Introduction to Ember and Handlebars

become a javascript sorcer{{#if user.isFemale}}ess {{else}}er{{/if}}

Dr. Hale
University of Nebraska at Omaha
Secure Web Application Development – Lecture 2

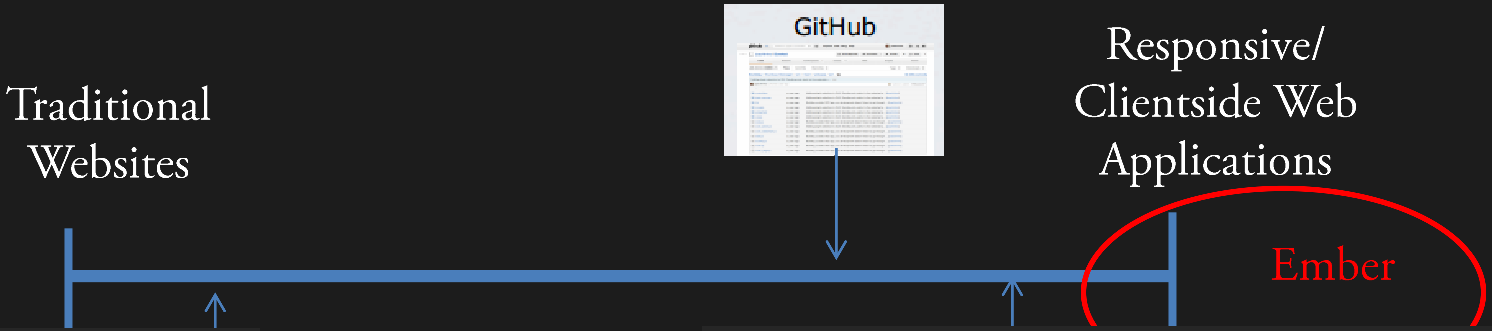
Today's topics:

Ember

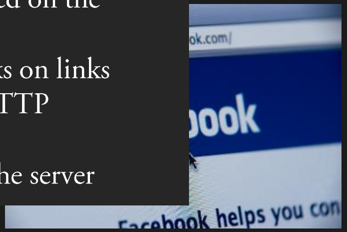
- What is Ember.js?
- Where does it sit in the server stack?
- Clientside Model/View(template)/Controller
- Ember bindings (HEAVEN)
- Handlebars Syntax
- Recap: Why its amazing (performance analysis details)
- Integrating 3rd Party jQuery Libraries
- Using Ember tools
- Integrating with a server-side API

Ember: What is it?

- A JavaScript-based Application Framework that helps you write rich web applications that are well structured (MVC) and backend agnostic.
- Helps you cut down the amount of boiler plate code within your application significantly
- Easily integrated with third party JavaScript libraries and CSS libraries like Twitter Bootstrap
- Technology of tomorrow for building responsive web applications



