People matter, results count.

Objectives of Ember JavaScript

- **Purpose:**
  - Basic understanding of basics of Ember Framework
- **Product:**
  - Understanding Ember Architecture
  - Understanding and Designing MVC ,SPA based Web Application using Ember
- **Process:**
  - Theory Sessions along with relevant assignments
  - A review at the end of the session and a Quiz

Table of Contents

Introduction

- Open-source JavaScript web framework
- Emberjs is a MVC+R (Model + View + Controller + Route) Framework.
- Ember is heavily route driven when compared with other MVC frameworks Used for designed for Single Page Applications
- Two-Way binding
- Although primarily considered a framework for the web, it is also possible to build desktop and mobile applications in Ember.
- The most notable example of an Ember desktop application is Apple Music, a feature of the iTunes desktop application.

Features of Ember

- Ember.js is used for to creating reusable and maintainable JavaScript web applications.
- Ember.js views are created by using Handlebars templates and are simple to develop the front-end design.
- It automatically determines the route and controller during declaration of the route resources.
- Ember.js eliminates the boilerplate(the section of code that has to be included in various places without any alteration) and provides standard application architecture.
- Ember.js has HTML and CSS at the core of the development model.
- The routes are core features of the Ember.js which are used for to manage an URL's.
- It has extensive view type support.
- Ember.js uses templates that helps to automatically update the model, if contents of an applications gets change.

Angular Vs Ember

Angular.js	Ember.js
✔ popular (#4 github.com)	✔ popular (#22 github.com)
✔ easy to learn	✘ ~~easy to learn~~
✘ ~~convention over configuration~~	✔ convention over configuration
✘ ~~url-support~~	✔ url-support
✘ ~~nested UIs~~	✔ nested UIs
✘ ~~minimizes DOM-updates~~	✔ minimizes DOM-updates
✘ ~~data-store~~	✔ data-store

Environment Setup

- To work on Ember ,few JavaScript libraries needs to be included in <script> tag.
- Before installing emberjs, it should require nodejs installed on system with compatible browser.
- Use the following command in nodejs command line interface to install emberjs.

  npm install -g ember-cli
- To create an application, use the following command.

  ember new my-app
- Once Completed, include following JavaScript libraries from CDN site.

  jQuery

  Handlebars

  Ember.js

Environment Setup(Contd…)

- https://cdnjs.cloudflare.com/ajax/libs/ember.js/2.0.1/ember-template-compiler.js
- https://cdnjs.cloudflare.com/ajax/libs/ember.js/2.0.1/ember-testing.js
- https://cdnjs.cloudflare.com/ajax/libs/ember.js/2.0.1/ember.debug.js
- https://cdnjs.cloudflare.com/ajax/libs/ember.js/2.0.1/ember.js
- https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/3.0.1/handlebars.min.js
- https://code.jquery.com/jquery-2.1.3.min.js
- https://cdnjs.cloudflare.com/ajax/libs/ember.js/2.0.1/ember.min.js

