MVC in Node: Express

SoC

MVC

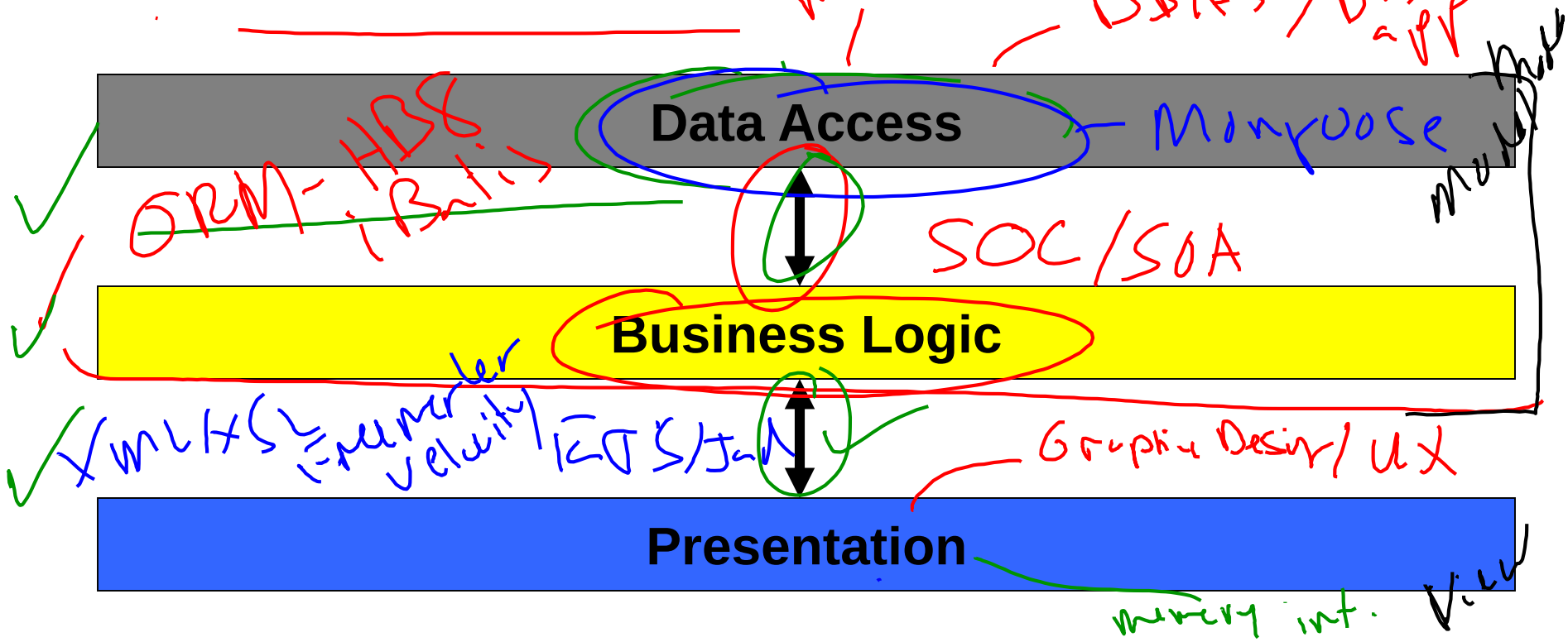
MVC in Node using Express

Routes

View templates

Separation of Concerns (SoC)

Web applications have 3 important aspects, or *concerns*



It is important to have *Separation of Concerns* (SoC):

- Allows custom expertise and frameworks to be applied to each concern
- Allows for greater run-time deployment flexibility & optimizations

Separation of Concerns

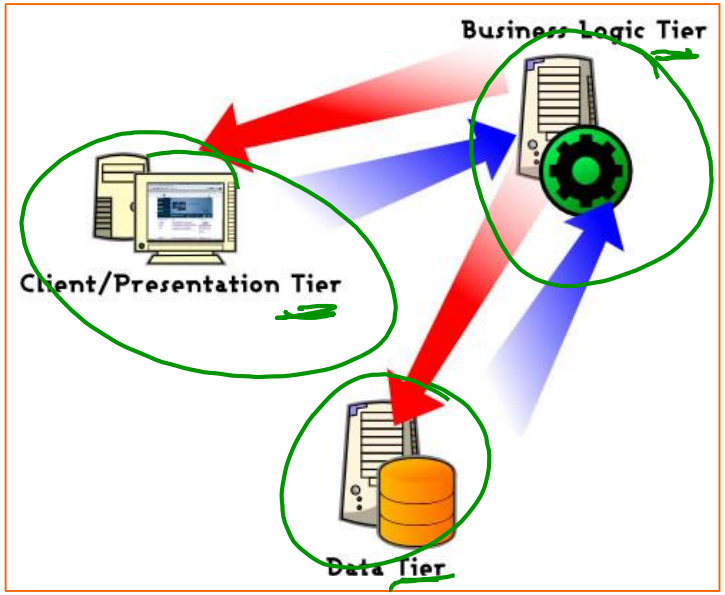
Allows for greater run-time (deployment) flexibility