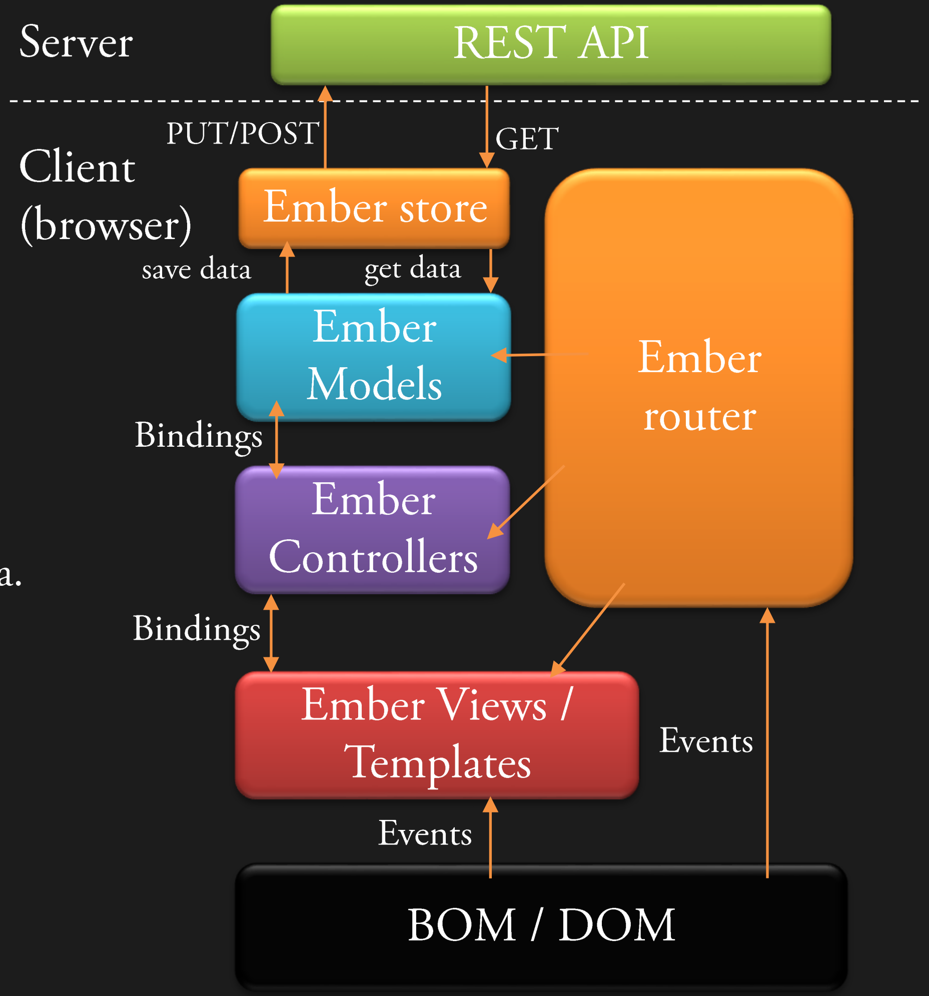
- HTML, CSS and JavaScript serialized on the Server-side and passed to the client
- Full-page refresh when the user clicks on links
- Heavily reliant on the traditional HTTP Request/response lifecycle
- Easy to cache the website pages on the server



