Ember 2

Section 7 - Components

Component Essentials
Building Complex Components

Section Objectives

- Be able to define a Component
- Understand the Component lifecycle
- Be able to pass properties to a Component
- Be able to wrap content in a Component
- Be able to customize the Component’s Element
- Be able to return values from the Component to be used in a block expression
- Be able to handle events in a Component
- Be able to trigger actions from a Component
- Be able to create a Component that filters results

Components

Component Essentials

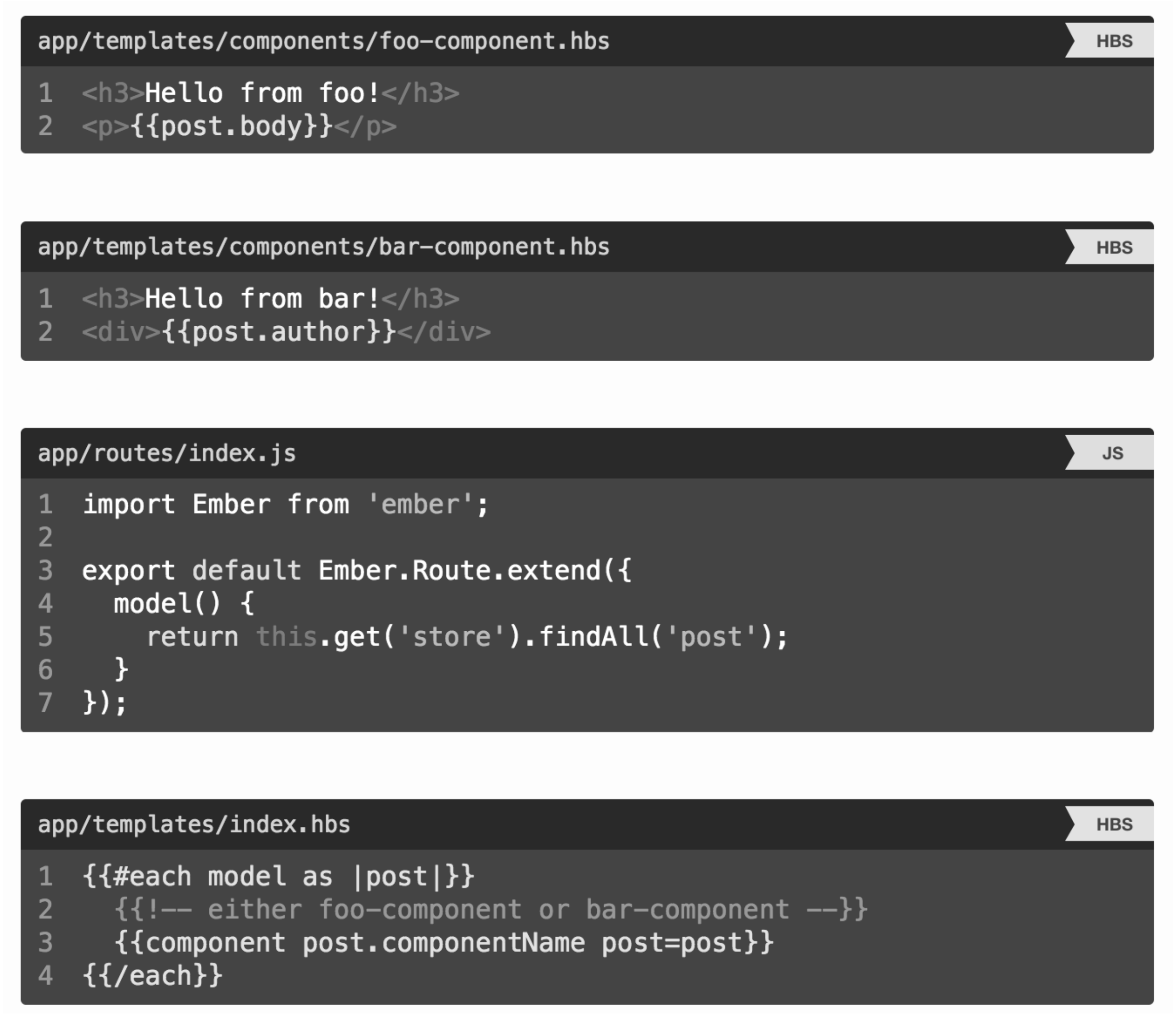
Component Essentials
Building Complex Components