- Businesses had (still have) a wealth of proprietary (and poorly understood) business rules
- Businesses had (still have) a wealth of proprietary (and poorly understood) datasources
- Often these two layers were quite incestuous
  - Business rules coded as stored procedures, database triggers, long SQL queries, or (Blech!) COBOL

SoC says to separate these layers & deploy independently

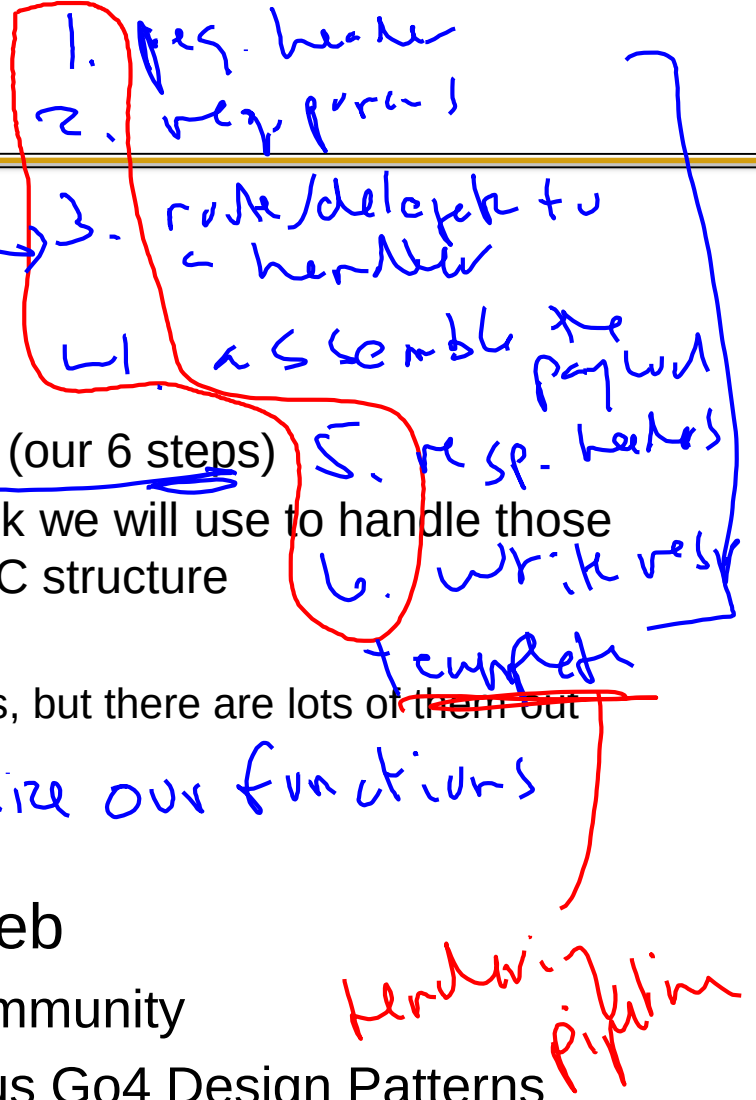
MVC – Model-View-Controller

- *MVC != SOC*, and *MVC != n-tier*
- the **View** is the Presentation
- the **Model** comes from Datasources,
- the **Controller** is not the *Business Logic*
  - that also is in the **Model**
  - Controller **routes** requests



http://www.adobe.com/devnet/coldfusion/articles/ntier.html

Model-View-Controller (MVC)



THE design pattern of the web

- What is a *design pattern* again?
  - A reusable solution in context
  - The *context* here is a web application (our 6 steps)
  - The reusable solution is the framework we will use to handle those common 6 steps according to the MVC structure
    - Yes MVC is a Structural pattern
    - The framework we will use is Express, but there are lots of them out there in lots of languages

Go4 - 00 patterns

Organize our functions

History: MVC did not start with the web

- Started in the 70s in the Smalltalk community
- Yes, the same community that gave us Go4 Design Patterns
- See the “Thing-Model-View_Editor” reading

rendering pipeline

The servlet community gave us a web version

Was called "Model 2" MVC:

- Controller servlet introduced to route requests
- JSP or Templates used for View only
- Java beans marshall data ("world model") between the Model & the View
- The Model include the Business Logic layer as well.