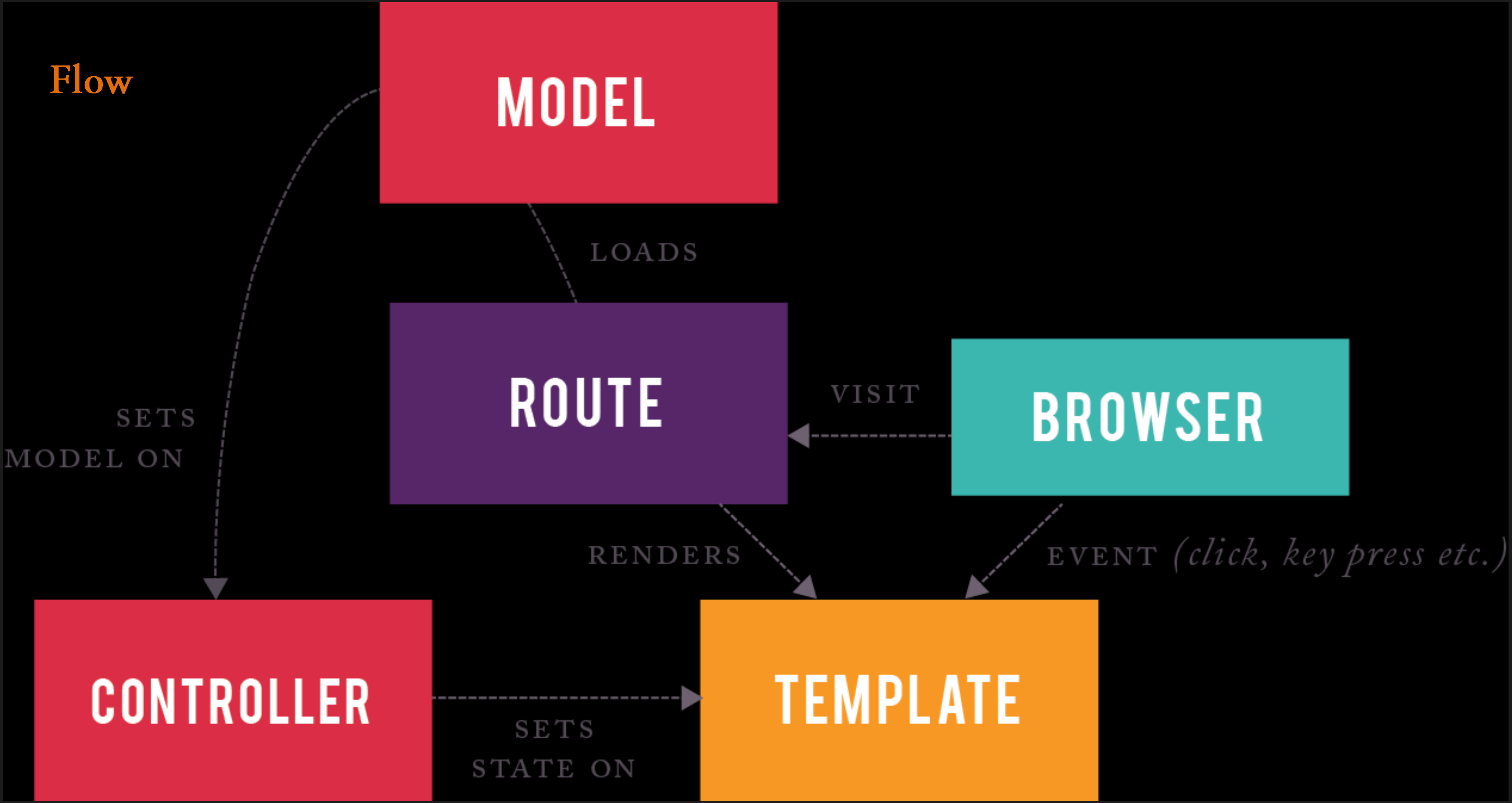
- Full JavaScript application sent on the first request
- Subsequent requests only transmit data
- Usually have a rich application-like user interface
- Faster navigation and user interaction
- Supports the rich feature set that users have come to expect from modern web based applications