Defining a Component

- ***ember g component note-row***
- Creates a template and a source file that contains properties and actions to be used by the component
- Used in a template by it’s name: **{{note-row}}**
- Can use the **{{component <name> <attributes=values>}}** helper if the specific component will be determined at runtime
- Must have at least one dash in its name to prevent clashing with current or future HTML elements

Defining a Component

- Aligns with the W3C Custom Elements spec
- The template is wrapped in a <div>



app/templates/components/foo-component.hbs — HBS

```handlebars
<h3>Hello from foo!</h3>
<p>{{post.body}}</p>
```

app/templates/components/bar-component.hbs — HBS

```handlebars
<h3>Hello from bar!</h3>
<div>{{post.author}}</div>
```

app/routes/index.js — JS

```js
import Ember from 'ember';

export default Ember.Route.extend({
  model() {
    return this.get('store').findAll('post');
  }
});
```

app/templates/index.hbs — HBS

```handlebars
{{#each model as |post|}}
  {{!-- either foo-component or bar-component --}}
  {{component post.componentName post=post}}
{{/each}}
```

https://guides.emberjs.com/v2.11.0/components/defining-a-component/

Lab 6 – Component Definition

- Create a component for folder rows similar to what we did for the note-row.
  - Model:
    - title (string)
    - selected (boolean)
    - edit (boolean)

Component Lifecycle

- On Initial Render
  1. init()
  2. didReceiveAttrs()
  3. willRender()
  4. didInsertElement()
  5. didRender()
- On Re-Render
  1. didUpdateAttrs()
  2. didReceiveAttrs()
  3. willUpdate()
  4. willRender()
  5. didUpdate()
  6. didRender()

Component Lifecycle

- On Component Destroy
  1. willDestroyElement()
  2. willClearRender()
  3. didDestroyElement()

Passing Properties

notes-folder.hbs

```
{{#each model.notes as |note|}}
    {{#link-to 'note' note}}
        {{note-row note=note selectNote=(action "showNote")
                   editMode=model.editMode
        }}
    {{/link-to}}
{{/each}}
```

Wrapping Content in a Component

- Use the component in block form and wrap the content between the tags

hbs

```
{{#note-row note=note showNote=(action "showNote")
            editMode=model.editMode}}

// wrapped content

{{/note-row}}
```

- Must use the **{{yield}}** within your template to indicate where to render the wrapped content

Components

Building Complex Components

Component Essentials
Building Complex Components

Customizing the Component’s Element

- By default, the component is wrapped in a <div>
- To change it to a different element, set the component’s ‘**tagName**’ property

js
```
export default Ember.Component.extend({
    tagName: 'nav'
});
```

- You can add class names to the component’s element by using it’s ‘**classNames**’ property

js
```
export default Ember.Component.extend({
    tagName: 'nav’,
    classNames: [‘nav-bar’]
});
```

Customizing the Component’s Class

- You can add class names to the component’s element by using it’s ‘**classNames**’ property

js

```
export default Ember.Component.extend({
    tagName: 'nav’,
    classNames: [‘nav-bar’]
});
```

- Add the class directly to the component placeholder

hbs

```
{{app-nav class=‘nav-bar’}}
```

- Use class name bindings

js

```
export default Ember.Component.extend({
    tagName: 'nav’,
    classNameBindings: [‘selected’],
    selected: false
});
```

ClassName Binding Options

- If your property is camel-case, the class name will be hypenated

js
```
export default Ember.Component.extend({
    tagName: 'nav’,
    classNameBindings: [‘isSelected’],
    isSelected: true
});
```

```
<nav class=‘is-selected’></nav>
```
hbs

- You can customize your class name like so

js
```
export default Ember.Component.extend({
    tagName: 'nav’,
    classNameBindings: [‘isSelected:selected’],
    isSelected: true
});
```

```
<nav class=‘selected’></nav>
```
hbs

ClassName Binding Options

- You can also provide a class name if the binding is false

js

```
export default Ember.Component.extend({
    tagName: 'nav’,
    classNameBindings: [‘isEnabled:enabled:disabled’],
    isEnabled: false
});
```

```
<nav class=‘disabled’></nav>
```

hbs

- If you only want a class name when the result if false

js

```
export default Ember.Component.extend({
    tagName: 'nav’,
    classNameBindings: [‘isEnabled::disabled’],
    isEnabled: false
});
```

```
<nav class=‘disabled’></nav>
```

hbs

ClassName Binding Options

- If the bound value is a string, then the value will be used as the class name

js

```
export default Ember.Component.extend({
    classNameBindings: [‘status’],
    status: ‘error’
});
```

hbs

```
<div class=‘error’></div>
```

Using Block Params

- Using your component as a block parameter allows you to wrap other content and share data with that content

index.hbs

```
{{blog-post post=model}}
```

blog-post.hbs

```
{{yield post.title post.body post.author}}
```

index.hbs

```
{{#blog-post post=model as |title body author|}}
    <h2>{{title}}</h2>
    <p class="author">by {{author}}</p>
    <div class="post-body">{{body}}</div>
{{/blog-post}}
```

