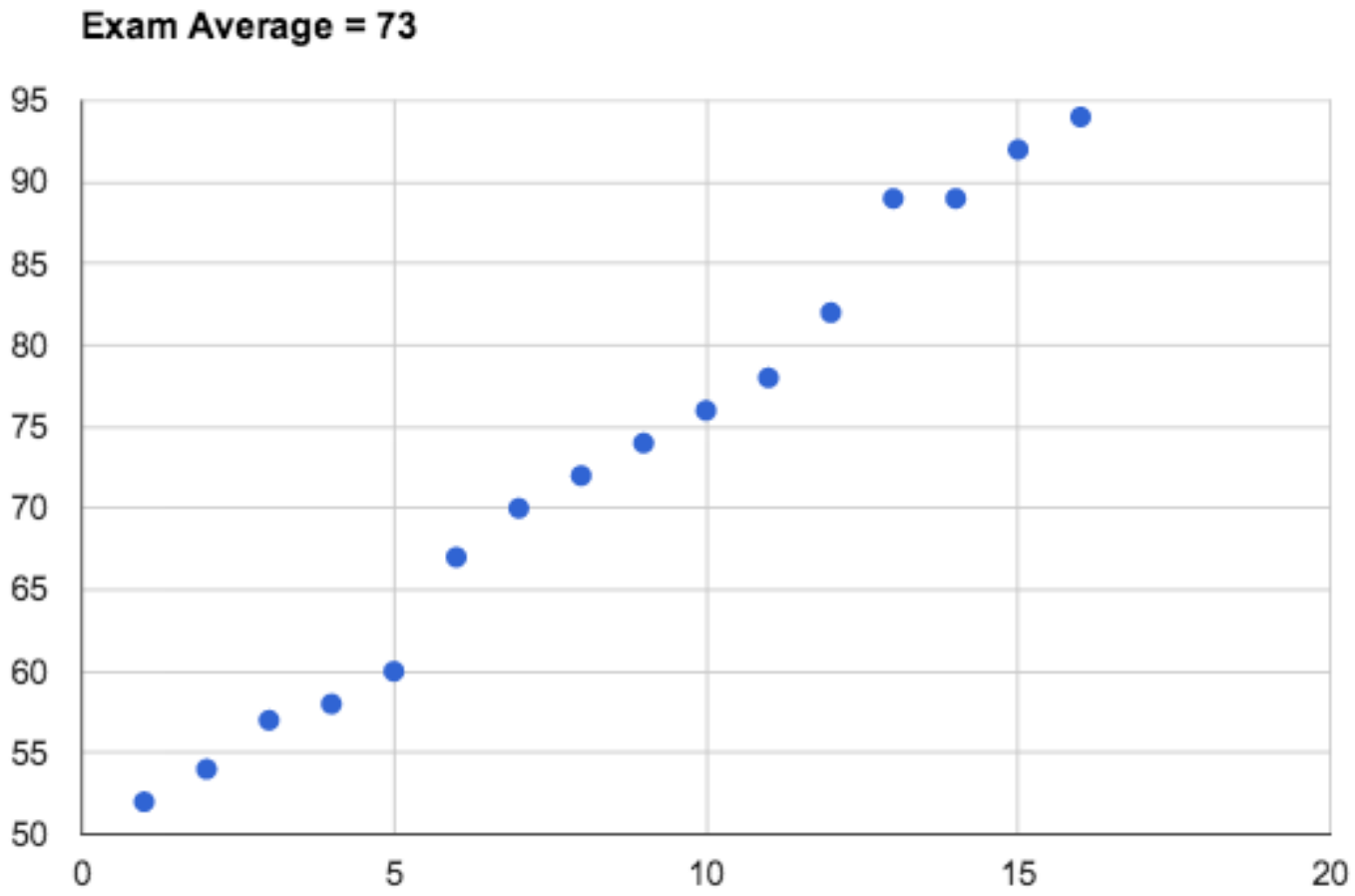
EJS Templates

Lecture 9

Serving dynamic content with server-side JavaScript

Exam 1



Dynamic Content

Inherent in any server-side platform is the ability to customize the appearance of a page

Later in the semester, we'll discuss many server-side languages - such as ASP.NET, PHP, and JSP.