Where ember resides

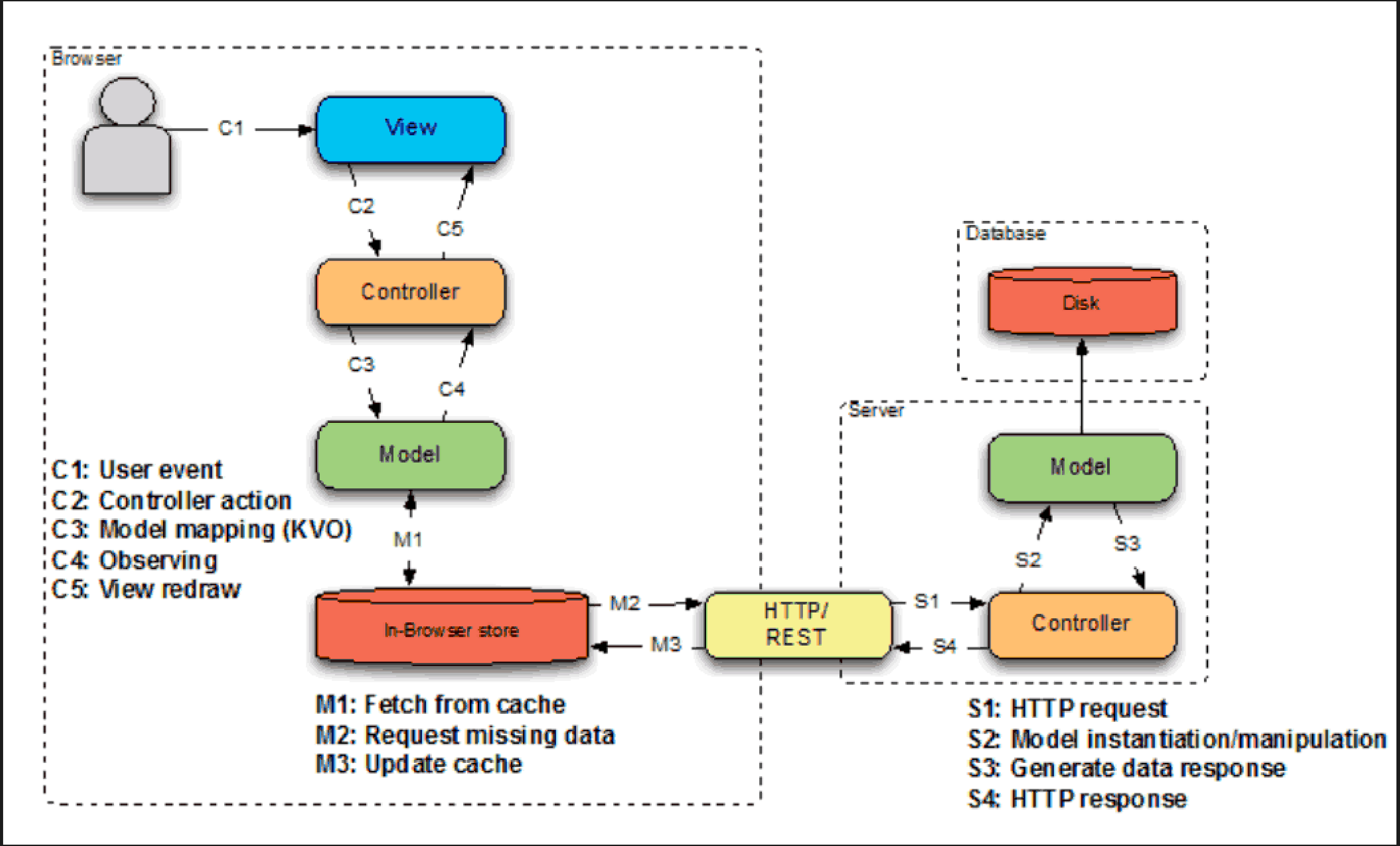
Ember executes purely on the clientside. It requires 0 server side functionality to work. Serverside functionality is only used for getting or persisting data.



Flow


MODEL
LOADS
ROUTE
VISIT
BROWSER
SETS MODEL ON
RENDERS
EVENT (click, key press etc.)
CONTROLLER
SETS STATE ON
TEMPLATE

Flow



C1: User event
C2: Controller action
C3: Model mapping (KVO)
C4: Observing
C5: View redraw

M1: Fetch from cache
M2: Request missing data
M3: Update cache

S1: HTTP request
S2: Model instantiation/manipulation
S3: Generate data response
S4: HTTP response

MVC

Views / Templates

- User interface
- Responsible for rendering elements
- Responsible for reacting to user actions

Controller / Router

- Handles events dispatched by the view/template
- Makes data structures available to view/template
- Responsible for client-server communication (if any)

Model

- Holds the data for the application
- Responsible for most data formatting
- Structures data fields
- Manipulated by controllers

Router

The router is the switchboard statemachine that tells ember what to load based on the user’s request (urls).

- urls are states
- Handles actions that are transitions between states
- Sets up model data for the controller to manipulate
- Renders a template for user viewing

Router (example)

```
Router.map(function() {
  this.route('home'); //path will be at /home
  this.route('about', { path: '/favs' });
  this.route('photos');
  this.route('photo', { path: '/photo/:photo_id'}); //dynamic segments can pass parameters
});
```

Each route maps to a URL and can be modified in the file of the same name e.g. ../app/routes/photo

A single Route (example)

```
//app/routes/photo.js
export default Ember.Route.extend({//creates a route called “PhotoRoute”
  somevariable: null,
  beforeModel(transition) {//an authenticated route redirection
    if (!this.controllerFor('auth').get('userIsLoggedIn')) {
      var loginController = this.controllerFor('login');
      loginController.set('previousTransition', transition);
      this.transitionTo('login');
    }
  },
  model(params) {//special hook that allows you to set up a data model
    this.set(‘somevariable’, params.photo_id);
    return Ember.$.getJSON('https://www.[myapi].com/photos/‘+params.photo_id);
  },
  setupController(controller, model) {//hook that allows you to set controller variables
    controller.set(‘somevariable‘, this.get(‘somevariable’));
    controller.set(‘photo‘, model);
  },
  actions: {//set of actions that can be invoked by the controller
    refreshModel (){
      this.refresh();//an action that forces the route to refresh its data model
    }
  }
});
```

