# Title: Frontend Overview

The purpose of this document is to walkthrough the fundamentals of Ember application development and is primarily intended for developers who are new to ember to get them started.

# **1. Ember Object Fundamentals**

**EmberObject Class**

Base class for all ember object classes.

* Data binding related functionalities. Once the object changes, the change has to be propagated to the user interface

**Create a Ember class**

By subclassing EmberObject class

let HumanClass = EmberObject.extend({  
name,  
age,  
speak() {

}  
});

**Creating an object**

let human = HumanClass.create({  
name: 'Joe',  
age: 23  
});  
human.speak();

**Sharing state across all instances: (Arrays and Objects)**

Note **shoppingList** in the below class Person

import { A } from '@ember/array';  
import EmberObject from '@ember/object';

const Person = EmberObject.extend({  
shoppingList: A(['eggs', 'cheese'])  
});

Person.create({  
name: 'Stefan Penner',  
addItem() {  
this.shoppingList.pushObject('bacon');  
}  
});

Person.create({  
name: 'Robert Jackson',  
addItem() {  
this.shoppingList.pushObject('sausage');  
}  
});

// Stefan and Robert both trigger their addItem.  
// They both end up with: ['eggs', 'cheese', 'bacon', 'sausage']

To avoid this behavior, set it as an instance property like below,

const Person = EmberObject.extend({  
init() {  
this.set('shoppingList', A(['eggs', 'cheese']));  
}  
});

**Setters and Getters**  
set, setProperties  
get, getProperties

**Reopening classes and instances**

Person.reopen({  
isPerson: true  
});

**Difference between reopen and extend**

Extend lets you to subclass where as reopen lets you to add props and methods to existing class (basically modifying the original class)

**reopen vs reopenClass**

**reopen -** at instance level  
**reopenClass -** at class level for static properties

**Computed Properties**

Person = EmberObject.extend({  
// these will be supplied by `create`  
firstName: null,  
lastName: null,

fullName: computed('firstName', 'lastName', function() {  
return `${this.firstName} ${this.lastName}`;  
})  
});

Computed properties only run when they are accessed and **not every time dependency changes**

Computed properties are meant to behave like pure functions without causing any side effects

Things to note while using computed properties

* Make sure all of the instance level properties used inside a computed property function is added as a dependency on the top
* Make sure that appropriate notation is used on the dependency added
* Make sure setters are configured if the computed property will at any point be set by the business logic

How to write dependencies for computed properties?

* In case of dependent props being objects, target specific dependent properties inside an object and not the whole object as such.

address: computed('location.{streetName,streetNumber}', function() {  
return `${this.location.streetNumber} ${this.location.streetName}`;  
})

* In case of arrays, make sure the right notation is used and right property is observed

computed('todos.[]', function() {  
});

computed('todos.@each.isDone', function() {  
});

**Setting computed properties**

fullName: computed('firstName', 'lastName', {  
get(key) {  
return `${this.firstName} ${this.lastName}`;  
},  
set(key, value) {  
let [firstName, lastName] = value.split(/\s+/);  
setProperties(this, {  
firstName,  
lastName  
});  
return value;  
}  
})

**Macros => Shorthand versions for lesser code**

hasAccess: computed.or('isAdmin', 'isOwner')

**Important macros:**

**computed.alias()** => two way binding  
**computed.readOnly()** => one way binding  
**computed.oneWay()** => hybrid (accepts changes from parent, but once changed, changes will not propagate to the parent and at the same time will stop listening to changes from parent)

![](data:text/html;base64,)

and many more ...

<https://api.emberjs.com/ember/3.11/modules/@ember%2Fobject>

**Observers:**

Triggers a function every time the observed property(s) changes.

fullNameChanged: observer('fullName', function() {  
// deal with the change  
console.log(`fullName changed to: ${this.fullName}`);  
})

**EmberArray**

  new Ember.A([2,4])

  Subclass of MutableArray and an extension of native javascript array with additional convenience methods.

