Aprameyah
Technologies Pvt. Ltd.

CanJS

JS Web Development Framework

Constructor Functions

- Can JS uses its own constructor function for creating classes
- Create constructor functions by extending empty objects like this:

```
var User = can.Construct.extend({}, {
    init: function(name) {
         this.name = name;
    },
     somefunction: function(){
        console.log("Hello there...");
     },
     getName: function() {
        return this.name;
     }
});

var u = new User("Sarah");
console.log(u.getName());
```

Aprameyah
Technologies Pvt. Ltd.

Inheriting in CanJS

- Call extend on an existing Constructor function to inherit.

    - Dont forget to call the base class init

```
var PayingUser = User.extend({
    init: function(name, memberType) {
       User.prototype.init.apply(this, arguments);
       this.memberType = memberType;
    },
    getMemberType: function(){
       return this.memberType;
    }
});
```

Aprameyah Technologies Pvt. Ltd.

Observables

- CanJS observables let you make changes to data and listen to those changes
- We have three types of observeables
  - can.Map - Used for Objects.
  - can.List - Used for Arrays.
  - can.compute - Used for values.

Aprameyah
Technologies Pvt. Ltd.

Observe Objects

- Observe Objects

```
var observedUser = new can.Map(new User("Sona"));
```

- Access attributes of observed objects

```
console.log(observedUser.attr(“name"));
observedUser.attr("name","Vinay")
```

- When a property is changed with attr, two events happen: A change event and an event with the same name as the property that was changed.

```
observedUser.bind('change', function(event, attr, how, newVal, oldVal) {
    console.log(attr); // 'name'
    console.log(how); // 'set'
    console.log(newVal); // Vinay
    console.log(oldVal); // Sona
});

observedUser.bind('name', function(event, newVal, oldVal) {
    console.log("Name event: "+newVal); // Sona
    console.log("Name event: "+oldVal); // Vinay
});
```

Aprameyah Technologies Pvt. Ltd.

Observing Lists

- List can monitor arrays
  - the change event fires on every change to a List.
  - the set event is fired when an element is set.
  - the add event is fired when an element is added to the List.
  - the remove event is fired when an element is removed from the List.
  - the length event is fired when the length of the List changes.

Aprameyah Technologies Pvt. Ltd.

Observing Lists

```
var list = new can.List(['Alice', 'Bob', 'Eve']);
list.bind('change', function() { console.log('An element changed.'); });
list.bind('set', function() { console.log('An element was set.'); });
list.bind('add', function() { console.log('An element was added.'); });
list.bind('remove', function() {
   console.log('An element was removed.');
});
list.bind('length', function() {
   console.log('The length of the list changed.');
});
list.attr(0, 'Alexis'); // 'An element changed.'
// 'An element was set.'
list.attr(3, 'Xerxes'); // 'An element changed.'
// 'An element was added.'
// 'The length of the list was changed.'
list.attr(['Adam', 'Bill']); // 'An element changed.'
// 'An element was set.'
// 'An element was set.'
list.pop(); // 'An element changed.'
// 'An element was removed.'
// 'The length of the list was changed.'
```

Apprameyah Technologies Pvt. Ltd.

Computed Values

- A Compute represents a dynamic value that can be read, set, and listened to just like a Map.

```
// create a Compute
var age = can.compute(25),
previousAge = 0;
// read the Compute's value
age(); // 25
// listen for changes in the Compute's value
age.bind('change', function(ev, newVal, oldVal) {
previousAge = oldVal;
});
// set the Compute's value
age(26);
age(); // 26
previousAge; // 25
```

Aprameyah Technologies Pvt. Ltd.

Composite Computes

- Computes can also be used to generate a unique value based on values derived from other observable properties

```
var observedName = new can.Map({
   first: 'Will',
   last: 'Berger'
});

var fullName = can.compute(function() {
   // We use attr to read the values
   // so the compute knows what to listen to.
   return observedName.attr('first') + ' ' + observedName.attr('last');
});
console.log(fullName());
```

Aprameyah Technologies Pvt. Ltd.

Models

- Models are special Observes that connect to RESTful services.
- Models have these basic properties
  - findAll, which describes how to get a group of items.
  - findOne, which describes how to get a specific item.
  - create, which describes how to save a new item.
  - update, which describes how to update an existing item.
  - destroy, which describes how to delete an item.

Aprameyah Technologies Pvt. Ltd.

Example of a Model

```
var User = can.Model({
   findAll: 'GET /users',
   findOne: 'GET /users/{id}',
   create: 'POST /users',
   update: 'PUT /users/{id}',
   destroy: 'DELETE /users/{id}'
}, {});

var user = new User({name: 'Mina', age:33});

user.save(function(savedObj){
   console.log("Object saved");
});
```

Aprameyah Technologies Pvt. Ltd.

