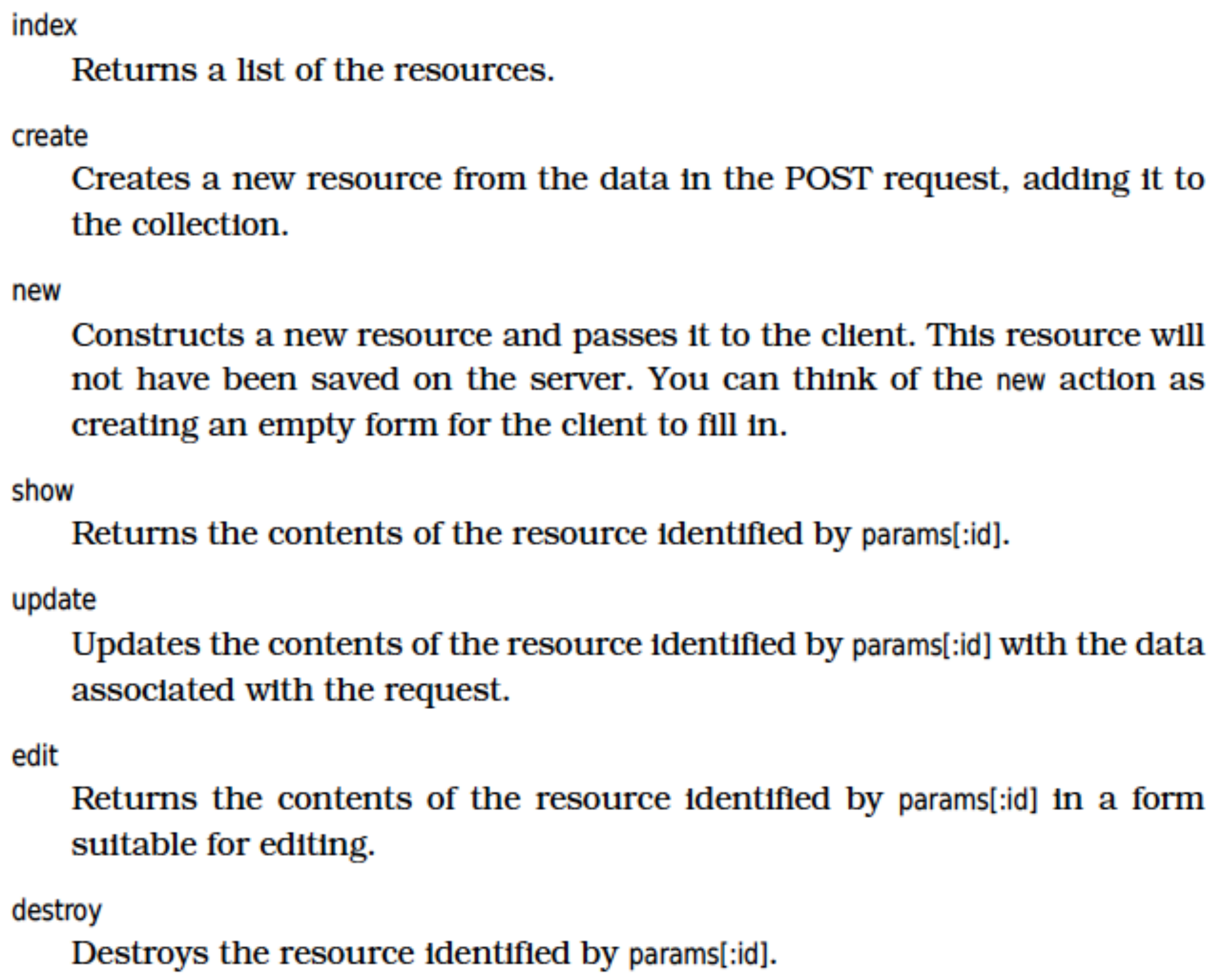
RESTful Resource Routing in Rails

- `rake routes`



```
     Prefix Verb   URI Pattern
                   Controller#Action
   products GET    /products(.:format)
                   {:action=>"index", :controller=>"products"}
            POST   /products(.:format)
                   {:action=>"create", :controller=>"products"}
new_product GET    /products/new(.:format)
                   {:action=>"new", :controller=>"products"}
edit_product GET   /products/:id/edit(.:format)
                   {:action=>"edit", :controller=>"products"}
    product GET    /products/:id(.:format)
                   {:action=>"show", :controller=>"products"}
            PATCH  /products/:id(.:format)
                   {:action=>"update", :controller=>"products"}
            DELETE /products/:id(.:format)
                   {:action=>"destroy", :controller=>"products"}
```

RESTful Controller Actions in Rails



index
: Returns a list of the resources.

create
: Creates a new resource from the data in the POST request, adding it to the collection.

new
: Constructs a new resource and passes it to the client. This resource will not have been saved on the server. You can think of the new action as creating an empty form for the client to fill in.

show
: Returns the contents of the resource identified by params[:id].

update
: Updates the contents of the resource identified by params[:id] with the data associated with the request.

edit
: Returns the contents of the resource identified by params[:id] in a form suitable for editing.

destroy
: Destroys the resource identified by params[:id].