**Why EmberArray?**  
  
    For ember to keep track of changes made to the array

![](data:text/html;base64,)

# **2. Ember Application Flow and Basic building blocks**

The application flow

![](data:text/html;base64,)

**Important Gotcha!**

![](data:text/html;base64,)

**Building Blocks:**

1. **Routes**
2. **Controllers**
3. **Templates**
4. **Components**
5. **Helpers**

**Routes:**

Entities that are url driven that acts as trigger to initiate the whole process of loading a page

**Application route** => Responsible for booting up the application

There are child routes configured by the **router.js**

**Route configuration:**

const Router = EmberRouter.extend({  
// ... router config related options such location, rootURL etc...  
});

Router.map(function() {  
// define routes and nested routes

// Home route  
this.route('dashboard', {  
path: '/'  
});

// Other routes

// Route 1  
this.route('custom-modules', {  
path: '/custom\_module/:module\_name',  
resetNamespace: true  
}, function() {

// Sub routes of Route 1  
this.route('filters', {  
path: '/view/:segment\_id'  
});

});

// Route 2  
..  
// Route 3  
..  
..  
// Route N  
});

A typical route configuration looks like below,

![](data:text/html;base64,)

**Important lifecycle hooks of a Router:**

**Starting phase:**

1. **willTransition**() -> best place to stop user from navigating away from current page eg: unsaved changes in a form

**Validation phase:**  
           2. **beforeModel**()  -> best place to make some checks and determine if user has to be redirected elsewhere before even the model is resolved for the current route  
           3. **model**()  -> collect data for the current route  
           4. **afterModel**()  -> based on the resolved model data, checks can be made to determine if the user has to be redirected elsewhere  
    5. **redirect**()  -> very similar to afterModel, but one diff is that once we reach to this stage, the route is termed as active and hence any subsequent redirections to any of its child route will not retrigger beforeModel, model and afterModel of the parent route

**Setup phase:**  
 6. **activate**()  -> triggers for the first time the route is entered and not on model changes => best place to put analytics  
 7. **deActivate**()  -> triggers when the route is exited and not on model changes  
          8. **renderTemplate**() -> Can be used to load a custom template and a custom controller on our own and not as per conventions, but DON'T use this hook. Try to avoid as much as possible  
          9. **setupController()** -> controller gets instantiated and can be used to set properties on the controller  
         10. **resetController()** -> reset controller properties

**Finishing phase:**  
         11. **didTransition()** -> once the transition is complete

**Controller**

     An entity that receives the resolved model as a property from the route which can then be used in the route's template. A controller implementation is not **mandatory!** and is completely optional. When no controller is created, ember creates a default one on its own.

**When to use controller?**

* Share actions and attributes across all child components on a page
* Define query params which when changed will trigger a route model's hook to resolved again.

![](data:text/html;base64,)

**Templates**

![](data:text/html;base64,)

**Conditional rendering:**

{{#if person}}  
Welcome back, <b>{{this.person.firstName}} {{this.person.lastName}}</b>!  
{{/if}}

* Use computed properties when there are multiple conditions to evaluate

**{{#if (or isAdmin isOwner)}}**  
**Congrats! you have access to this snippet**  
**{{/if}}**

Above can be written as

  hasAccess: computed.or('isAdmin', 'isOwner')

{{#if hasAccess}}  
Congrats! you have access to this snippet  
{{/if}}

**Iteration**

<ul>  
{{#each people as |person|}}  
<li>Hello, {{[person.name](http://person.name)}}!</li>  
{{/each}}  
</ul>

**Helpers**

Helpers are a way to use javascript logic into the templates.

* They don't maintain state
* They are pure functions
* They shouldn't cause any side effects

**How to create a helper?**

import { helper as buildHelper } from '@ember/component/helper';

export function sum(params) {  
return params[0] + params[1]  
};

export const helper = buildHelper(sum);

**How to use a helper?**       
<p>Total: {{sum 1 2}}</p>

**Nested helpers:**

{{sum (multiply 2 4) 2}}

**Built in helpers:**

<https://api.emberjs.com/ember/3.11/classes/Ember.Templates.helpers/>

**Components:**

    Capsule that holds presentation, state and behavior of an object.

