Ember.js

A framework for creating ambitious web applications.

Introduction

Ember is a JavaScript front-end framework designed to help you build websites with rich and complex user interactions. It does so by providing developers both with many features that are essential to manage complexity in modern web applications, as well as an integrated development toolkit that enables rapid iteration.

- *Ember.js is an open source JavaScript framework under MIT license.*
- *It provides the new binding syntax using the HTMLBars template engine which is a superset of the Handerlbars templating engine.*
- *It provides the Glimmer rendering engine to increase the rendering speed.*
- *It provides the Command Line Interface utility that integrates Ember patterns into development process and focuses easily on the developer productivity.*
- *It supports data binding to create the link between two properties and when one property changes, the other property will get upgraded with the new value*

Who's using Ember.js ?

SKYLIGHT

Square

code school

LinkedIn

zendesk

ibotta

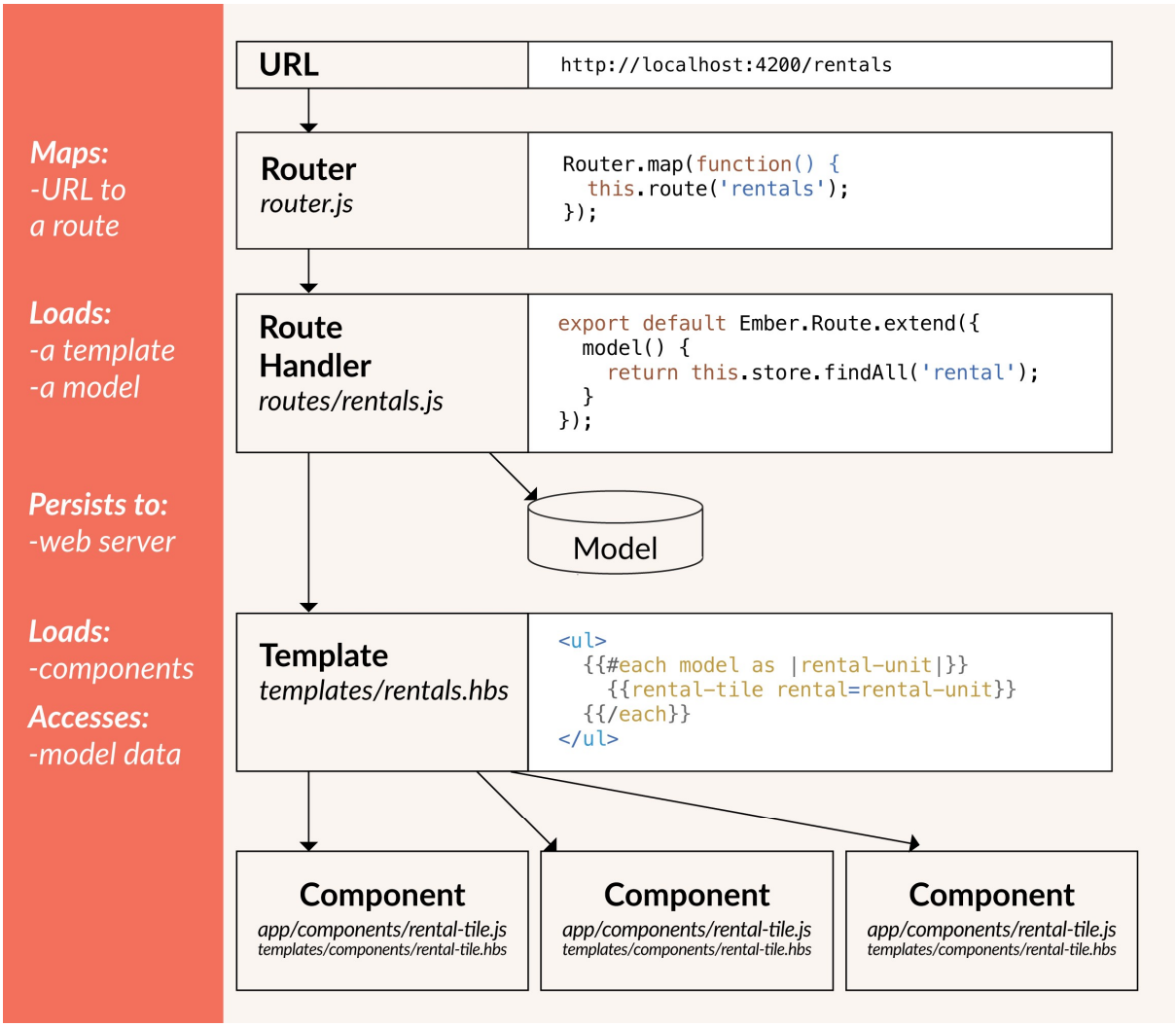
Microsoft

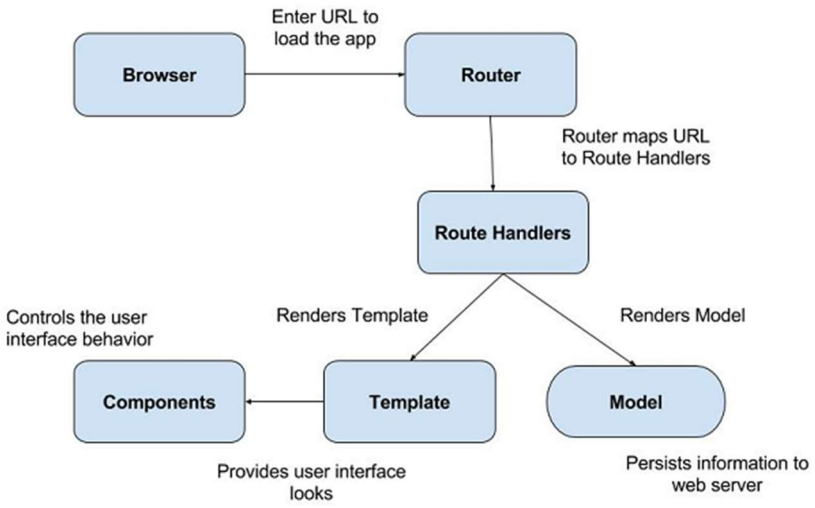
Blue Apron

NETFLIX

INTERCOM

Working of Ember Application





Some important features of Ember.js are

1. Routers & Route Handlers

- The URL loads the app by entering the URL in the address bar and user will click a link within the app. The Ember uses the router to map the URL to a route handler. The router matches the existing URL to the route which is then used for loading data, displaying the templates and setting up an application state.
- The Route handler performs the following actions −
  - It provides the template.
  - It defines the model that will be accessible to the template.
- If there is no permission for the user to visit a particular part of the app, then the router will redirect to a new route.

2. 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:
  `<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:
  `<div>Hi {{name}}, this is a valid Ember template!</div>`
- Here, `{{name}}` is a property provided by the template's context.
- Besides properties, double curly braces `({{}})` may also contain helpers and components, which we'll discuss later.

3. Models

- Models represent persistent state.
- For example, a property rentals application would want to save the details of a rental when a user publishes it, and so a rental would have a model defining its details, perhaps called the rental model.
- A model typically persists information to a web server, although models can be configured to save to anywhere else, such as the browser's Local Storage.

4. Object Model

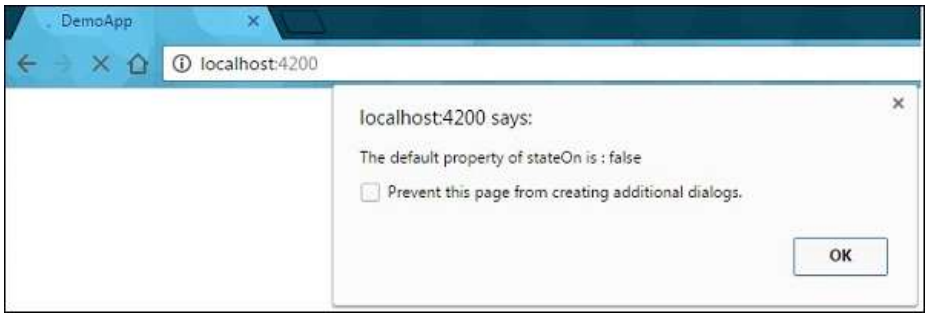
- In Ember.js, all objects are derived from the Ember.Object. Object-oriented analysis and design technique is called **object modeling**.
- Ember uses the Ember.Enumerable interface to extend the JavaScript Array prototype to give the observation changes for arrays and also uses the formatting and localization methods to extend the *String prototype*.
- The following table lists down the different types of object model in Ember.js along with their description –

- ***Classes and Instances*** → Class is a template or blue print, that has a collection of variables and functions, whereas instances are related to the object of that class. You can create new Ember class by using the Ember.Object's extend() method.

```
import Ember from 'ember';   //import ember module
export default function() {

   //new ember object
   const Demo = Ember.Object.extend ({
      init() {
         alert('The default property of stateOn is : '
+ this.get('stateOn'));
      },
      stateOn: false
   });

   const state = Demo.create();   //new instance from
object with create() method
   state.set('stateOn', true);
   console.log(state.get('stateOn'));
}
```