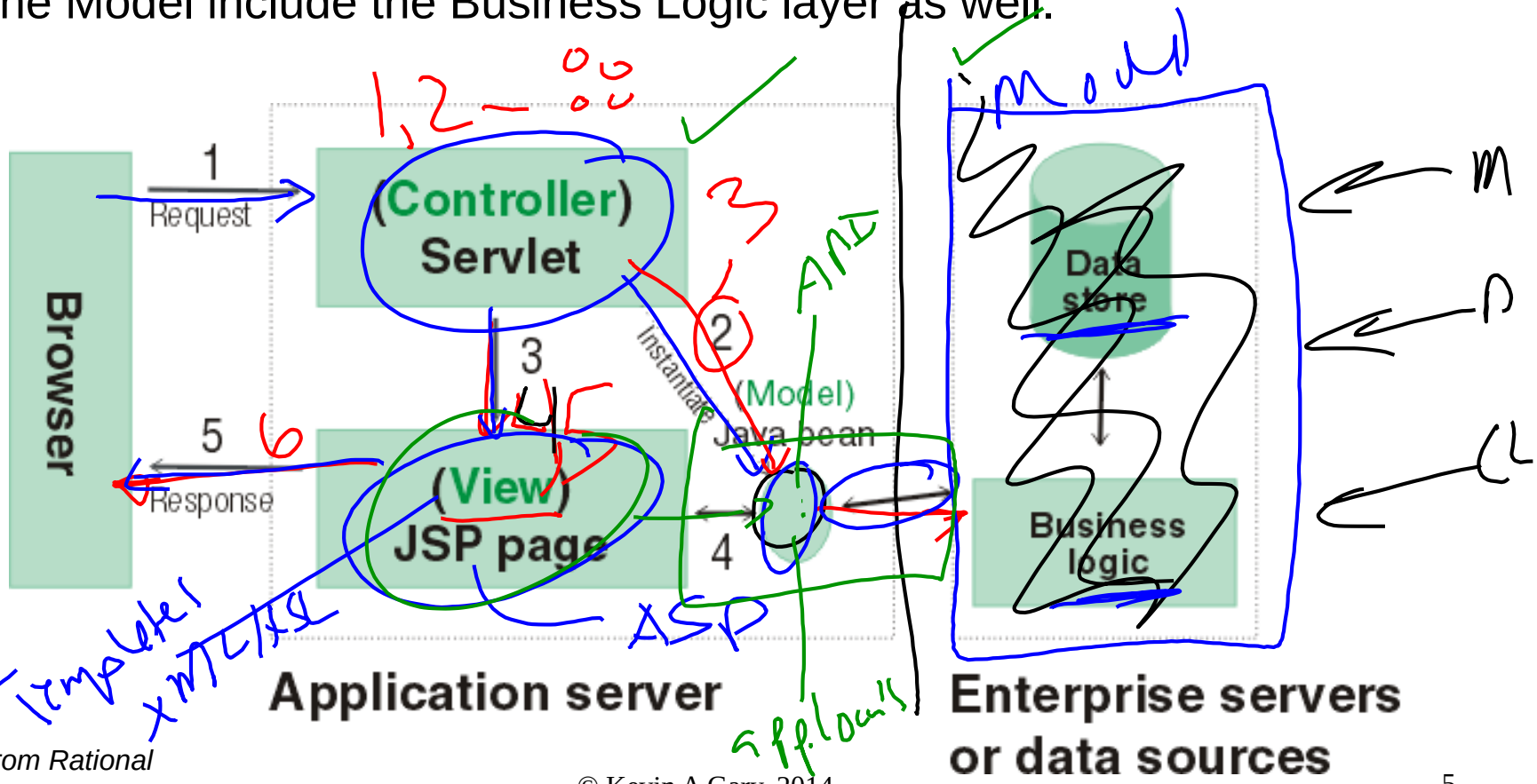


Image from Rational

Implementing MVC

Steps:

1. HTTP request made to *Controller*
2. Controller (*routes* or *delegates*) to the *Model*
3. The Model executes the *business logic* on the *world model* (non-volatile state, typically a datasource)
4. The Controller identifies the appropriate *View template*
5. The View (*read-only)* accesses the Model to get the dynamic output
6. The View produces the final rendered response to the client browser.