* presentation => template.hbs
* state and behavior  => component.js

**Why use components?**

* Improves **modularity** (break down a complex problem into multiple sub problems)
* Improves **code** **reusability**
* **Independent** and can be ported anywhere within the application, (depends on how **loosely coupled** the component is, avoid parent references) and can be plugged in anywhere.

**Example:**  
{{BlogPost title=title body=body }}

**Code:**

export default Ember.Component.extend({  
name,  
someMethod() {  
...  
},  
actions: {  
speak() {  
...  
}  
}  
});

**Component lifecycle**

1. **Initial Render**
   * init
   * didRecieveAttrs
   * didInsertElement
2. **Re-Render**
   * didUpdateAttrs
   * didRecieveAttrs
   * didInsertElement
3. **Destroy**
   * willDestroyElement

**Block form of components and data sharing:**

1. **Simple usecase:**

**templates/blogs.hbs (Parent)**

{{#BlogPost title=title}}  
<p>by {{this.author}}</p>  
<p class="my-own-class">{{this.body}}</p>  
{{/BlogPost}}

**blog-post/component.js (Child)**

<h2>{{this.title}}</h2>  
<div class="body">  
{{yield}}  
</div>

1. **Complex usecase:**

**templates/blogs.hbs (Parent)**

{{#BlogPost editStyle="markdown-style" content=postContent as |post|}}  
<p class="author">by {{this.author}}</p>  
{{post.body}}  
{{/BlogPost}}

**blog-post/component.js (Child)**

<h2>{{this.title}}</h2>  
<div class="body">  
{{yield (hash body=(component this.editStyle content=this.content))}}  
</div>

**Component can be both block-form as well simple type**

{{if has-block}}

# **3. Ember Data**

Ember Data - central repository, global state management

**Why Ember Data?**

* Avoids repetitive querying of server for same data from multiple parts of the application
* Avoids data inconsistencies between different parts of the same application
* Becomes easier to fix changes in APIs at one single place (model), rather than fixing it in multiple places

**Model:**  
    A model is a representation of your data in the Ember application. A typical model looks like below,

![](data:text/html;base64,)

**Adapters:**

    Adapters act as a bridge between the server APIs and Ember application. Whenever store gets a request for a record, or when record has to be saved by the store, the store contacts the adapter for it, which in turns talks to the server.

**What all can an adapter do?**

* Determine what endpoints it would hit on request of a particular resource
* Determine to serve cached version or fetch new record from server or do both while sending cached version first or hold until server version is resolved
* Determine what serializer to use
* Determine what headers to use for the request
* Determine how the requested model should be transformed as per backend's specification

When the requested record is not available in the store, below is the flow

![](data:text/html;base64,)![](data:text/html;base64,)

However if the requested data is available in the store, then below is the flow

![](data:text/html;base64,)

**Store actions:**

**Fetch:**

store.findRecord('book', 2);    // returns promise - network request is made  
store.findAll('book')      // returns a promise that will be resolved with all books - network request is made

store.peekRecord('book', 2);     // returns book with id 2  
store.peekAll('book')        //    returns all books

store.query('book', { author: 'David' });    // returns all books written by David  
store.queryRecord('current\_book', {});   // returns the currently opened book

**difference between queryRecord and findRecord?**  
use queryRecord in scenarios where id is not known before hand and only one record is expected to be returned

**Create:**

let book = store.createRecord('book', {  
author: 'David',  
name: 'Life is beautiful'  
});

**Update:**

set(book, 'author', 'Michael');    //    updates the model but it is not persistent yet  
To make it persistent, do  
book.save()