Supporting Block and Non-Block Usage

- Using the ‘hasBlock helper will allow you to handle the component rendering in block form or non-block form

index.hbs

```
{{#blog-post post=model as |title body author|}}
    <h2>{{title}}</h2>
    <p class="author">by {{author}}</p>
    <div class="post-body">{{body}}</div>
{{/blog-post}}
```

blog-post.hbs

```
{{#if hasBlock}}
    {{yield post.title}}
    {{yield post.body}}
    {{yield post.author}}
{{else}}
    <h1>{{post.title}}</h1>
    <p class="author">Authored by {{post.author}}</p>
    <p>{{post.body}}</p>
{{/if}}
```

Event Handling

- Handle events on your component by using the available event methods outlined below and the following slides

- To allow your events to bubble up to their parent, **return true;**

- **Touch Events**
  - touchStart()
  - touchMove()
  - touchEnd()
  - touchCancel()

Event Handling

- **Keyboard Events**
  - keyDown()
  - keyUp()
  - keyPress()

- **Form Events**
  - submit()
  - change()
  - focusIn()
  - focusOut()
  - input()

Event Handling

- **Mouse Events**
  - – mouseDown()
  - – mouseUp()
  - – contextMenu()
  - – click()
  - – doubleClick()
  - – mouseMove()
  - – focusIn()
  - – focusOut()
  - – mouseEnter()
  - – mouseLeave()

Event Handling

- **HTML 5 Drag-n-Drop**
  - dragStart()
  - drag()
  - dragEnter()
  - dragLeave()
  - dragOver()
  - dragEnd()
  - drop()

Action Triggers

1. Create the component

notes-folder.hbs

```
{{note-row note=note selectNote=(action "showNote")
           editMode=model.editMode
}}
```

2. Design the Action
   - In the parent, decide how you want to react to the action
   - In the component, decide when to tell the parent when to initiate the action

Action Triggers

3. Implement the Action

notes-folder.js

```
actions: {
    showNote(note) {
        this.set('model.selectedNote', note);
        this.get('model.notes').forEach((current_note) => {
            current_note.set('selected', false);
            if(current_note === note) {
                current_note.set('selected', true);
            }
        });
    }
}
```

Action Triggers

4. Implement the child component

note-row.js

```
actions: {
    selectNote() {
        if(!this.get('editMode')) {
            this.set('selected', true);
            this.sendAction('showNote', this.get('note'));
        }
    }
}
```

OR

note-row.js

```
actions: {
    selectNote() {
        if(!this.get('editMode')) {
            this.set('selected', true);
            this.get(‘selectNote’)(this.get('note'));
        }
    }
}
```

Passing Arguments

notes-folder.hbs

```
{{note-row note=note selectNote=(action "showNote” note)
           editMode=model.editMode
}}
```

note-row.js

```
actions: {
    selectNote() {
        if(!this.get('editMode')) {
            this.set('selected', true);
            this.get(’selectNote’)();
        }
    }
}
```

Creating a Filter

- Let’s see how we might filter notes by their folder

notes-folder.hbs

```
{{#notes–filter filter=(action ”filterByFolder” folder) as |notes|}}
    {{#each notes as |note|}}
        {{#link-to 'note' note}}
            {{note-row note=note
                       showNote=(action "showNote" note)
                       editMode=model.editMode
                       numSelected=model.numSelected
            }}
        {{/link-to}}
    {{/each}}
{{/note-filter}}
```

note-filter.hbs

```
{{input value=value key-up=(action ’filterNotes') class="light"
        placeholder="Filter By City"}}
{{yield results}}
```

Creating a Filter

■ Let’s see how we might filter notes by their folder

■ Filter Component

note-filter.js

```
export default Ember.Component.extend({
    classNames: [’notes'],
    folder: ‘’,
    results: null,
    filterNotes() {
       this.get(‘filter’)(this.get(‘folder’)).then((notes) =>
           this.set(‘results’, notes)
       );
    }
});
```