Rails `respond_to` Blocks

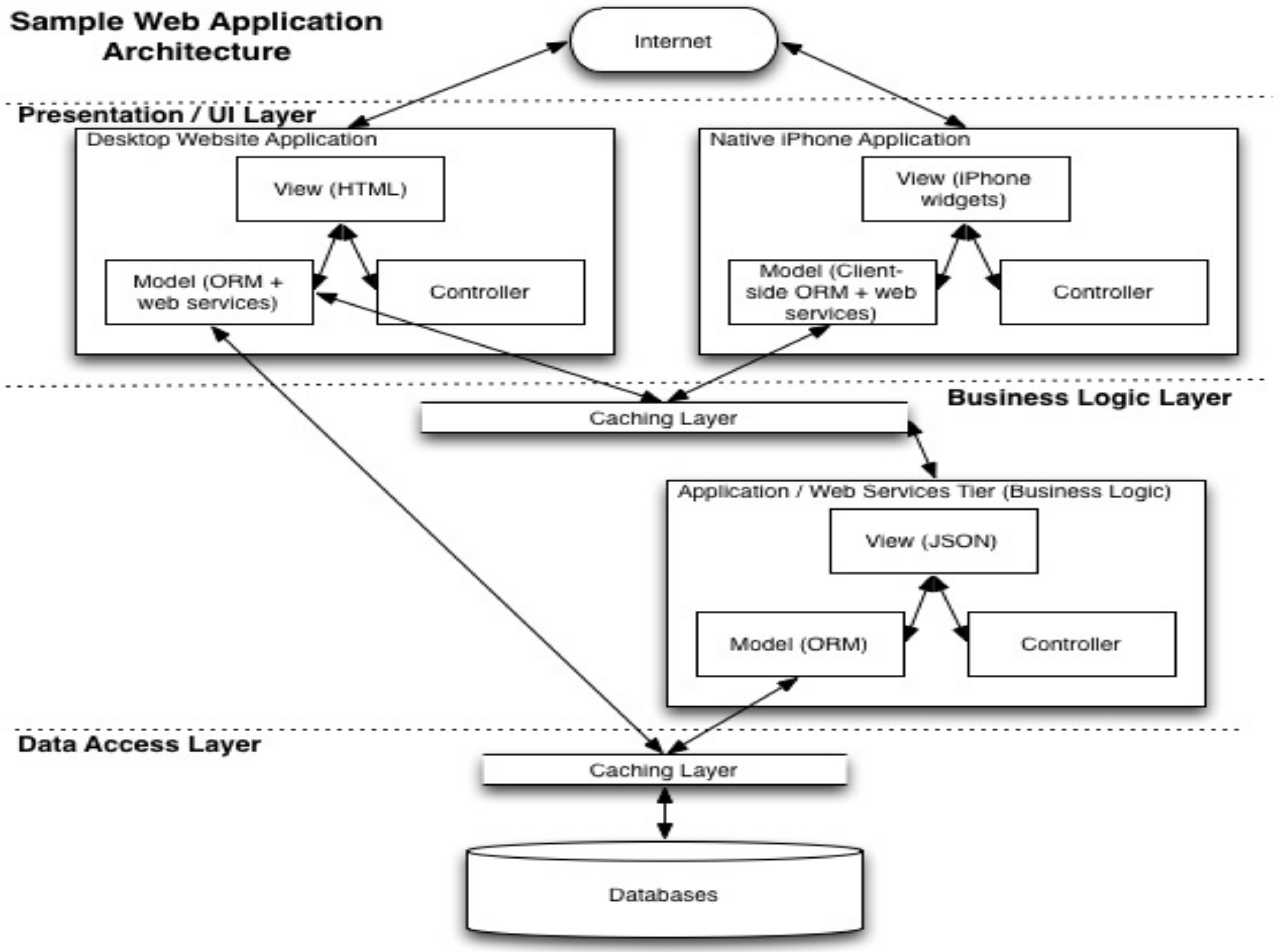
- Allows you to customize the content returned in a response based on its type
  - Each format corresponds to a view template variant

```
def show
  ...
  respond_to do |format|
    format.html { ... } # show.html.erb
    format.json { ... } # show.json.erb
    format.js { ... }   # show.js.erb
    format.xml { ... }  # show.xml.erb
    format.yaml { ... } # show.yaml.erb
  end
end
```