**hasDirtyAttributes:**  
Flag to tell if the model has any unsaved changes  
**changedAttributes():**  
Provides list of attributes that has undergone changes  
{ author: ['David', 'Michael'] }  
**rollbackAttributes():**  
rolls back unsaved changes to the previously committed state

**Delete:**  
book.deleteRecord()    //    marks the record as deleted but doesn't persist. Flag used to marking it as deleted is 'isDeleted'  
book.save()    // This persists the deletion

or we can use book.destroyRecord()    //    to do both at one shot

**Relationships:**

**one to one**

![](data:text/html;base64,)

**one to many**

![](data:text/html;base64,)

**many to many**

![](data:text/html;base64,)

**Polymorphic relationships:**

When a relationship model can be of multiple types, polymorphic relationships can be used

app/models/person.js

import DS from 'ember-data';

export default DS.Model.extend({  
vehicles: DS.hasMany('vehicle', { polymorphic: true })  
});

app/models/vehicle.js

import DS from 'ember-data';

export default DS.Model.extend({  
driver: DS.belongsTo('person'),  
});

app/models/vehicle-bike.js

import { computed } from '@ember/object';  
import Vehicle from './vehicle';

export default Vehicle.extend({  
handlebarType: DS.attr('string')  
});

app/models/vehicle-car.js

import { computed } from '@ember/object';  
import Vehicle from './vehicle';

export default Vehicle.extend({  
isFWD: DS.attr('string')  
});

**Creating records along with establishing relationships:**

let myBook = this.store.peekRecord('book', 1);  
let author = this.store.createRecord('author', {  
book: myBook, // this links this author to the book 1  
name: 'David'  
});  
author.save();

now get(myBook, '[author.name](http://author.name)') => 'David' as Ember automatically establishes that relationship

**Retrieving records with relationships**

this.store.findRecord('author', 2 , {include: 'books'});

**How are relationships side-loaded?**

{  
"users": [  
{  
"id": 1,  
"display\_name": "Sales Agent",  
"email": "umashankar.masilamani@[freshworks.com](http://freshworks.com)",  
"is\_active": true,  
"work\_number": "+919999999999",  
"mobile\_number": null  
}  
],  
"contacts": [],  
"sales\_accounts": [],  
"deal\_pipelines": [  
{  
"partial": true,  
"id": 1,  
"name": "Default Pipeline",  
"position": 1,  
"is\_default": true,  
"rotting\_days": 30  
}  
],  
"deal\_stages": [  
{  
"partial": true,  
"id": 3,  
"name": "Under review",  
"position": 3,  
"forecast\_type": "Open",  
"updated\_at": "2019-09-30T17:28:23+05:30",  
"deal\_pipeline\_id": 1,  
"choice\_type": 2,  
"probability": 100  
}  
],  
"deal\_types": [],  
"territories": [],  
"currencies": [  
{  
"partial": false,  
"id": 1,  
"is\_active": true,  
"currency\_code": "USD",  
"exchange\_rate": "1.0",  
"currency\_type": 1,  
"schedule\_info": null,  
"rate\_change\_ids": []  
}  
],  
"rate\_changes": [],  
"appointments": [],  
"tasks": [],  
"deals": [  
{  
"id": 14,  
"name": "deal18",  
"amount": "180.0",  
"base\_currency\_amount": "180.0",  
"expected\_close": null,  
"closed\_date": null,  
"stage\_updated\_time": "2019-10-23T20:28:58+05:30",  
"custom\_field": {},  
"probability": 100,  
"updated\_at": "2019-10-23T20:28:58+05:30",  
"created\_at": "2019-10-23T20:11:08+05:30",  
"deal\_pipeline\_id": 1,  
"deal\_stage\_id": 3,  
"age": 0,  
"links": {  
"conversations": "/deals/14/conversations/all?include=email\_conversation\_recipients%2Ctargetable%2Cphone\_number%2Cphone\_caller%2Cnote%2Cuser&per\_page=3",  
"document\_associations": "/deals/14/document\_associations",  
"notes": "/deals/14/notes?include=creater",  
"tasks": "/deals/14/tasks?include=creater,owner,updater,targetable,users,task\_type",  
"appointments": "/deals/14/appointments?include=creater,owner,updater,targetable,appointment\_attendees"  
},  
"recent\_note": null,  
"completed\_sales\_sequences": null,  
"active\_sales\_sequences": null,  
"web\_form\_id": null,  
"upcoming\_activities\_time": null,  
"collaboration": {},  
"last\_assigned\_at": "2019-10-23T20:11:09+05:30",  
"tags": [],  
"last\_contacted\_sales\_activity\_mode": null,  
"last\_contacted\_via\_sales\_activity": null,  
"expected\_deal\_value": "180.0",  
"deal\_freddy\_metrics": null,  
"is\_deleted": false,  
"team\_user\_ids": null,  
"rotten\_days": null,  
"owner\_id": 1,  
"creater\_id": 1,  
"updater\_id": 1,  
"contact\_ids": [],  
"sales\_account\_id": null,  
"deal\_type\_id": null,  
"territory\_id": null,  
"currency\_id": 1,  
"appointment\_ids": [],  
"task\_ids": [],  
"lookup\_information": {}  
}  
],  
"meta": {  
"total": 1,  
"total\_potential": null  
}  
}