Ember Cli

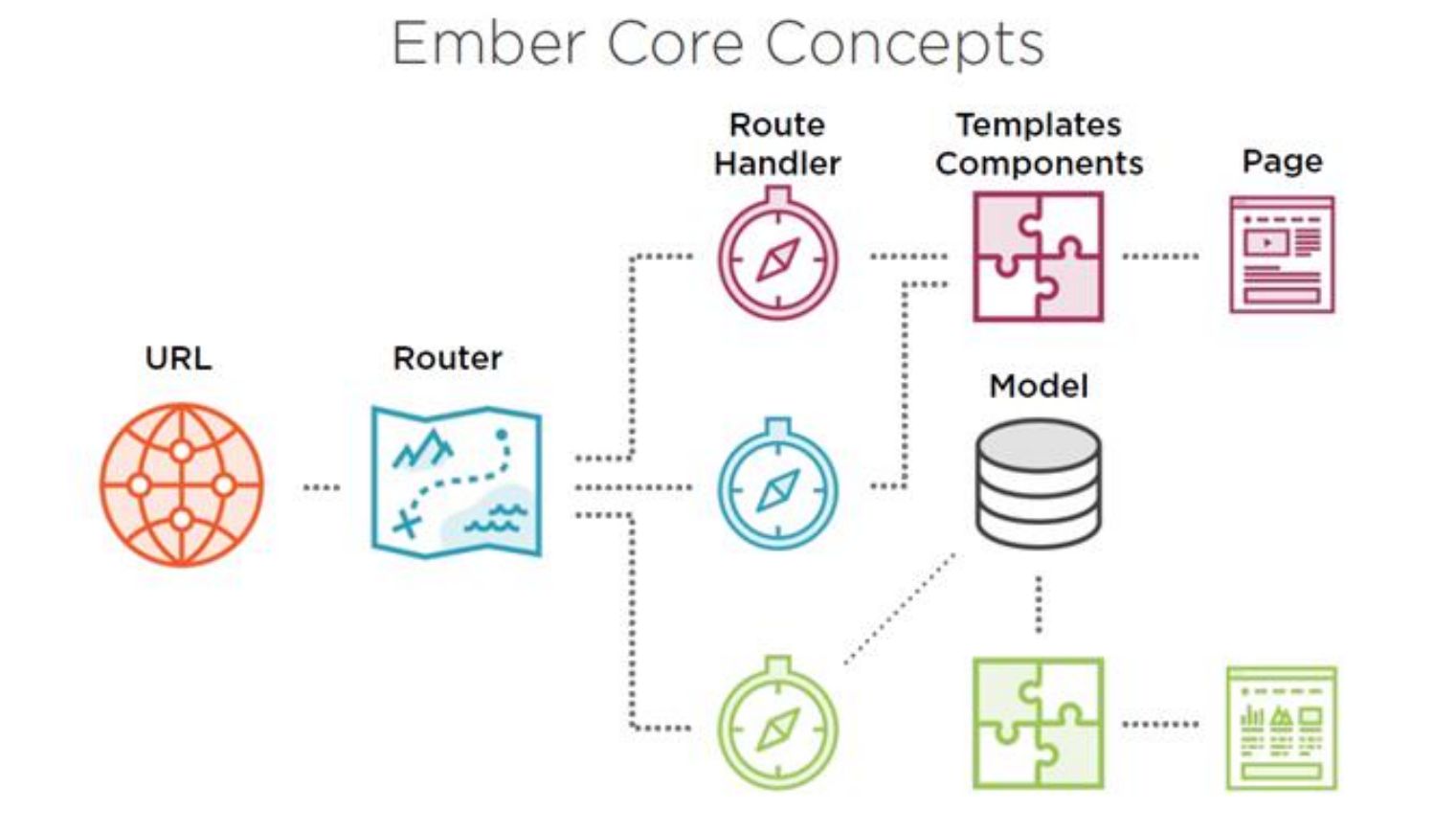
- Ember CLI is the Ember.js command line utility that provides a fast Strong conventional project structure, and a powerful addon system for extension.
- Below is the command to install Ember cli:

  npm install -g ember-cli
- Ember CLI has support for:
  - Handlebars
  - HTMLBars
  - Emblem
  - LESS
  - Sass
  - Compass
  - Stylus
  - CoffeeScript
  - EmberScript
  - Minified JS & CSS

Ember Cli(Contd…)

Command	Purpose
ember	Prints out a list of available commands.
ember new <app-name>	Creates a directory called <app-name> and in it, generates an application structure. If git is available the directory will be initialized as a git repository and an initial commit will be created. Use --skip-git flag to disable this feature.
ember init	Generates an application structure in the current directory.
ember build	Builds the application into the dist/ directory (customize via the --output-pathflag). Use the --environment flag to specify the build environment (defaults to development). Use the --watch flag to keep the process running and rebuilding when changes occur.
ember server	Starts the server. The default port is 4200. Use the --proxy flag to proxy all ajax requests to the given address. For example, ember server --proxy http://127.0.0.1:8080 will proxy all ajax requests to the server running at http://127.0.0.1:8080. Aliases: ember s, ember serve
ember generate <generator-name> <options>	Runs a specific generator. To see available generators, run ember help generate. Alias: ember g
ember destroy <generator-name> <options>	Removes code created by the generate command. If the code was generated with the --pod flag, you must use the same flag when running the destroy command. Alias: ember d
ember test	Run tests with Testem in CI mode. You can pass any options to Testem through a testem.json file. By default, Ember CLI will search for it under your project's root. Alternatively, you can specify a config-file. Alias: ember t
ember install <addon-name>	Installs the given addon into your project and saves it to the package.json file. If provided, the command will run the addon's default blueprint.