  - URLs can end with format extension or parameter
    - http://example.com:3000/products/1/show.xml
    - http://example.com:3000/products/1/show?format=json

Web Application Architecture

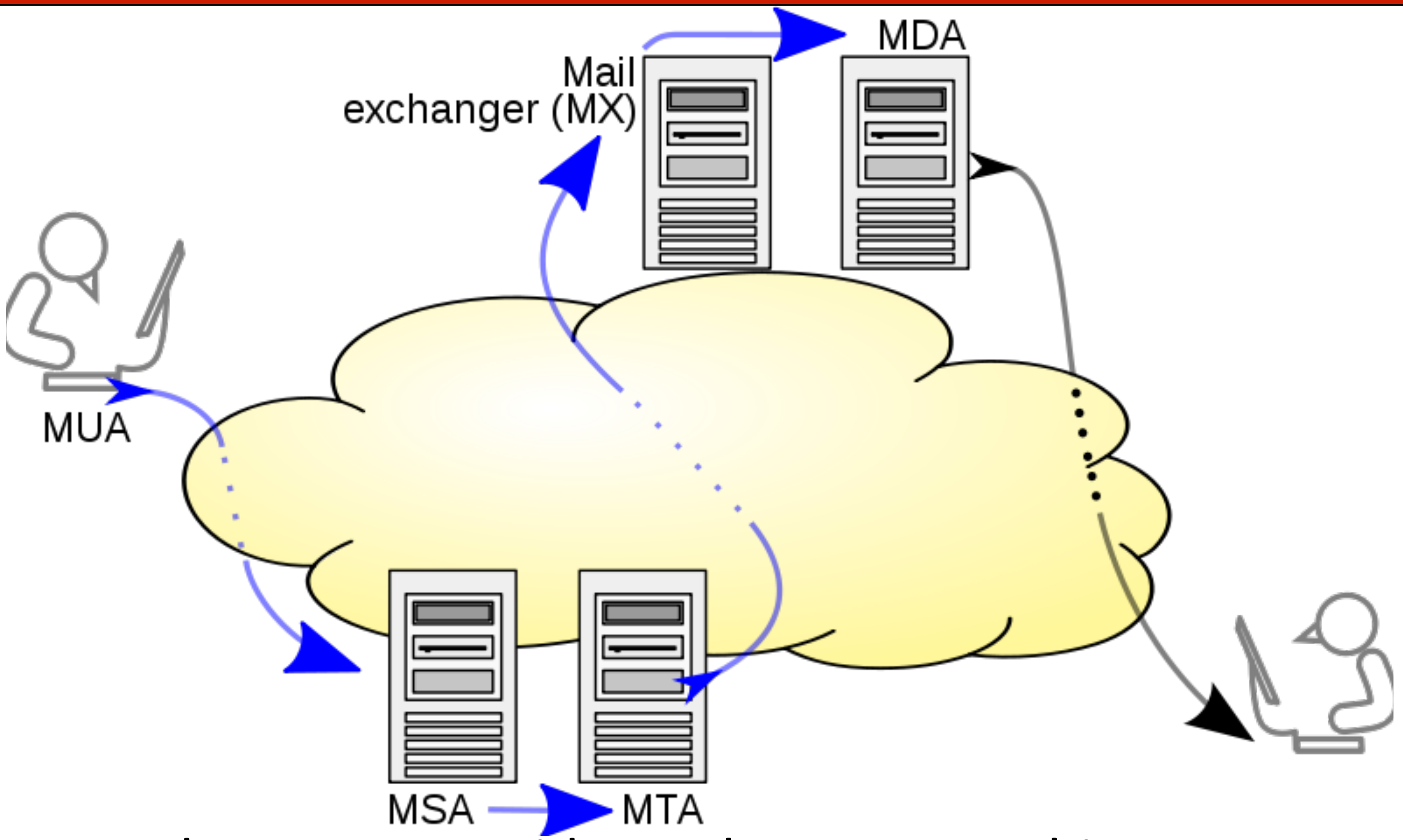
- Two-tier
  - Presentation / UI layer
  - Data access layer
- Three-tier
  - Presentation / UI layer
  - Application / business logic layer
    - Web services reside here
  - Data access layer
- In a three-tier architecture, the presentation layer invokes web services to manipulate data
  - Less expensive than going directly to the data layer (fewer database connections, cached resources, etc.)
  - Presentation layer can also invoke third-party web services (i.e. Facebook, Google, Amazon)