Compare these 6 steps to the 6 steps of the Template + Strategy Pattern we discussed in the HTTP notes

A Super-Simple Express Example

Express is a popular MVC framework in Node

- npm install express
- require('express')
- Then you can do

```
var app = express();
```

```
var express = require('express');
var app = express();
app.listen(80);
app.get('/', function(req, res){
  res.send('Hello from Express');
});
```

Compare the Express way to the old way →

```
var http = require('http');
http.createServer(function (req, res) {
   res.writeHead(200,
    {'Content-Type': 'text/plain'});
   res.end('Hello World\n');
    }).listen(8081);
```

- Note it's not all that different from a basic HTTP example in structure, except now we have this express object "app"
- Express object supports an HTTP verb method (get)
- There is a small amount of *delegation*, or *routing* going on here

A little more involved example

```
// See express_send.js
var express = require('express');
var url = require('url');
var app = express();
app.listen(80);
app.get('/', function (req, res) {
  var response = '<html><head><title>Simple Send</title></head>' +
                 '<body><h1>Hello from Express</h1></body></html>';
  res.status(200);
  res.set({
    'Content-Type': 'text/html',
    'Content-Length': response.length
  });
  res.send(response);
  console.log('Response Finished? ' + res.finished);
  console.log('\nHeaders Sent: ');
  console.log(res.headerSent);
});
app.get('/error', function (req, res) {
  res.status(400);
  res.send("This is a bad request.");
});
```

Routes in Express

Action-oriented mapping of URLs/Verb/Parameter/Header info

- 2 general framework approaches to server-side web dev: *action* (or *page)* oriented frameworks versus *component-oriented* frameworks
- We don't really need component-oriented on the server-side anymore with the advent of rich GUI libraries on the client-side
- But we do need to know how to dispatch actions to the proper delegation logic. That is, we still need a *controller*

endpoint route

In Express (and we will see in Angular), these are called ***routes***

General form: *app.<verb>(<path>, [<middleware…>], <cback>)*

- *Middleware* we'll talk about later
- So for now we have examples like (*see express_simple_routes.js*):

```
app.get("/foo", function(req, res) { res.send("bar"); });
app.post("/pink", function(req, res) { res.send("elephant"); });
app.all("/user/*", function(req,res){ res.send("Any verb for users");});
```

More on Routes

Routes can use params from query strings or POST payloads

- Query string processing we saw with HTTP

Can also use regex or defined params

- Defined parameters provide a simple *interceptor pattern* approach – each defined parameter found may invoke a callback that is executed before the route handler (like a servlet filter).
- `app.get('/usr/:id',function(req,res) { res.send("User:"+req.param("id"));});`
- `app.param('id', function(req,res,next,val) { console.log('id='+val);next();});`

middleware

- Filter is invoked before app.get on /usr
- next() must be invoked to keep *Chain of Responsibility* pattern going
- See *express_routes.js*

View layer - templates

View templates have also been around for a long time

- Benefit is the straightforward simplicity
- Disadvantage is potential coupling, hardwiring of a layout

Node can do templates too – in fact lots of them

- These are again straightforward and simple
- Two popular variants – jade and ejs
  - Ejs is simpler, uses that embedding style
  - Jade is a bit more popular as it has more features

In Java template engines are fronted by a servlet. Node?

- In Express, the app *engine* is responsible for rendering.
- You set the engine and *render*, not *send*, a response
  - `app.engine(<file extension>, <module>.<callback>);`
    `app.engine('jade', jade.__express);`
    `app.engine('html', ejs.renderFile);`
    `app.render(<file>, <callback>) or res.render(<file>)`
  - (*see express_templates.js*)

Express request and response objects

The Request object wraps an HTTP request object with a convenience API

- You can get a number of properties associated with HTTP request, like IP, Host, Method, etc.
- Can also get any header with req.get(<header>) or req.headers
  - See *express_request.js*

The Response object wraps an HTTP response

- you can set headers res.set(header, val)
- type(<mime>) sets Content-type, status(<code>) sets status (default 200)
- Transmitting response is as simple as res.send(<String>)
- You can send JSON back using res.json(<code>,<json>) or via JSONP as res.jsonp provided you 1st app.set('jsonp callback name', <name>)
  - JSONP allows client to send a callback name to server to repeat back to client on response so it is called on next page load – client-side callback
  - *See express_send_json.js*
- Can also send files using res.sendfile(<path>) or res.download(<path>)
  - Former detects MIME type via file extension
  - Latter uses an attachment (sets the Content-Disposition header)
- See *express_send_file.js* and *express_send_download.js*

Summary

What parts of MVC have we covered?

1. Incoming request – check!
2. Delegate into the business layer – yes on *delegates* as we have *routes*
3. What about the “world model state”?
   - This is whatever persistent store you decide to use as a datasource, such as a traditional DB (MySQL), a newfangled DB (MongoDB), or just a plain ‘ol flat file system (fs module)
4. We factor presentation into a rendering layer based on templates
   - However we do not “pass control” as with servlets, instead rendering here just becomes more work for the event queue
5. The views bind values to symbols for use in the template
   - It is a pretty low-tech but simple way to do it!
6. We do send the response back (or render the response)

MVC is the predominant web app design pattern, even for client-centric apps. You should look for it in any web dev framework on any platform you choose!