Architecture



Architecture

Maps:
-URL to a route

Loads:
-a template
-a model

Persists to:
-web server

Loads:
-components

Accesses:
-model data

URL

```
http://localhost:4200/rentals
```

Router
router.js

```
Router.map(function() {
  this.route('rentals');
});
```

Route Handler
routes/rentals.js

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

Model

Template
templates/rentals.hbs

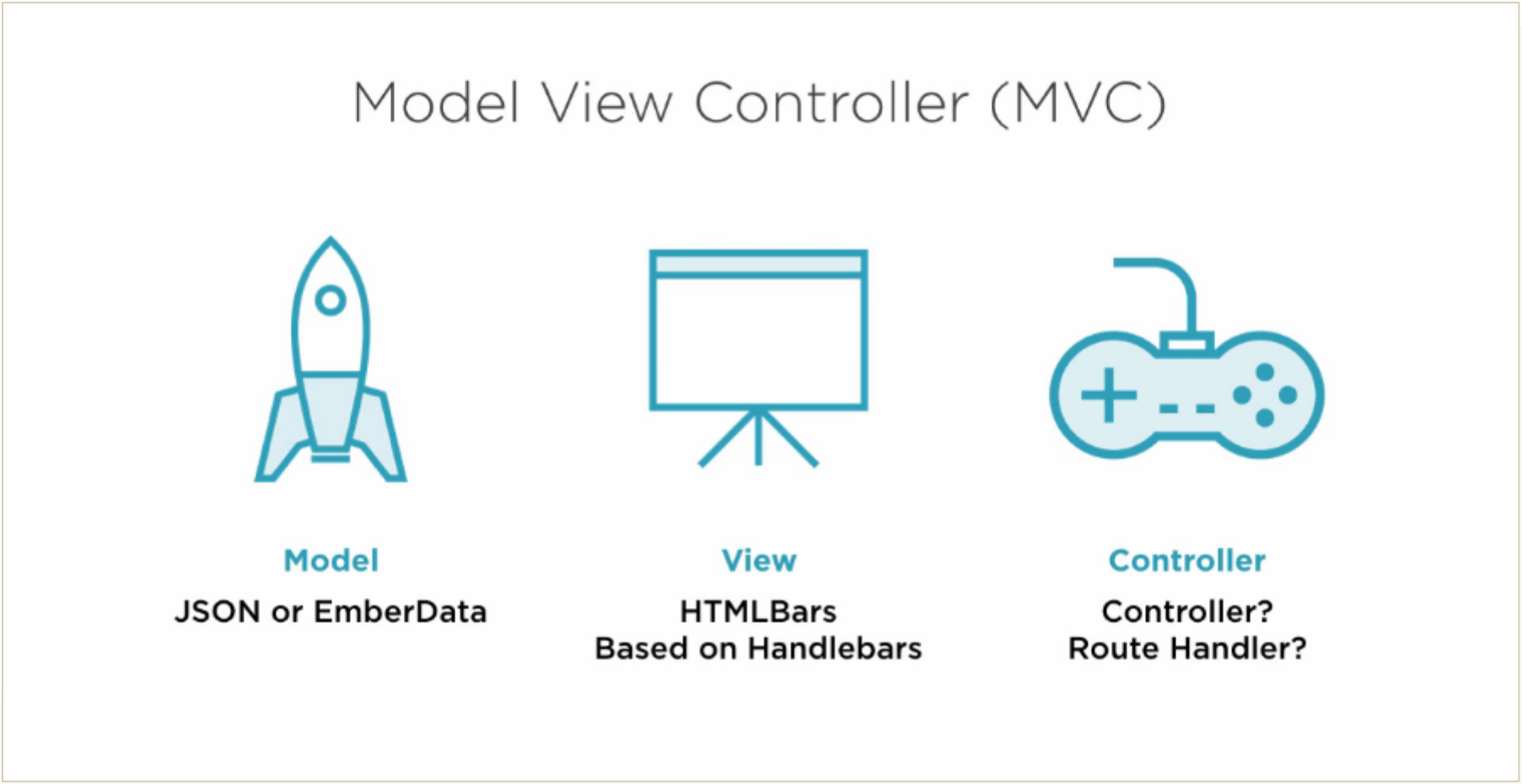
```
<ul>
  {{#each model as |rental-unit|}}
    {{rental-tile rental=rental-unit}}
  {{/each}}
</ul>
```