- ***Computed Properties*** → A computed property declares functions as properties and Ember.js automatically calls the computed properties when needed and combines one or more properties in one variable.

```
import Ember from 'ember';
export default function() {
   var Car = Ember.Object.extend ({
      //The values for below variables will be
supplied by 'create' method
      CarName: null,
      CarModel: null,
      carDetails: Ember.computed('CarName',
'CarModel', function() {
         //returns values to the computed property
function 'carDetails'
         return ' Car Name: ' + this.get('CarName') +
'<br>' +
            ' Car Model: ' + this.get('CarModel');
      })
   });
```

```
    var mycar = Car.create ({
        //initializing the values of Car variables
        CarName: "Alto",
        CarModel: "800",
    });
    //Displaying the information of the car
    document.write("<h2>Details of the car:
<br></h2>");
    document.write(mycar.get('carDetails'));
}
```

- ***Bindings*** → The binding is a powerful feature of Ember.js which helps to create a link between two properties and if one of the properties gets changed, the other one is updated automatically.

```
import Ember from 'ember';

export default function() {
    var CarOne = Ember.Object.create ({
        //primary value
        TotalPrice: 860600
    });

    var Car = Ember.Object.extend ({
        //creates property which is an alias for another
property
        TotalPrice:
Ember.computed.alias('CarOne.TotalPrice')
    });

    var CarTwo = Car.create ({
        CarOne: CarOne
    });
    document.write('Value of car before updating: ' +
CarTwo.get('TotalPrice'));

    //sets the car price
    CarTwo.set('TotalPrice', 930000);

    //above car price effects the CarOne
    document.write('<br>Value of car after updating: '
+ CarOne.get('TotalPrice'));
```

```
}
```

5. Managing Dependencies

- Ember uses NPM and Bower for managing dependencies which are defined in package.json for NPM and *bower.json* for Bower. For instance, you may require installing SASS for your style sheets which is not installed by Ember while developing Ember app.
- ***Addons*** → The *ember install command* will save all the dependencies to the respective configuration file.
- ***Bower*** → It basically maintains and monitors all packages and examines new updates. It uses the configuration file *bower.json*to keep track of applications placed at the root of the Ember CLI project.
  `bower install <dependencies> --save`
- ***Environment Specific Assets*** → The different assets can be used in different environments by defining object as first parameter which is an environment name and the value of an object should be used as asset in that environment. In the *ember-cli-build.js*manifest file, you can include as –

```
app.import ({
   development: 'bower_components/ember/ember.js',
   production:  'bower_components/ember/ember.prod.js'
});
```

6. Application Concerns

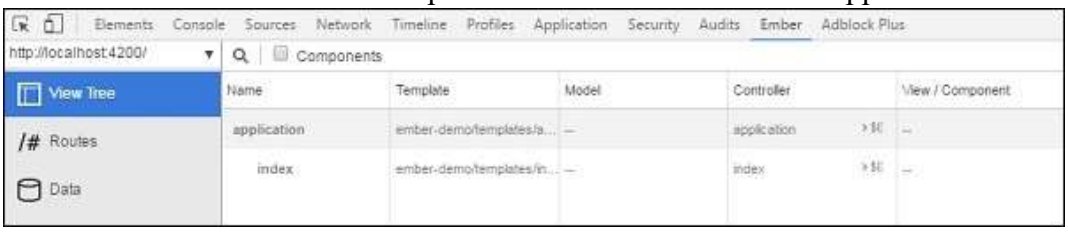
- Application creates the *Ember.ApplicationInstance* class while running which is used for managing its aspects and it acts as owner for instantiated objects. In short, the *Ember.Application* class defines the application and the *Ember.ApplicationInstance class* manages its state.

7. Configuring Ember.js

- Configuring App and Ember CLI → You can configure the Ember App and CLI for managing the application's environment.
- Disabling Prototype Extensions and Specifying URL Type → The prototype extensions can be disabled by setting the EXTEND_PROTOTYPES flag to false and specifying the URL type by using the Ember router options.