Model

Ember models are just javascript objects to represent data on the clientside

- Just provides an abstraction layer to encapsulate data
- You can use ember data or jQuery (Ember.$.getJSON()) to get data from a server
- Ember-data uses a store (next)

A single model (example)

```
//app/models/photo.js

import DS from 'ember-data';

export default DS.Model.extend({
          title: DS.attr('string'),
          dates: DS.attr('object'),
          owner: DS.attr('object'),
          description: DS.attr('string'),
          link: DS.attr('string'),
          views: DS.attr('number'),
          tags: DS.attr('object'),
          //flickr url data
          id: '',
          farm: DS.attr('number'),
          secret: DS.attr('string'),
          server: DS.attr('string'),
          url: function(){//computed property
                    return "https://farm"+this.get('farm')+
                    ".staticflickr.com/"+this.get('server')+
                    "/"+this.get('id')+"_"+this.get('secret')+"_b.jpg";
          }.property('farm','server','id','secret'),
});
```

The JSON produced will look like:

```
photo: {
    title: ‘hi’,
    dates: [‘8/31/15’],
    description: ‘saying hi to my friend’,
    link: ‘www.mywebsite.com’,
    views: 271,
    tags: [‘hi’, ‘2015’, ‘friends’, ‘fb’],
    id: ‘9087193412341234’,
    farm: 2,
    secret: ‘b35213dasfasd9124’,
    server: ‘17’,
    url: ‘https://farm2.staticflickr.com/17/9087193412341234_
          b35213dasfasd9124_b.jpg’
}
```

Controller

The controller is the brains of the operation.

- Handles user actions, relay url state changes to the router.
- Manipulates model data
- Can make additional ajax requests to API
- Used to control elements rendered in a template

A single controller (example)

```
//app/controllers/photo.js
import Ember from 'ember';

export default Ember.Controller.extend({
   somebooleanvariable: true,
   somevalue: 1,
   somecomputedproperty: function(){
      return this.get(‘somevalue’) +1;
   }.property(‘somevalue’),
   actions: {
     someAction: function(){
        //do stuff here
        console.log(‘this will create a debug statement in the console’)
     },
     refreshData: function(){
       this.send(‘refreshModel’);//this will fire an action off to the router
     }
   },
   init: function(){
     //this method is called when the controller first runs
   }
});
```

Handlebars Templates

Handlebars templates are the user interface in ember. They generate HTML when rendered and maintain bindings to controller/model variables.

- Update automatically whenever bound variables are changed in the controller/model
- Rendered by the router when the user visits a particular url
- Pass user events to controller/router to be handled

A single template (example)

```
//app/templates/photo.hbs
<div class="container">

          {{#if somebooleanvariable}}
                   {{photo.title}} ({{photo.username}})<br>
                    <img class="feed-img" src={{photo.url}} />

                    <ul class="list-group">
                              {{#each tag in photo.tags}}
                                 <li class="list-group-item">#{{tag}}</li>
                              {{/each}}
                    </ul>
                    <button {{action ‘refreshData'}}>Refresh Photo</button>
          {{else}}
                    <div class="jumbotron">
                              <p>Search for a photo</p>
                    </div>
          {{/if}}
          <button {{action ‘someAction'}}>some action</button>
          Text field binding: <input type="text" value="somevalue"/>
          {{somecomputedproperty}} – will display ^ +1

</div>
```

Handlebars

- A JavaScript-based template engine that allows you to write complex templates in a familiar HTML-esque style
- Ember.js uses ~~Handlebars.js~~ HTMLBars in order to enrich the view-layer with semantic templates
- Contains both simple expressions and block expressions that allow you to express template-based logic

Handlebars basics

- Simple expressions are identifiers that tell Handlebars.js to replace the contents of the expression at runtime

```
<h1>{{title}}</h1>
```