Creating a Filter

- Parent Template

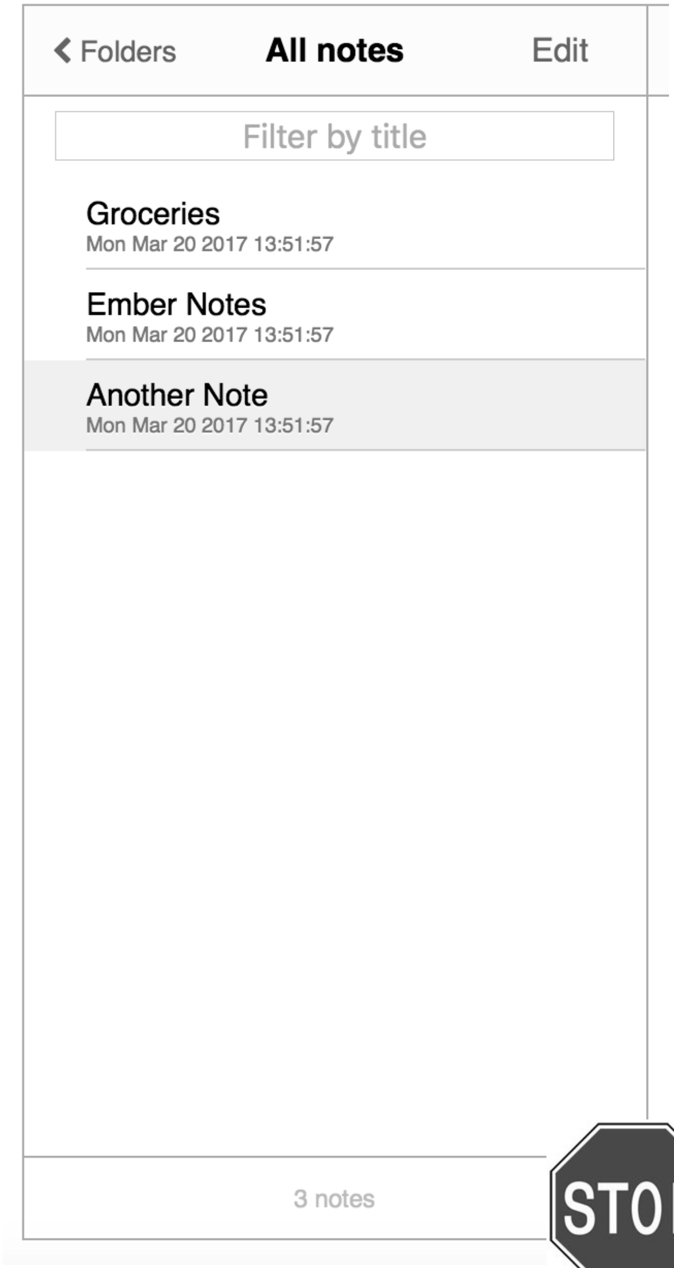
notes-folder.hbs

```
actions: {
    filterByFolder(param) {
       let notes = this.get('store').peekAll('note');
       if (param == '') {
          return notes;
       } else {
          return notes.filter((note) => note.folder === param);
       }
}
```

Lab 7 – Notes Filter

- Filter the notes under their folder.
  (Hint: Design first and think through it based on your current architecture)

- Implement a search filter for notes within a folder.
  (Hint: You will need to use the component didUpdatedAttrs() to update notes under custom folders when adding new notes to update the results)



Groceries
Mon Mar 20 2017 13:51:57

Ember Notes
Mon Mar 20 2017 13:51:57

Another Note
Mon Mar 20 2017 13:51:57

Lab 8 – Refactor to Components

- Refactor markup that is being duplicated into components
  - master header
  - toolbar

- Reuse the ‘note-row’ component for the folder rows in the notes template. You will have to genericize your component to work for both. Be sure to spend time designing before jumping into the code.

STOP

Lab 9 – Delete Action

- Provide the implementation for the Delete Button
  - **HINT**: User *unloadRecord(<record>)* from the store.

Section Review

- Be able to define a Component
- Understand the Component lifecycle
- Be able to pass properties to a Component
- Be able to wrap content in a Component
- Be able to customize the Component's Element
- Be able to return values from the Component to be used in a block expression
- Be able to handle events in a Component
- Be able to trigger actions from a Component
- Be able to create a Component that filters results

End of Section

- This slide is purposely devoid of any information