Component
app/components/rental-tile.js
templates/components/rental-tile.hbs

Component
app/components/rental-tile.js
templates/components/rental-tile.hbs

Component
app/components/rental-tile.js
templates/components/rental-tile.hbs

MVC Architecture



Ember Definitions

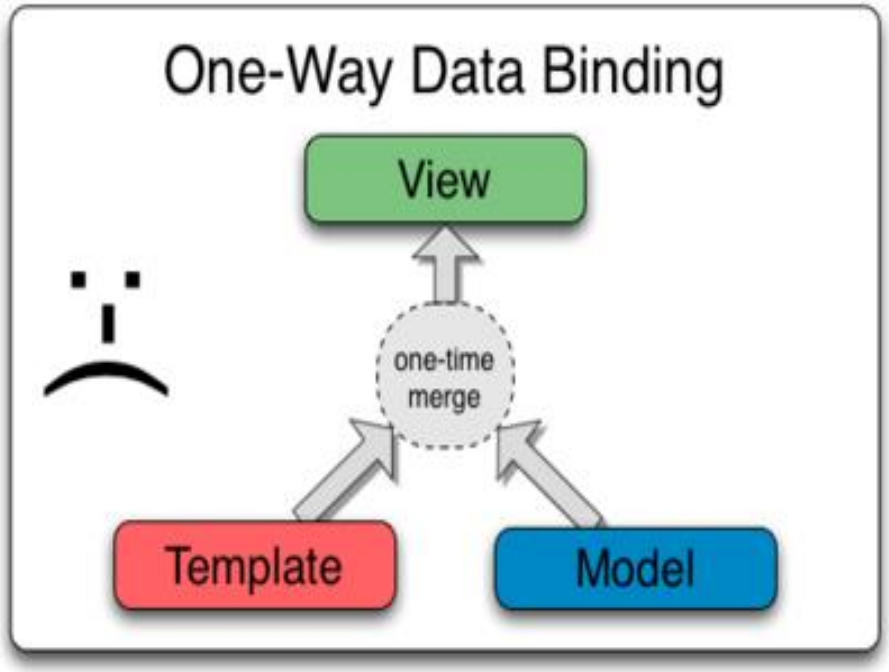
Ember.js Definitions

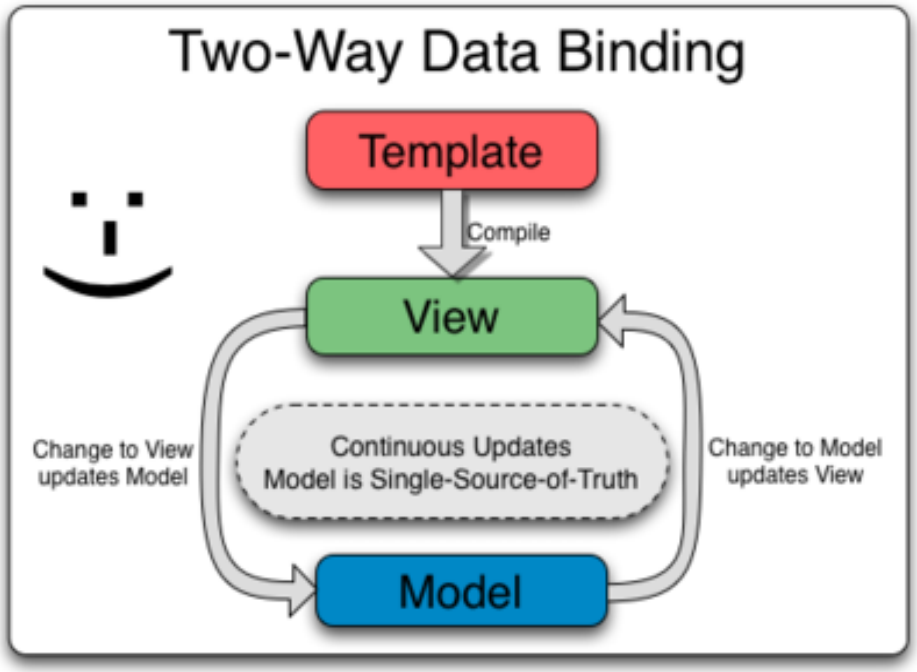
Router	Maps the URL to a route handler
Route Handler	Loads the model and renders the template
Model	Represents persistent state
Templates	Organize and describe how the interface looks
Components	Control how the interface behaves

Bindings

- Data binding is a technique to establish a binding or connection between the data reflected in UI and that present in your model, processed by business logic.
- In Two way data binding ,if you change something in your model it will change the view accordingly and vice versa. Here model doesn't refer to any separate or specific scope.
- Even if you are reflecting a property or variable of view in your template and in some point of time the value of the variable got changed; the changed value will be reflected automatically in your UI template.
- The concept of data binding has almost removed the use of 'print' or 'document.write'.
- Now you can easily get read of writing a function to print the new value again and again; you don't have to take the overhead of managing your UI according to your model.