**Removing relationships**

let book = this.store.peekRecord('book', 1);  
let author = this.store.peekRecord('author', 2);  
book.set('author', null);  
book.save();

**Sync vs Async relationships:**

**Sync:**  
Returns local cache of the data in the store, if not available returns null  
// app/models/post.js  
export default DS.Model.extend({  
comments: DS.hasMany('comment', { async: false })  
});

**Async:**  
Returns a promise that will resolve in future or immediately depending on store's cache of records

// app/models/post.js  
export default DS.Model.extend({  
comments: DS.hasMany('comment', { async: true })  
});

**Adapters In Detail:**

JSONAPIAdapter  
<https://api.emberjs.com/ember-data/release/classes/JSONAPIAdapter>

{  
"data": {  
"type": "articles",  
"id": "1",  
"attributes": {  
"title": "JSON:API paints my bikeshed!"  
},  
"links": {  
"self": "<http://example.com/articles/1>"  
},  
"relationships": {  
"comments": {  
"data": [  
{ "type": "comments", "id": "5" },  
{ "type": "comments", "id": "12" }  
]  
}  
}  
},  
"included": [{  
"type": "comments",  
"id": "5",  
"attributes": {  
"body": "First!"  
},  
"links": {  
"self": "<http://example.com/comments/5>"  
}  
}, {  
"type": "comments",  
"id": "12",  
"attributes": {  
"body": "I like XML better"  
},  
"links": {  
"self": "<http://example.com/comments/12>"  
}  
}]  
}

RESTAdapter  
<https://api.emberjs.com/ember-data/release/classes/RESTAdapter>

ActiveModelAdapter  
<https://api.emberjs.com/ember-data/1.13/classes/DS.ActiveModelAdapter>