- Expressions can traverse into your object-structure via dot-notation

```
<h1>{{book.title}}</h1>
```

- Simple expressions can also render complex views, and have properties and property-bindings

```
{{render header}}
{{view LDBB.KeyListView contentBinding="this"}}
```

Handlebars basics

ships with a range of simple expressions

`{{view}}` – Renders a view

`{{bindAttr}}` – Renders a DOM element attribute

`{{action}}` – Triggers an action on click

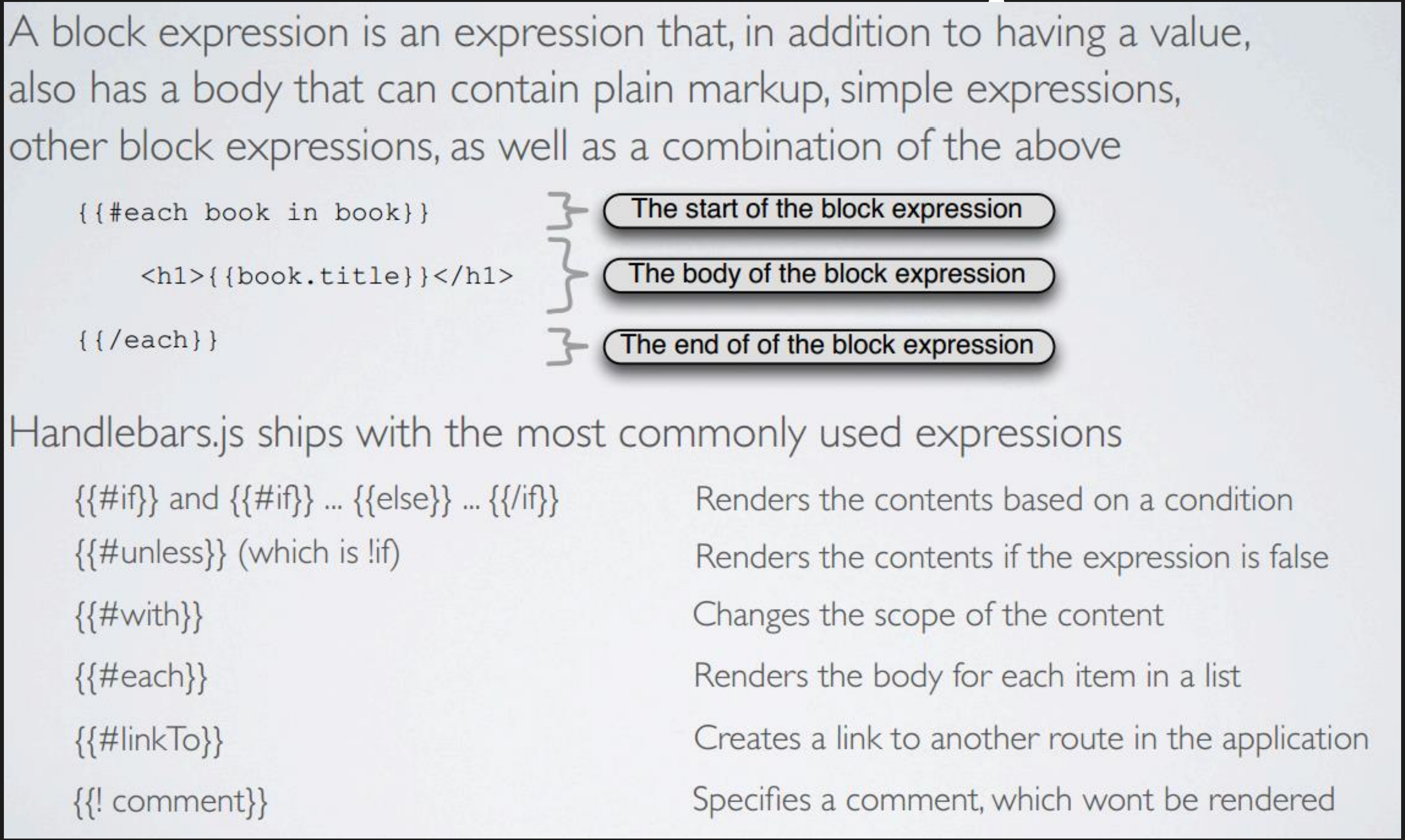
`{{outlet}}` – Renders a sub route

`{{template}}` – Render a template

`{{render}}` – Render a template backed by a controller and a view

`{{component}}` – Render a component

Handlebars Block Expressions



A block expression is an expression that, in addition to having a value, also has a body that can contain plain markup, simple expressions, other block expressions, as well as a combination of the above