Sample Web Application Architecture
Internet
Presentation / UI Layer
Desktop Website Application
View (HTML)
Model (ORM + web services)
Controller
Native iPhone Application
View (iPhone widgets)
Model (Client-side ORM + web services)
Controller
Business Logic Layer
Caching Layer
Application / Web Services Tier (Business Logic)
View (JSON)
Model (ORM)
Controller
Data Access Layer
Caching Layer
Databases

Email

- Simple Mail Transfer Protocol (SMTP)
  - Protocol for sending email over the Internet
- Mail Transport Process
  - Email submitted by a mail client (MUA: mail user agent) to a mail server (MSA: mail submission agent)
  - MSA delivers the mail to its mail transfer agent (MTA)
  - MTA uses DNS to look up the mail exchange (MX) record for the recipient's domain and target host and send the email message there
  - MTA sends the message to the MX target
  - MX target hands off message to a mail delivery agent (MDA) which stores the message for later use
  - The message recipient (another MUA) retrieves the message from the MDA via Internet Message Access Protocol (IMAP) or Post Office Protocol (POP)

Mail Transport Model



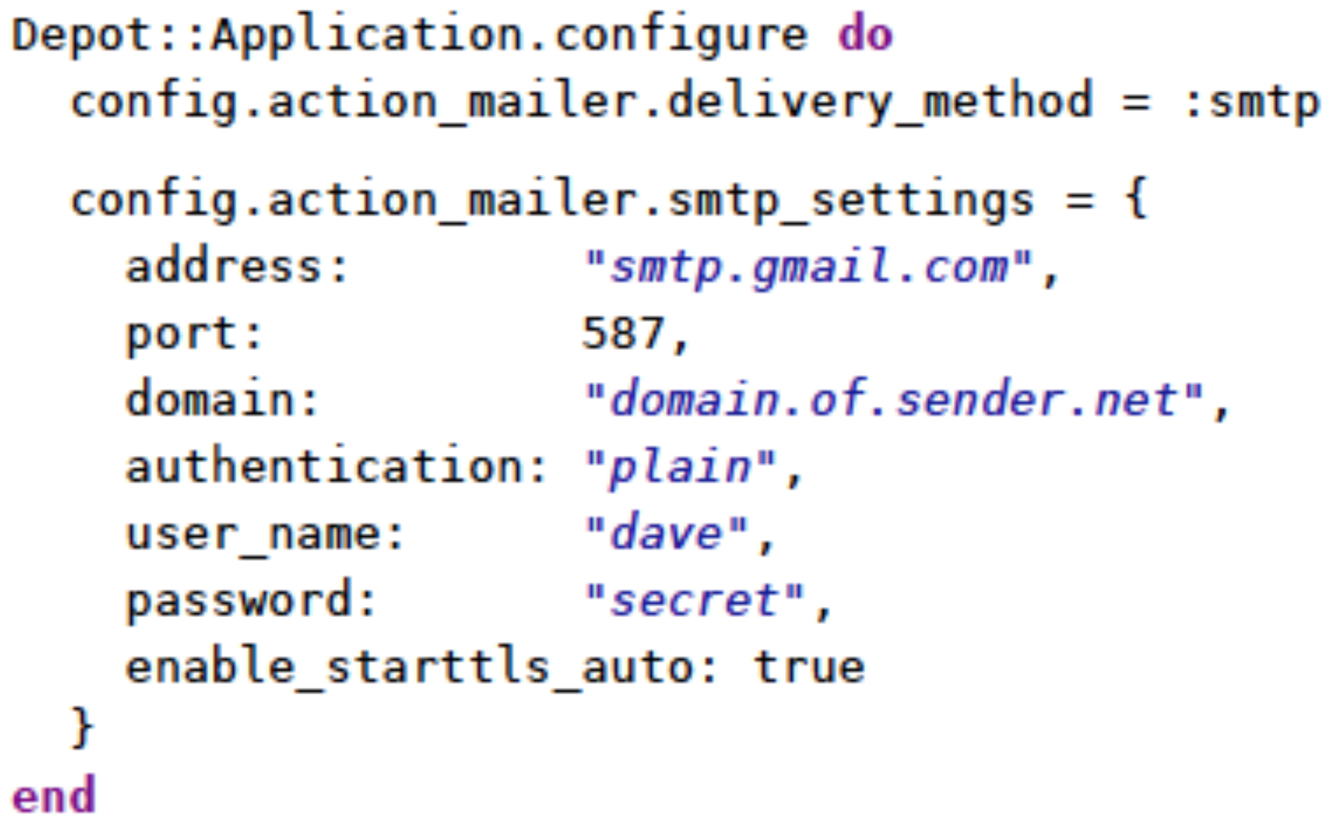
- MSA and MTA may reside on the same machine or on different ones
- MX and MDA may also reside together or on separate hosts