8. Ember Inspector

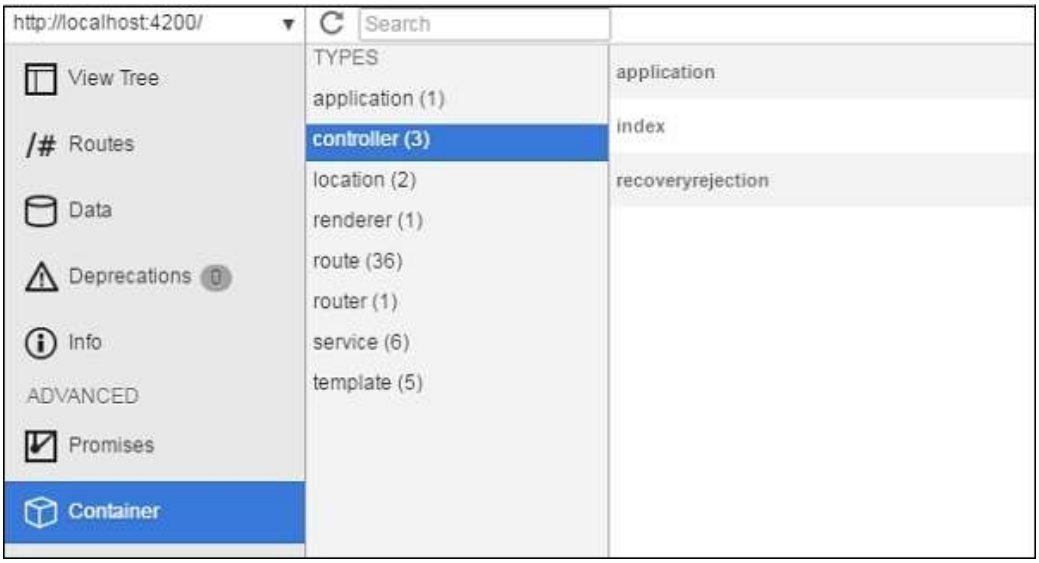
- Ember inspector is a browser add-on which is used to debug the Ember applications.
- ***The View Tree*** → The view tree provides the current state of an application.



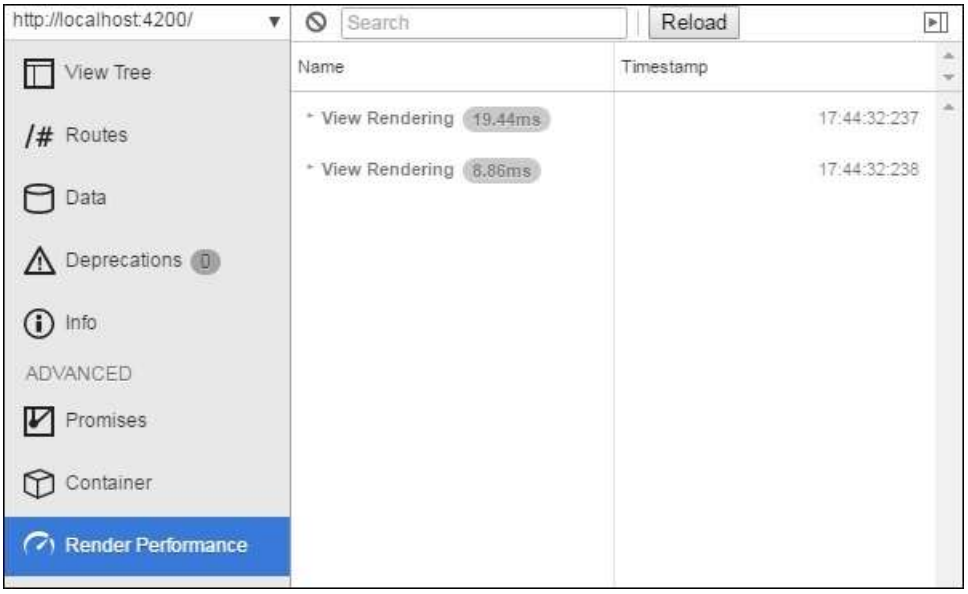
- ***Inspecting Routes, Data Tab and Library Info*** → You can see the list of application's routes defined by the inspector and the Data tab is used to display the list of model types.

9. Inspecting Objects and Rendering Performance

- In the Ember application, object instances are maintained by the container for inspecting the objects. You can inspect the objects by using the Container tab as shown below –



- Ember inspector allows you to compute the application's render time by using the Render Performance tab.



10. Debugging Promises

- Ember inspector provides promises based on their states such as Fulfilled, Pending and Rejected. Click on the Promises tab and you will see the list of Promises with the specified state.
- You can trace the promises by using the Trace promises option. By default, this option is disabled; you can enable it by checking the Trace promises checkbox as shown below –

WEB TECHNOLOGIES 1 (UE16CS204) ASSIGNMENT
AKASH N
01FB16ECS041
3RD A, CSE
PESU