{  
"users": [  
{  
"id": 1,  
"display\_name": "Sales Agent",  
"email": "umashankar.masilamani@[freshworks.com](http://freshworks.com)",  
"is\_active": true,  
"work\_number": "+919999999999",  
"mobile\_number": null  
}  
],  
"contacts": [],  
"sales\_accounts": [],  
"deal\_pipelines": [  
{  
"partial": true,  
"id": 1,  
"name": "Default Pipeline",  
"position": 1,  
"is\_default": true,  
"rotting\_days": 30  
}  
],  
"deal\_stages": [  
{  
"partial": true,  
"id": 3,  
"name": "Under review",  
"position": 3,  
"forecast\_type": "Open",  
"updated\_at": "2019-09-30T17:28:23+05:30",  
"deal\_pipeline\_id": 1,  
"choice\_type": 2,  
"probability": 100  
}  
],  
"deal\_types": [],  
"territories": [],  
"currencies": [  
{  
"partial": false,  
"id": 1,  
"is\_active": true,  
"currency\_code": "USD",  
"exchange\_rate": "1.0",  
"currency\_type": 1,  
"schedule\_info": null,  
"rate\_change\_ids": []  
}  
],  
"rate\_changes": [],  
"appointments": [],  
"tasks": [],  
"deals": [  
{  
"id": 14,  
"name": "deal18",  
"amount": "180.0",  
"base\_currency\_amount": "180.0",  
"expected\_close": null,  
"closed\_date": null,  
"stage\_updated\_time": "2019-10-23T20:28:58+05:30",  
"custom\_field": {},  
"probability": 100,  
"updated\_at": "2019-10-23T20:28:58+05:30",  
"created\_at": "2019-10-23T20:11:08+05:30",  
"deal\_pipeline\_id": 1,  
"deal\_stage\_id": 3,  
"age": 0,  
"links": {  
"conversations": "/deals/14/conversations/all?include=email\_conversation\_recipients%2Ctargetable%2Cphone\_number%2Cphone\_caller%2Cnote%2Cuser&per\_page=3",  
"document\_associations": "/deals/14/document\_associations",  
"notes": "/deals/14/notes?include=creater",  
"tasks": "/deals/14/tasks?include=creater,owner,updater,targetable,users,task\_type",  
"appointments": "/deals/14/appointments?include=creater,owner,updater,targetable,appointment\_attendees"  
},  
"recent\_note": null,  
"completed\_sales\_sequences": null,  
"active\_sales\_sequences": null,  
"web\_form\_id": null,  
"upcoming\_activities\_time": null,  
"collaboration": {},  
"last\_assigned\_at": "2019-10-23T20:11:09+05:30",  
"tags": [],  
"last\_contacted\_sales\_activity\_mode": null,  
"last\_contacted\_via\_sales\_activity": null,  
"expected\_deal\_value": "180.0",  
"deal\_freddy\_metrics": null,  
"is\_deleted": false,  
"team\_user\_ids": null,  
"rotten\_days": null,  
"owner\_id": 1,  
"creater\_id": 1,  
"updater\_id": 1,  
"contact\_ids": [],  
"sales\_account\_id": null,  
"deal\_type\_id": null,  
"territory\_id": null,  
"currency\_id": 1,  
"appointment\_ids": [],  
"task\_ids": [],  
"lookup\_information": {}  
}  
],  
"meta": {  
"total": 1,  
"total\_potential": null  
}  
}

Unlike the DS.RESTAdapter, async relationship keys must be the singular form of the relationship name, followed by "\_id" for DS.belongsTo relationships, or "\_ids" for DS.hasMany relationships.

**namespace** => [https://test.freshsales.io/**custom-namespace**/deals/1](http://test.freshsales.io/custom-namespace/deals/1)

eg:  [https://test.freshsales.io/](http://test.freshsales.io/custom-namespace/deals/1)modules/custom/deals/1        where namespace = modules/custom

**host => custom-host**/deals/1

eg: <https://test.freshsales.io/deals/1>      where host =  [https://test.freshsales.io](https://test.freshsales.io/deals/1)

**pathForType =>** customise how the model should be transformed when requesting. By default it is dasherized, but can be altered to have it as underscored or any other pattern

eg: /user-profile/1 will be the default and can be changed to /user\_profile/1

**Set custom headers:**  
init() {  
this.\_super(...arguments);

this.set('headers', {  
'API\_KEY': 'secret key',  
'ANOTHER\_HEADER': 'Some header value'  
});  
}

**defaultSerializer =>** Determine what serializer to use

**shouldBackgroundReload** =>  returns cached version of the record and if this fn returns true fetches record(s) from server, else doesn't do anything

* all
* individual record

**shouldReload**   =>  if this fn returns true, fetches record(s) from server, else gives cached version

* all
* individual record

**Serializers - data formatters:**

Serializers format data that is sent to and received from backend to client. If the json format doesn't conform to the ember application that we use such as