Sending Email with Rails

- Rails facilitates sending email messages using many of the same building blocks used to create web applications
  - Environment-specific configuration
  - Controllers and views
- Action Mailer

Rails Email Configuration

- In config/environment.rb
  - Or in one of the environment-specific configurations in config/environment/

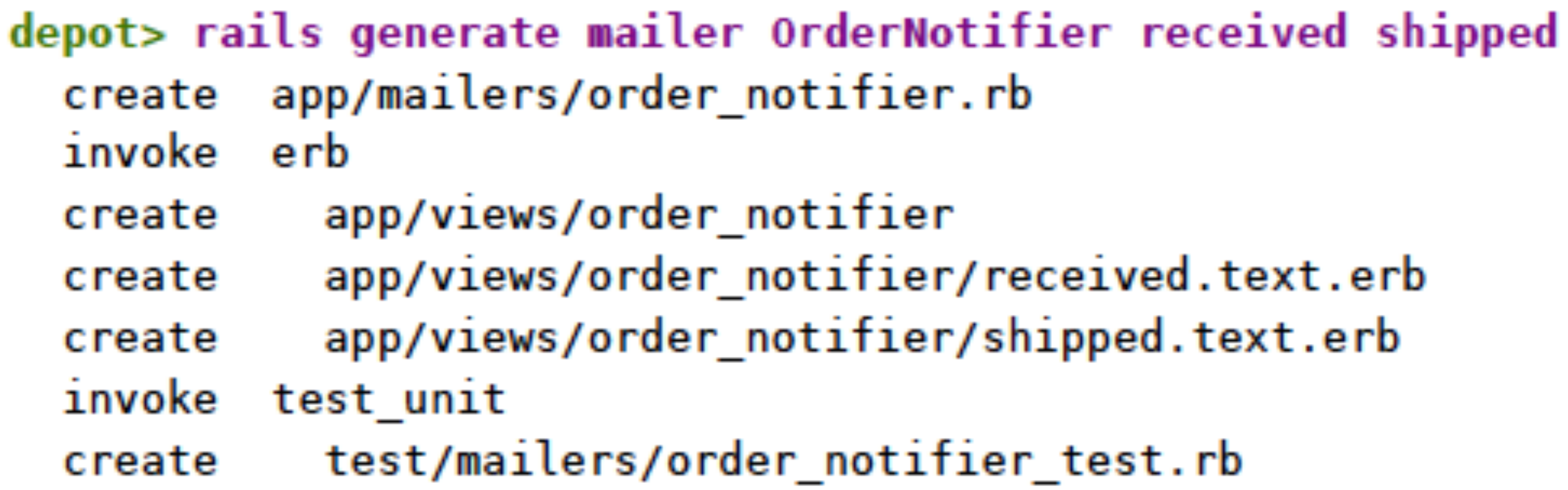


```ruby
Depot::Application.configure do
  config.action_mailer.delivery_method = :smtp

  config.action_mailer.smtp_settings = {
    address:              "smtp.gmail.com",
    port:                 587,
    domain:               "domain.of.sender.net",
    authentication: "plain",
    user_name:            "dave",
    password:             "secret",
    enable_starttls_auto: true
  }
end
```