Bindings(contd…)





Handlebars

- Templates are useful when building web applications by allowing you to easily display repeating data.
- Dynamic content inside a Handlebars expression is rendered with data-binding.
- This means if you update a property, your usage of that property in a template will be automatically updated to the latest value.
- A templating engine such as Handlebars.js allows you to take repeating data and easily display it in an organized fashion.
- Templates keep your code clean and easily maintainable while separating your logic-based code from your views.
- Handlebars templates can be used in JavaScript frameworks such as Backbone and Ember.
- A handlebars expression is a {{, some contents, followed by a }}
- Adding handlebars to your project is pretty straightforward. Just go to http://handlebarsjs.com/ , and click the download button to get the latest version.

Handlebars(contd…)

Conditionals:

- {{if isFast "zoooom" "putt-putt-putt"}}

  {{if}} in this case returns "zoooom" when isFast is true and "putt-putt-putt" when isFast is false.

- {{if isFast (if isFueled "zoooom")}}

  {{#if person}}
  Welcome back,
  {{person.firstName}} {{person.lastName}}!
  {{/if}}

- {{#if person}}

  Welcome back, {{person.firstName}} {{person.lastName}}!

  {{else}}

  Please log in.

{{#if person}} Welcome back, {{person.firstName}} {{person.lastName}}! {{else}} Please log in. {{/if}} {{/if}}

Handlebars(contd…)

- {{#each people as |person|}}
  Hello, {{person.name}}!

  {{/each}}
- {{#each people as |person index|}}
  Hello, {{person.name}}! You're number {{index}} in line
  {{/each}}

- **Empty List**
- {{#each people as |person|}}
- Hello, {{person.name}}! {{else}} Sorry, nobody is here. {{/each}}

Handlebars(contd…)

- **Binding Element Attributes:**

```
<div id="logo">
<img src={{logoUrl}} alt="Logo">
</div>
```

- If you use data binding with a Boolean value, it will add or remove the specified attribute.

```
<input type="checkbox" disabled={{isAdministrator}}>
```

- **Link to Router:**

create a link to a route using the {{link-to}} helper.

```
{{#link-to "photos.edit" photo}}{{photo.title}}{{/link-to}}
```

- The name of a route. In this example, it would be index, photos, or photos.edit.
- At most one model for each dynamic segment.
- By default, Ember.js will replace each segment with the value of the corresponding object's id property.

Handlebars(contd…)

Actions:

- Your app will often need a way to let users interact with controls that change application state.
- {{action}} is used to achieve actions in handlebars.

Specifying the Type of Event

- By default, the {{action}} helper listens for click events and triggers the action when the user clicks on the element.
- You can specify an alternative event by using the on option.
- <button {{action "select" post on="mouseUp"}}>✓</button> {{post.title}}
  - You should use the camelCased event names, so two-word names like keypressbecome keyPress.
- For more info on HTML tags with handlebars, visit below link:

  https://guides.emberjs.com/v2.12.0/templates/input-helpers/

Handlebars(Contd…)

- Handlebars templates look like regular HTML, with embedded handlebars expressions.

```
<div class="entry">
 <h1>{{title}}</h1>
 <div class="body">
 {{body}}
 </div>
</div>
```

```
<script id="entry-template" type="text/x-
handlebars-template">
 <div class="entry">
 <h1>{{title}}</h1>
 <div class="body">
 {{body}}
</div>
</div>
</script>
```

Object Model

- In Ember.js, all objects are derived from the *Ember.Object.*
- Ember.Object also provides a class system, supporting features like mixins and constructor methods.

Defining Classes

- To define a new Ember class, call the extend() method on Ember.Object:

```
const Person = Ember.Object.extend({
  say(thing)
  {
    alert(thing);
  }
                                        });
```

- This defines a new Person class with a say() method.