Other CRUD operations

```
User.findOne({id: '5503269bd7b8d3c3e07a9462'}).done(function(user) {
      console.log(user.attr('name',"Suresh2"));
      user.save().done(function(updatedObj){
         console.log(updatedObj.attr("name"));
      })
});

User.findAll({},function(users){
      users.forEach(function(user,index,list){
      user.attr("name",'Maruthi');
      user.save(function(){});
   });
});

user.destroy(function(deletedUser){
   console.log("User deleted");
});
```

Listening to Events

- Because Models are Observes, you can bind to the same events as on any other Observe. In addition to those events, Models emit three new kinds of events:
  - created, when an instance is created on the server.
  - updated, when an instance is updated on the server.
  - destroyed, when an instance is destroyed on the server.

```
User.bind('created', function(ev, created) {
    console.log("Created user ");
});
```

Aprameyah Technologies Pvt. Ltd.

Templates

- can.view loads and renders templates with the data you provide, and returns a documentFragment with the populated template.
- Embedded Javascript (EJS) is a templating language supported along with Mustache

Apprameyah Technologies Pvt. Ltd.

Using Templates

```
User.findAll({}, function(users) {
    console.log("Rendering template");
      $('#users').html(can.view('usersList', {
                                    list: users
                                }, {
                                    getSize: function(list){
                                        return list.length;
                                    }
                                }))
});
```

Here userList is the id of the script tag that contains the view

```
<script type="text/ejs" id="usersList">
<% can.each(this.list, function(val, key) { %>
    <li><%= val.attr('name') %></li>
<% }); %>
<%= getSize(this.list) %>
</script>
```

EJS

- EJS is CanJS's default template language
- <% %> will run any JavaScript code inside of it.
- <%= %> will evaluate a JavaScript statement and write the HTML-escaped result into the populated template.

```
<div>Here is a bold element <%= '<b>bold</b>' %>.</div>

<div>Here is a bold element &lt;b&gt;bold&lt;/b&gt;.</div>
```

- <%== %> does not escape

Aprameyah Technologies Pvt. Ltd.

EJS Live Binding

- Live binding will automatically update your EJS templates in the DOM whenever the data they are populated with changes.
    - populate templates with Observes and use attr to read properties.

Aprameyah Technologies Pvt. Ltd.

EJS Element Callbacks

- If the code inside <%= %> or <%== %> evaluates to a function, the function will be called back with the element it's inside as its first argument

```
<img src="hidden.gif" <%= function(element) { element.style.display =
'none'; } %>/>

<li <%= function(element){ $(element).data('user', user) } %>>
```

Controls

- Controls are classes that manage the models and views (MVC controller)
- Defines event handlers for elements of a view
- Loads view templates with data when created

Aprameyah
Technologies Pvt. Ltd.

Simple Control

```
var Users = can.Control({
    init: function(el, options) {
      var self = this;
      User.findAll({}, function(users) {
          self.element.html(can.view('userList', users));
      });
   }
});

var usersList = new Users('#users', {});
```

Aprameyah Technologies Pvt. Ltd.

Controls

- Init function param1 = wrapped NodeList for the provided selector
- param2 = options provided during creation extending options provided during class definition

Defining options

```
var Users = can.Control({
      defaults: {
         viewTemplate: 'userList'
      }
   },{
     init: function(el, options) {
      var self = this;
      User.findAll({}, function(users) {
           self.element.html(can.view(options.viewTemplate,
users));
      });
   }
});
```

Handling Events on Controller

- Inside controller we can define functions with a selector and event like this:

```
'li click': function(el, ev) {
   console.log('You clicked ' + el.text());
},

'div.info click': function(el, ev) {
   var li = el.closest('li'),
}
```

Lets Try It

```
var Users = can.Control({
        init: function(el, options) {
          var self = this;
          User.findAll({}, function(users) {
             self.element.html(can.view('usersList', users));
          });
        },
       'li .destroy click': function(el, ev) {
          var li = el.closest('li'),
          user = li.data('user');
          user.destroy();
        }
    });


var userControl = new Users(“#users",{});

<script type="text/ejs" id="usersList">
   <% this.each(function(user) { %>
      <li <%= function(element){ $(element).data('user', user) } %>>
         <%= user.attr('name'); %>
         <a class="destroy">X</a>
      </li>
   <% }) %>
</script>
```

Aprameyah
Technologies Pvt. Ltd.

Templating Event Handlers

- If a variable is placed in braces in the event handler key, can.Control will look up that key in the Control's options

```
'div.info {openUser}’: function(el, ev) {
   var li = el.closest('li'),
}

var usersList = new Users('#users', {openUser: ‘click’});
```

Apprameyah
Technologies Pvt. Ltd.

Rebinding Events

- You can unbind and rebind all a Control's event handlers by calling “on” on it.

- This is useful when a Control starts listening to a specific Model, and you want to change which model it is listening to.

```
setUser: function(user) {
   this.options.user = user;
   this.on();
 },

 '{user} updated': function() {
   //Handle the updated user
 },
```