Rails Mailer Generation

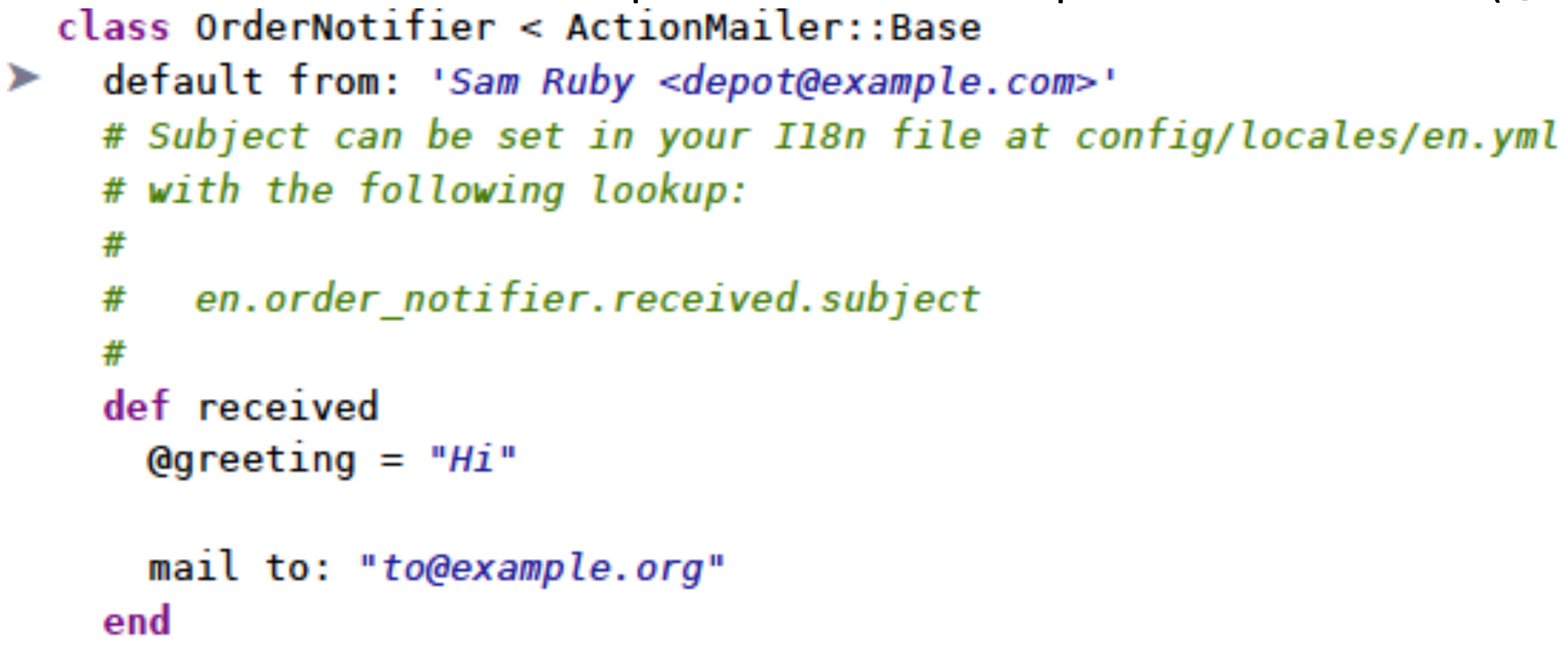
- The rails command has a generator for mailers
  - New mailer class
    - Integrates into existing controller(s)
  - Creates view templates and tests



```
depot> rails generate mailer OrderNotifier received shipped
  create  app/mailers/order_notifier.rb
  invoke  erb
  create    app/views/order_notifier
  create    app/views/order_notifier/received.text.erb
  create    app/views/order_notifier/shipped.text.erb
  invoke  test_unit
  create    test/mailers/order_notifier_test.rb
```

Mailer Class

- Set up like a controller
  - One method per action
  - Calls mail instead of render
    - Accepts values for :from, :cc, :subject, etc.
    - Default values can be set via :defaults at the top of the class declaration
    - Action methods can take parameters to incorporate custom data (@order)

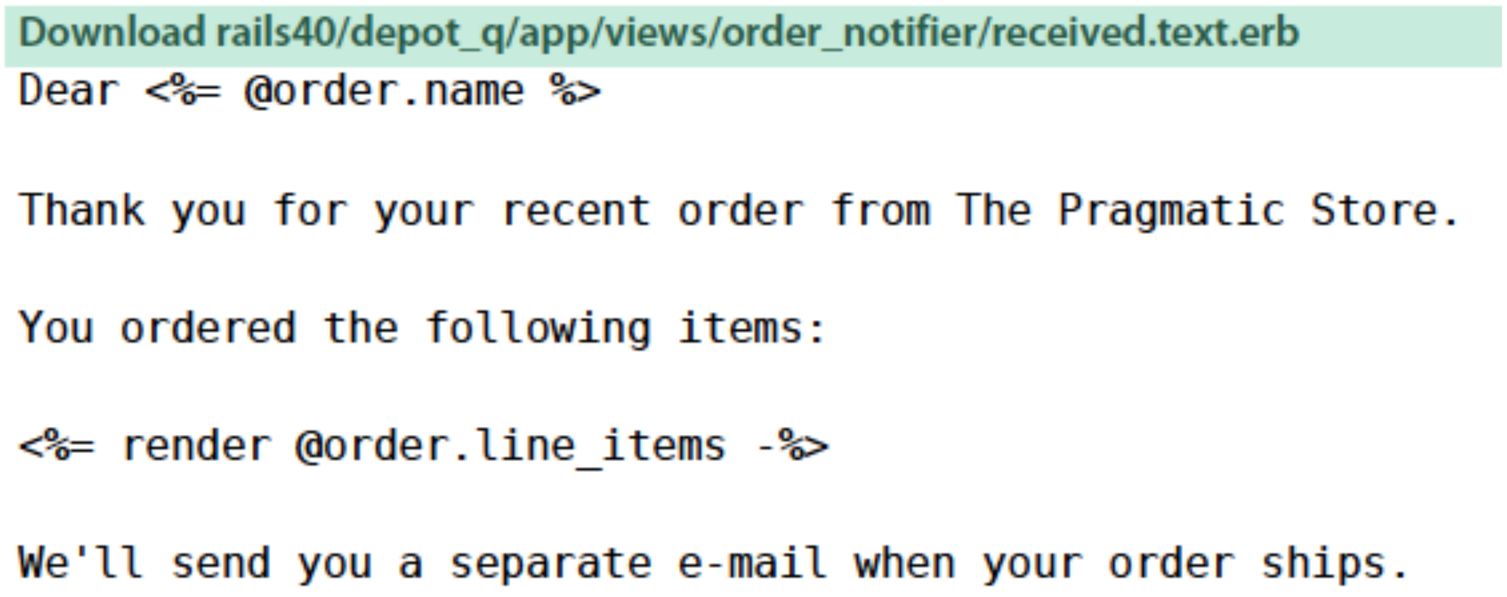


```
class OrderNotifier < ActionMailer::Base
  default from: 'Sam Ruby <depot@example.com>'
  # Subject can be set in your I18n file at config/locales/en.yml
  # with the following lookup:
  #
  #   en.order_notifier.received.subject
  #
  def received
    @greeting = "Hi"

    mail to: "to@example.org"
  end
```