Object Model(Contd…)

Creating Instances

- Once you have defined a class, you can create new *instances* of that class by calling its create() method.
- Any methods, properties and computed properties you defined on the class will be available to instances:

```
const Person = Ember.Object.extend(
{
 say(thing)
{
alert(`${this.get('name')}
says: ${thing}`);
}
});
 let person = Person.create();
person.say('Hello'); // alerts " says: Hello"
```

Object Model(Contd…)

Accessing Object Properties:

- When accessing the properties of an object, use the get() and set() accessor methods:

```
const Person = Ember.Object.extend({
    name: 'Robert Jackson'
     });
let person = Person.create();
person.get('name'); // 'Robert Jackson‘
 person.set('name', 'Tobias Fünke');
 person.get('name'); // 'Tobias Fünke'
```

- Make sure to use these accessor methods; otherwise, computed properties won't recalculate, observers won't fire, and templates won't update.

Templates

- Ember uses templates to organize the layout of HTML in an application.
- Most templates in an Ember codebase are instantly familiar, and look like any fragment of HTML. For example:

```
1 <div>Hi, this is a valid Ember template!</div>
```

- Ember templates use the syntax of Handlebars templates. Anything that is valid Handlebars syntax is valid Ember syntax.
- Templates can also display properties provided to them from their context, which is either a component or a route's controller.
- For example:

```
1 <div>Hi {{name}}, this is a valid Ember template!</div>
```

Here, {{name}} is a property provided by the template's context.

Routing

- Router is a core feature of the EmberJs which translates an URL into the series of templates and represents the state of an application.
- The Ember uses the router to map the URL to a route handler.
- The router matches the current URL to other routes which are used for loading data, displaying the templates and to set up an application state.
- Route handler performs some actions such as:
- It provides the template.
- It defines the model and it will be accessible to the template.
- If there is no permission for user to visit the particular part of an app, then router will redirect to a new route.

Routing(contd…)

- The map() method of your Ember application's router can be invoked to define URL mappings.
- When calling map(), you should pass a function that will be invoked with the value this set to an object which you can use to create routes.

```
 Router.map(function()
{
this.route('about', { path: '/about' });
 this.route('favorites', { path: '/favs' });
}
);
```

Now, when the user visits /about, Ember will render the about template. Visiting /favs will render the favorites template

Routing(contd…)

- URL redirection or forwarding mechanism, that makes a web page available for more than one URL address. Ember.js defines a transitionTo() method moves the application into another route and it behaves like link-to helper.
- To redirect from one route to another route, define the beforeModel hook into the route handler.

Syntax

```
Ember.Route.extend({
 beforeModel()
 {
this.transitionTo('routeToName');
 }
});
```

Routing(contd…)

ReplaceWith vs Transition

- Transition into another route while replacing the current URL, if possible.
  This will replace the current history entry instead of adding a new one. Beside that, it is identical to transitionTo in all other respects.

```
beforeModel()
{
this.replaceWith('rentals‘
);
 }
```

```
this.transitionTo('blogPost', 1,
 { queryParams: {
showComments: 'true'
}
});
```

Models

- Model is a class that extends the functionality of the Ember Data.
- When a user refreshes the page, the contents of page should be represented by a model.
- In Ember.js, every route has an associated model. The model helps to improve application robustness and performance.
- The Ember Data manipulates the stored data in the server and also works easily with streaming APIs like socket.io and Firebase or WebSockets.
- Core Concepts
  - Store
  - Models
  - Records
  - Adapter
  - Caching

Models(contd…)

Store

- The store is a central repository and cache of all records available in an application. The route and controllers can access the stored data of your application.
- The DS.Store is created automatically to share the data among the entire object.

```
import Ember from 'ember';
 export default Ember.Route.extend({
model() {
return this.store.find();
 } });
```

Models

- Model is a class that extends the functionality of the Ember Data, which specifies relationships with other objects.
- When a user refreshes the page, the contents of page should be represented by a model.
- import DS from 'ember-data';
- export default DS.Model.extend({

Models(contd…)

- owner: DS.attr(),
- city: DS.attr() });

Records

- A record is an instance of a model that includes the information, which is loaded from a server and you can identify the record by its model *type* and *ID*.

```
//It finds the record of type 'person' and an 'ID' of 1 this.get('store').findRecord('person', 1); // => { id: 1,
name: 'steve-buscemi' }
```

Adapter

- An adapter is an object that is responsible for translating requested records from Ember into appropriate calls to particular server backend.
- For instance, if you want to find a person with ID of 1, then Ember will load the URL by using HTTP as */person/1*.