But since we know JavaScript - we'll stick with it…

We'll look at Jade later in the semester…

EJS

This will be our first foray into the world of MVC

- Model
- View
- Controller

- Our Node.js code is the *controller*, an EJS file will be the *view*.
- We will create a JavaScript object to serve as the model.

EJS

EJS files do not get directly served to the browser

Instead, they are **rendered**, server-side.

- They are mostly HTML, but include special instructions in them to dynamically build parts of the HTML
- The rendering step accepts a JavaScript object (model) as input - which can be used by the EJS code.
- The EJS code is… JavaScript

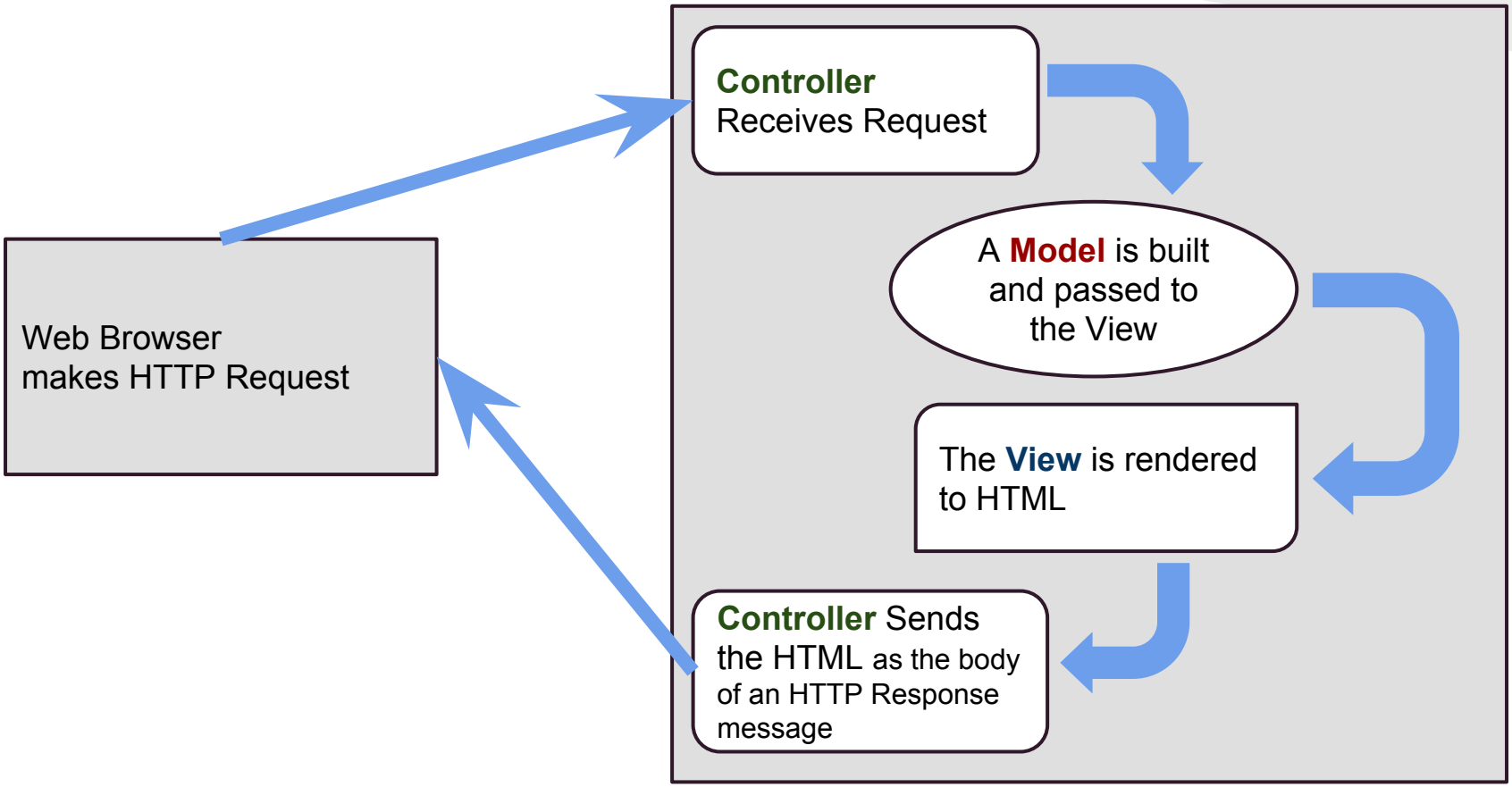
Not your father's JavaScript...