* Trimming extra fields that are completely irrelevant to the application but still the api sends it for other consumers
* Altering specific fields in the json response

**normalize** method => server to client (to format response received from server back to client)  
**serialize** method => client to server (to format data sent from client to server)

**Eg:**

What server sends:  
{  
"data": {  
"id": "1",  
"type": "product",  
"attributes": {  
"name": "My Product",  
"amount": 100,  
"currency": "SEK"  
}  
}  
}

What we need instead:  
{  
"data": {  
"id": "1",  
"type": "product",  
"attributes": {  
"name": "My Product",  
"cost": {  
"amount": 100,  
"currency": "SEK"  
}  
}  
}  
}

How the data can be altered?

import DS from 'ember-data';

export default DS.JSONAPISerializer.extend({  
serialize(snapshot, options) {  
let json = this.\_super(...arguments);

json.data.attributes.cost = {  
amount: json.data.attributes.amount,  
currency: json.data.attributes.currency  
};

delete json.data.attributes.amount;  
delete json.data.attributes.currency;

return json;  
},  
});

Again serialising it back to make the server happy

import DS from 'ember-data';

export default DS.JSONAPISerializer.extend({  
normalizeResponse(store, primaryModelClass, payload, id, requestType) {  
payload.data.attributes.amount = payload.data.attributes.cost.amount;  
payload.data.attributes.currency = payload.data.attributes.cost.currency;

delete payload.data.attributes.cost;

return this.\_super(...arguments);  
},  
});

**Transforms:**

  You can transform data assigned to your attributes if you are not happy with the way the api is implemented. Lets say the server returns amounts in cents, then a custom transform like below will let you handle amount always in dollars in ember app, while still maintaining amount in cents on the server side

![](data:text/html;base64,)

**Ember Inflectors:**

To teach ember on how to deal with pluralisation of certain words

import Inflector from 'ember-inflector';

export function initialize(/\* application \*/) {  
const inflector = Inflector.inflector;

// Tell the inflector that the plural of "campus" is "campuses"  
inflector.irregular('campus', 'campuses');

// Tell the inflector that the plural of "advice" is "advice"  
inflector.uncountable('advice');  
}

export default {  
name: 'custom-inflector-rules',  
initialize  
};

# **4. Writing Ember Tests**

Testing framework: QUnit

* Acceptance Test
* Integration Test
* Unit Test

QUnit ref: <https://api.qunitjs.com/config/QUnit.assert>  
Built-in-helpers: <https://github.com/emberjs/ember-test-helpers/blob/master/API.md#waituntil>  
HelpersFromAddon: <https://github.com/cibernox/ember-native-dom-helpers>  (additional: scrollTo and selectfiles)

* **Unit Tests**

Testing Utils:

import \* as ArrayUtils from 'frontend/utils/array-flatten';  
import { module, test } from 'qunit';  
module('Unit | Utility | array flatten');  
test('array flatten works nested arrays', function(assert) {  
let result = ArrayUtils.arrayFlatten([1, 2, [3, 4, [5, 6]]]);  
assert.deepEqual([1, 2, 3, 4, 5, 6], result, 'Expected output arrived1');  
});

**Other Unit test (Eg: Model)**  
app/models/some-thing.js

import EmberObject, { computed } from '@ember/object';

export default EmberObject.extend({  
foo: 'bar',

computedFoo: computed('foo', function() {  
const foo = this.get('foo');

return `computed ${foo}`;  
})  
});

import { moduleFor, test } from 'ember-qunit';

moduleFor('model:some-thing', 'Unit | some thing', {  
unit: true  
});

test('should correctly concat foo', function(assert) {  
const someThing = this.subject(); // test helper that handles object instantiation. you can also pass state {name: 'shankar'}  
someThing.set('foo', 'baz');

assert.equal(someThing.get('computedFoo'), 'computed baz');  
});

**Skip tests:**  
  test('should correctly concat foo', function(assert) should be replaced with  
  skip('should correctly concat foo', function(assert)