Models(contd…)

- Defining Model

```
import DS from 'ember-data';
export default
DS.Model.extend({ bookName:
DS.attr(), authorName:
DS.attr() });
```

Push and remove

```
import Ember from 'ember';
export default Ember.Route.extend({
model() {
 //model will display these records when you
execute the code
return [{ id: 1, name: 'Category One' },
 { id: 2, name: 'Category Two' }]; },
actions: { //it adds records to model
addNewCategory(id, name) {
this.controller.get('model').pushObject({id,na
me});
}, //it removes the records from model
deleteCategory(category) {
this.controller.get('model').removeObject(cat
egory); } }
 });
```

Debugging using Ember inspector

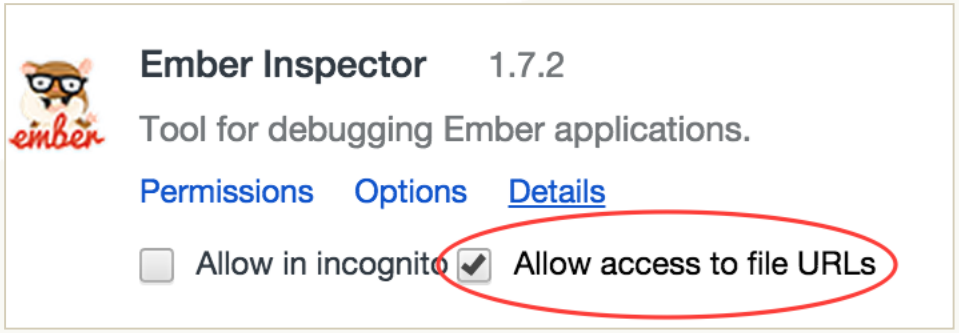
- The Ember Inspector is a browser add-on designed to help you understand and debug your Ember applications.
- You can install the Inspector on Google Chrome, Firefox, other browsers (via a bookmarklet), and on mobile devices.

CHROME

- You can install the Inspector on Google Chrome as a new Developer Tool. To begin, visit the Extension page on the Chrome Web Store.
- Click on "Add To Chrome“:

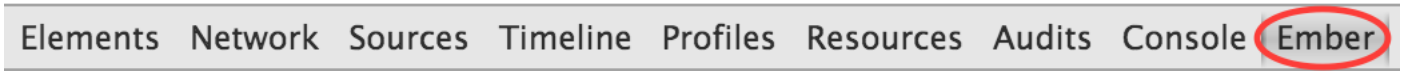
Debugging using Ember inspector(contd…)

- **ENABLE TOMSTER**
- You can configure a Tomster icon to show up in Chrome's URL bar whenever you are visiting a site that uses Ember.
- Visit chrome://extensions, then click on Options.
- Make sure the "Display the Tomster" checkbox is checked.

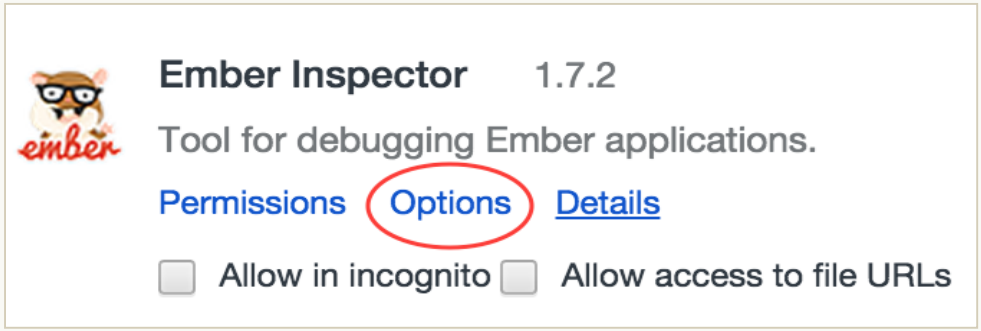


Debugging using Ember inspector(contd…)

- Once installed, go to an Ember application, open the Developer Tools, and click on the Ember tab at the far right.



- **FILE:// PROTOCOL**
- To use the Inspector with the file:// protocol, visit chrome://extensions in Chrome and check the "Allow access to file URLs" checkbox:



Ember - Helper

- **{{outlet}}** -> This will provide a stub/hook/point into which you can render Components(Controller + View). One would use this with the render method of routes. This would render the DetailsController with DetailsView into the outlet 'detailsOutlet' of the index template.

- **{{yield}}** -> Denotes an area of a template that will be rendered inside of another template

- **{{render}}** -> Renders the NavigationController and NavigationView at this place. This is helper is good for places, where the Controller and View do not change, e.g. a navigation.