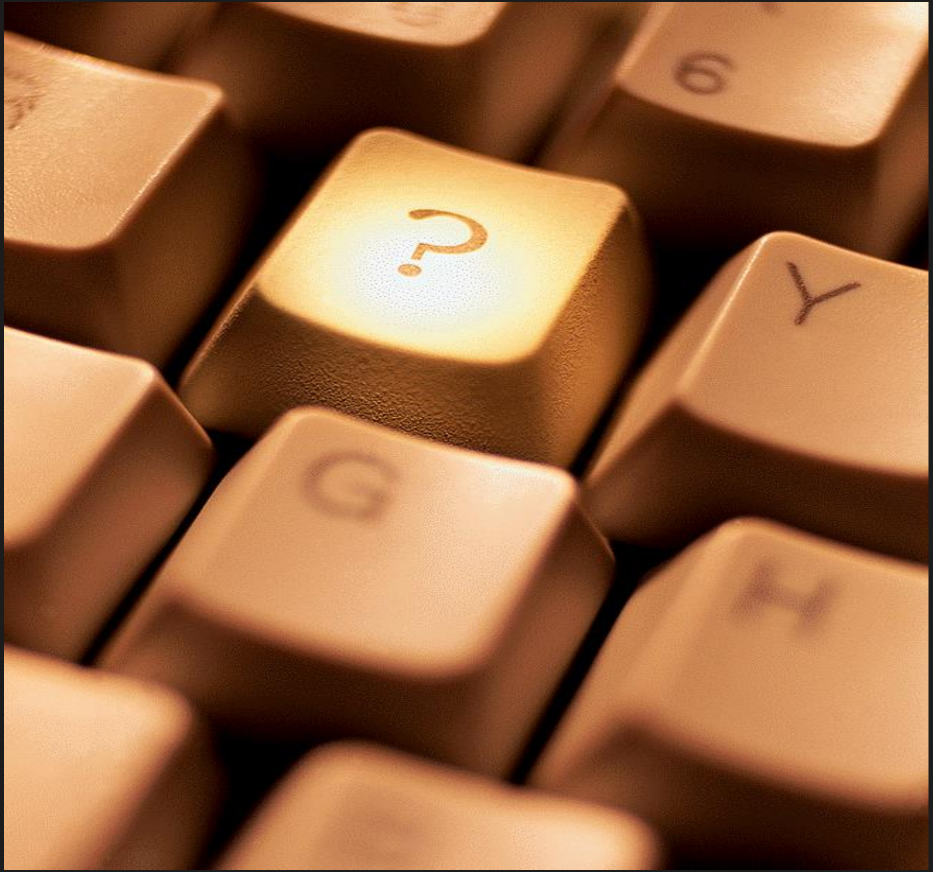
```
{{#each book in book}}

    <h1>{{book.title}}</h1>

{{/each}}
```

The start of the block expression
The body of the block expression
The end of of the block expression

Handlebars.js ships with the most commonly used expressions

{{#if}} and {{#if}} ... {{else}} ... {{/if}}	Renders the contents based on a condition
{{#unless}} (which is !if)	Renders the contents if the expression is false
{{#with}}	Changes the scope of the content
{{#each}}	Renders the body for each item in a list
{{#linkTo}}	Creates a link to another route in the application
{{! comment}}	Specifies a comment, which wont be rendered

Questions?



Matt Hale, PhD

University of Nebraska at Omaha

Interdisciplinary Informatics

mlhale@unomaha.edu

Twitter: @mlhale_

© 2015-2016 Matthew L. Hale