Mailer Views

- View contains the text of the email message
  - Work just like normal views (with instance variables, partials, etc.)
  - Mailers can send email in different formats (plaintext, HTML)
    - Separate view for each format with appropriate extension



Download rails40/depot_q/app/views/order_notifier/received.text.erb

```
Dear <%= @order.name %>

Thank you for your recent order from The Pragmatic Store.

You ordered the following items:

<%= render @order.line_items -%>

We'll send you a separate e-mail when your order ships.
```

Controller Integration

- Mailer’s deliver method invoked in the appropriate controller

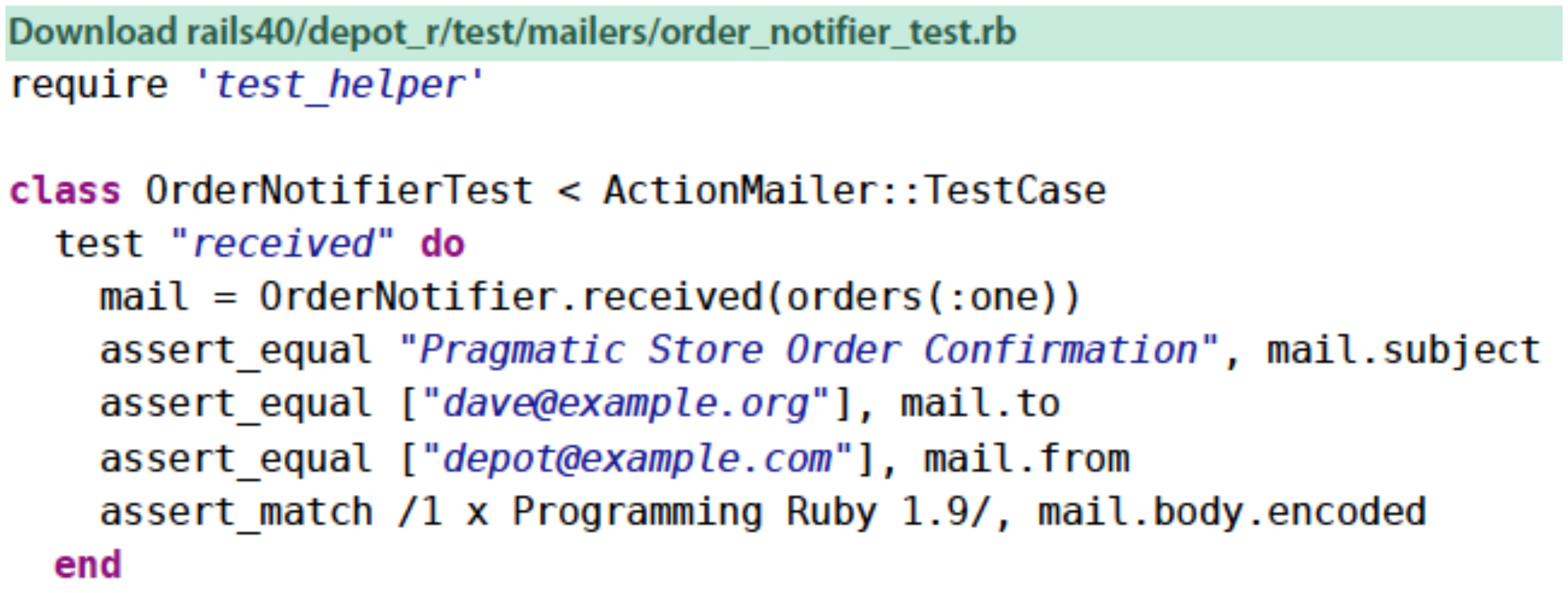
Download rails40/depot_r/app/controllers/orders_controller.rb

```
def create
  @order = Order.new(order_params)
  @order.add_line_items_from_cart(@cart)

  respond_to do |format|
    if @order.save
      Cart.destroy(session[:cart_id])
      session[:cart_id] = nil
➤     OrderNotifier.received(@order).deliver
      format.html { redirect_to store_url, notice:
        'Thank you for your order.' }
      format.json { render action: 'show', status: :created,
        location: @order }
    else
      format.html { render action: 'new' }
      format.json { render json: @order.errors,
        status: :unprocessable_entity }
    end
  end
end
```