- Be perfectly clear - none of the JavaScript we put in an EJS file ever goes to the browser
- We are not talking about JavaScript that you see on the web!
- We are using JavaScript to **generate** HTML, on the server.

```
ejs = require('ejs');
var model = {"title": "my first ejs page", "message" : "this is a message!"};
ejs.renderFile("testing.ejs",model,
     function(err, result) {
          if (!err) {
               // the result is a string of HTML,
               // the rendering result
               res.end(result);
          }
          else {
               res.end("An error occurred");
          }
     });
```

```
<html>
     <head>
          <title> <% title %> </title>
     </head>
     <body>
          <p> <% message %> </p>
     </body>
</html>
```

The MVC Model



EJS Code

All of your EJS code is simply JavaScript

It must be enclosed in `<% %>` delimiters

You can actually change the delimiters if you want…

To print a variable to HTML, just add an equal sign and type the variable

```
<%= title %>
```

EJS Conditional Statements

```
<% if (users.length > 0) { %>
    <p> There are <%= users.length %> users</p>
<% }
    else { %>
    <p> There are no users! </p>
<% } %>
```

EJS Loops

```
<% users.forEach(function (user) { %>
    <li><%= user.name %></li>
<% }); %>
```

Loop through an **array**

```
<% for ( property in object ) { %>
    <li>
        <%= property %> -> <%= object[property] %>
    </li>
<% } %>
```

Loop through an **object**

EJS Includes

We can add common parts to all of our pages (like headers and footers) by including external EJS files

```
<% include header %>
Hello, <%= title %>!
<% include footer %>
```

header.ejs and footer.ejs need to be in the same directory as the ejs calling out to them.

Path notations are also acceptable (**include ../parts/header**)

Showing the Post Data

Lets build an example where the data sent in the HTML Form is redisplayed to the screen

We'll just send the entire (parsed) post object to the view and loop through it in EJS

EJS Syntax

```
<p>Data from Node</p>
<ul>
   <% for ( q in post ) { %>
   <li>
      <%= q %> -> <%= post[q] %>
   </li>
   <% } %>
</ul>
```

Example: Guessing Game

Our first multi-page example:

- **/start.html**: The server will assign a random number to the "user" between 1 and 10. The page will have a text box for the user to enter a guess
- **/check.html:** the form will submit to check.html, and the server will decide if the guess is too low, too high, or correct.
    - If the guess is incorrect, check.html will have a text box/form to make another guess
- **/complete.html**: If correct, the server will issue a **redirect** to complete.html

Maintaining State

- Notice that the server assigns a random number when the user hits the start.html page
- Throughout the series of guesses, the server needs to know what that number was.
- Remember - HTTP is stateless…
  - **One solution**: Put the assigned number in the HTML as a hidden form field.
  - The number will be sent back to the server each time…

Source code for initial version of guessing game is posted as **guessing_1.zip**

Example: Guessing Game

Key point with our design:

- Recall HTTP is *stateless* - meaning on the server, we really have no idea what previous pages were visited.
- On GET /start, we create a random number. We pass that number to the /start and /guess page as a **hidden** field.
- Each time the user enters a guess (POST), the server examine the hidden field, and the user's guess

Next Lecture - State revisited

Our example works, but lets [view source]...

State information that we don't want the user to be able to see needs to stay on the server!

Next time, we will learn about **Session** data - where the server associates data with a particular user over a series of HTTP requests