Ember - Helpers

- Allow you to add **additional functionality** to your templates beyond what is included out-of-the-box in Ember

- Helper Names:
  - Unlike components, which require a dash in the name to follow the Custom Element spec, helper names can be single or multi-word. If your helper's name is multi-word, it should be dasherized.

Ember - Helper

- **{{partial}}** -> The partial helper renders another template without changing the template context:

```
{{foo}}
{{partial "nav"}}
```

- The above example template will render a template named "nav", which has the same context as the parent template it's rendered into, so if the "nav" template also referenced {{foo}}, it would print the same thing as the {{foo}} in the above example.

- If a "_nav" template isn't found, the partial helper will fall back to a template named "nav".

Component

- Components must have at least one dash in their name.
- Each component, under the hood, is backed by an element. By default Ember will use **a element** to contain your component's template.

- If you need to customize the behavior of the component you'll need to define a subclass of **Ember.Component**.

- Dynamically rendering a component
  - {{component ‘my-component'}}

Component LifeCycle

- Components are rendered, re-rendered and finally removed, Ember provides *lifecycle hooks* that allow you to run code at specific times in a component's life.

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

On Component Destroy

1. willDestroyElement
2. willClearRender
3. didDestroyElement

Controller

- Controllers behave like a specialized type of Component that is rendered by the router when entering a Route.

- The controller receives a single property from the Route – model – which is the return value of the Route's **model()** method.

Services

- An **Ember.Service** is an Ember object that lives for the duration of the application, and can be made available in different parts of your application.
- Useful for features that require shared state or persistent connections. Example uses of services might include:
  - User/session authentication.
  - Geolocation.
  - WebSockets.
  - Server-sent events or notifications.
  - Server-backed API calls that may not fit Ember Data.
  - Third-party APIs.
  - Logging.

Services

- Defining Services

```
ember generate service mystore
```

- Accessing Services
  - Ember.inject.service

```
import Ember from 'ember';
export default Ember.Component.extend({
//will load the service in file /app/services/shopping-cart.js
shoppingCart: Ember.inject.service()
});
```

Model

- Models tend to be *persistent*.
- Ember Data:
  - Each model is represented by a subclass of Model that defines the ***attributes, relationships, and behavior*** of the data that you present to the user.

```
import DS from 'ember-data';

export default DS.Model.extend({
  firstName: DS.attr('string'),
  birthday:  DS.attr('date')
});
```

- Records:
  - Is an instance of a model that contains data loaded from a server. Your application can also create new records and save them back to the server.

```
this.get('store').findRecord('person', 1); // => { id: 1, name: 'steve-buscemi' }
```

Adapter and Caching

- **Adapter:**
  - An **adapter** is an object that translates requests from Ember (such as "find the user with an ID of 1") into requests to a server.

- **Caching:**
  - The store will automatically cache records for you. If a record had already been loaded, asking for it a second time will always return the same object instance. This minimizes the number of round-trips to the server, and allows your application to render its UI to the user as fast as possible.

Initializers

- To configure your application as it boots.
- Two types of initializers:
  - application initializers
  - application instance initializers.

- ***Application initializers*** are run as your application boots, and provide the primary means to configure dependency injections in your application.

- ***Application instance initializers*** are run as an application instance is loaded. They provide a way to configure the initial state of your application, as well as to set up dependency injections that are local to the application instance (e.g. A/B testing configurations).

ember generate initializer shopping-cart

ember generate instance-initializer logger

Ember Addons

- Ember has a rich ecosystem of addons that can be easily added to projects.
- Addons provide a wide range of functionality to projects, often saving time and letting you focus on your project.
- To browse addons, visit the Ember Observer website.
- It catalogs and categorizes ember addons that have been published to NPM and assigns them a score based on a variety of criteria.
- Run the following command to install the addon:
  - ember install ember-cli-tutorial-style
- For more info Ember Add-ons refer below link:

  https://gist.github.com/kristianmandrup/ae3174217f68a6a51ed5

Ember Popup

- ember-modal-dialog
- ember-dialog

- http://yapplabs.github.io/ember-modal-dialog/

References

- https://guides.emberjs.com/v2.12.0/
- https://code-maven.com/introduction-to-handlebars-javascript-templating-system
- http://builtwithember.io/
- http://ember-animation.github.io/liquid-fire/

Recap

Twowaybinding

ember data

handlebar

Emberdata

LiquidFire

Model

Thank You For Your Time

People matter, results count.