Testing Mailers

- Mailer test class can be used to verify the expected contents of an email message.



Download rails40/depot_r/test/mailers/order_notifier_test.rb

```ruby
require 'test_helper'

class OrderNotifierTest < ActionMailer::TestCase
  test "received" do
    mail = OrderNotifier.received(orders(:one))
    assert_equal "Pragmatic Store Order Confirmation", mail.subject
    assert_equal ["dave@example.org"], mail.to
    assert_equal ["depot@example.com"], mail.from
    assert_match /1 x Programming Ruby 1.9/, mail.body.encoded
  end
```

Milestone 8

Question

Is technology morally neutral?

Definition

- Technology is “the human activity of using tools to transform God’s creation for practical purposes.” (*From the Garden to the City* by John Dyer)
- Four perspectives
  - Technology as hardware – natural resources, raw materials
  - Technology as manufacturing – processes and procedures to build and use things
  - Technology as methodology – skill and know-how to use tools
  - Technology as social usage – cultural customs and rules surrounding how we use tools

Stories of Technology

- We use technology to transform our world.
- Technology transforms us as we use it.
- Technology shapes our souls as we interact with it.

Embedded Values

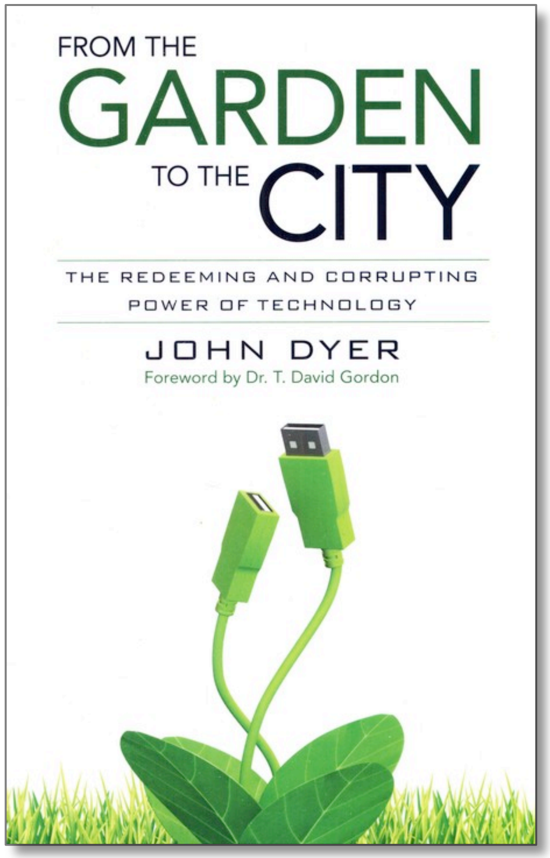
- Technology reflects the values of those who build and use it.
- Example: the Internet
  - What are its values?
  - Where do they come from?
  - How do they square with Christian values?

How do we Respond?

- Work to redeem technology?
  - Somewhere between blind acceptance and full withdrawal
  - In the world, but not of the world
- Some suggested responses
  - Be mindful of how your tools shape you.
    - Evaluate and experiment
  - Moderate your use of technology and its impact on you.
    - Focal practices.
  - Use tools for good
  - Build good tools
  - Hope for Christ's restoration of the world – including technology

Resources

- http://toolsforworship.net
  - My blog exploring the connections between Christian worship and technology
- *From the Garden to the City* by John Dyer


FROM THE
GARDEN
TO THE
CITY
THE REDEEMING AND CORRUPTING POWER OF TECHNOLOGY
JOHN DYER
Foreword by Dr. T. David Gordon