* **Integration Tests**

moduleForComponent('funnel-draggable-card', 'Integration | Component | funnel draggable card', {  
integration: true,  
beforeEach() {  
injectI18n(this);  
injectStore(this);  
injectSession(this);  
injectModuleCustomization(this);  
let store = get(this, 'store');  
store.pushPayload(deal);  
page.setContext(this);  
this.on('openDealSummaryModal', function() {});  
setProperties(this, {  
'deal': store.peekRecord('deal', 2),  
'nameCustomization': nameCustomization  
});  
},  
afterEach() {  
currentUser.abilities = []; // clearing up the abilities to ensure other tests are not affected  
}  
});

test('funnel-draggable-card is rendered with basic information', async function(assert) {  
let deal = get(this, 'deal');  
let { name, amount, '[salesAccount.name](http://salesAccount.name)': accountName } = getProperties(deal, ['name', 'amount', '[salesAccount.name](http://salesAccount.name)']);  
await this.render(hbs`{{funnel-draggable-card  
nameCustomization=nameCustomization  
deal=deal  
openDealSummaryModal=(action 'openDealSummaryModal')  
}}`);  
assert.expect(6);  
assert.equal(page.funnelCardExists, true, 'Funnel card has been rendered in the Dom');  
assert.equal(page.cardTitleText, name);  
assert.equal(Number(page.dealAmountText.replace(/(\,|\$)/g, '')), Number(amount)); // eslint-disable-line no-useless-escape  
assert.equal(page.taskHandleExists, true, 'Task icon is rendered in funnel card');  
assert.equal(find(page.salesAccountTag).textContent.trim(), accountName, 'account name is rendered properly');  
assert.equal(findAll(page.expectedCloseTag).length, 1, 'expected close date exists');  
});

* **Acceptance Tests**

moduleForAcceptance('Acceptance - End to end | Deals funnel page', {  
beforeEach() {  
localStorage.removeItem('dealsModifiedFiltersRuleV1');  
localStorage.setItem('1\_dealsSegment', '25');  
}  
});  
test('Test if deals funnel page is redirected correctly', async function(assert) {  
await visit('/deals');  
assert.equal(currentURL(), '/deals/view/25/funnel?pipeline\_id=1');  
});  
test('if owner dropdown is loaded initially with all users', async function(assert) {  
await visit('/deals');  
await clickTrigger(page.ownerDropdown);  
assert.equal(jQuery(page.ownerDropdownOptions).length, 9);  
});

# **5. Ember Concurrency Task**

* Tasks, unlike Promises, support cancelation.
* Tasks expose their underlying state (whether they're running or idle) which makes it trivial to build loading indicators without having to manually track / mutate state yourself.
* Task Modifiers make it trivial to prevent two executions of the same task from running at the same time, e.g. you can prevent double form submissions using the .drop() modifier, or you can configure a task to be .restartable() so that it starts over when you click a "Restart" button. Implementing this logic without tasks requires a lot of boilerplate code and defensive programming.
* Tasks that live on Components are automatically canceled when that Component is unrendered; no more if(this.isDestroyed) checks to prevent timers or ajax responses causing "set on destroyed object" errors.

task(function \*(model) {  
yield get(this, 'customStore').getRequest('.........')  
})

Task Modifiers link: <http://ember-concurrency.com/docs/task-concurrency>

# **6. Common Problems and Pitfalls that new developers face**

* Computed properties - dependencies
* Computed properties - blind set
* Setting properties to destroyed components
* Shallow cloning of array and deeply nested objects
* Forgetting to reset controller properties

# **7. Frontend best practices and deprecations**

### <https://confluence.freshworks.com/display/freshsales/Front+End+Best+Practices>

**Deprecations:**

* Select-2
* link-modal
* Rl-dropdown

For simple dropdowns, lets use **ember-basic-dropdown**  
For dropdowns on hover, lets use **dropdown-hover**  
For dropdowns with search enabled, lets use **power-select**