**BackboneJS**

BackboneJs là một thư viện javascript dung để phát triển fontend.Nó gồm những thư viện BackboneJs,Jquery,Undercore.

Cấu trúc của BackboneJS bao gồm các module: Model, View, Collection, Event, Router, Sync.

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh trên là mô hình hoạt động của backbone.

***1.View***

View là module để hiển thị dữ liệu lên trang giạo diện sau khi đã nhận được sự kiện từ người dùng, sau khi Model đã xử lí dữ liệu, logic. BackboneJS cung cấp những option giúp cho việc hiển thị dữ liệu ra ngoài giao diện như TagName, ClassName.

Cấu trúc khai báo một View trong BackboneJs:

var DocumentRow = Backbone.View.extend({

    tagName: "li",

    className: "document-row",

    events: {

      "click .icon":          "open",

      "click .button.edit":   "openEditDialog",

      "click .button.delete": "destroy"

    },

//events liệt kê những sự kiện có thể xảy ra theo key:value,với key là className ứng với sự kiện xả ra,còn value là phương thức nếu xảy ra sự kiện đó.

    initialize: function() {

      this.listenTo(this.model, "change", this.render);

    },

    render: function() {

      ...

    }

  });

***2.Model***

Model là trái tim, là cốt lõi của Backbonejs cũng như bất kì ứng dụng javascript nào . Model là thể hiện dữ liệu,là nơi chứa dữ liệu để tương tác, nơi để xử lí logic từ yêu cầu của người dung theo tác tên fontend.

Cấu trúc khai báo một Model trong BackboneJS.

var model\_name = Backbone.Model.extend({

    defaults: {

name: "Applesauce cake",

     price: "20"

//khai báo giá trị mặc định của attribute theo key:value

    },

    initialize: function(){

        //Phương thức contructor của object,phương thức initialize sẽ được gọi khi một Model được khởi tạo.

    },

    myMethod: function(){

        //Phương thức của object

    }

//

  });

***3.Colection***

Collection là nơi để lưu 1 list những Model.Dùng để sắp xếp các đối tượng được xác định trước đó sau, để thao tác với list Model này cần nhiều function(). Collecttion dung phương thức add() và remove() để them hoặc xóa Model trong danh sách các Model.

Cách khởi tạo Collection:

var Shape = Backbone.Model.extend({

    defaults: { x:50, y:50, width:150, height:150, color:'black' },

     setTopLeft: function(x,y) { this.set({ x:x, y:y }); },

     setDim: function(w,h) { this.set({ width:w, height:h }); },

     isCircle: function() { return !!this.get('circle'); }

});

var Document = Backbone.Collection.extend({ model: Shape });

// với Shape là Model của chương trình

// Vi dụ về cách sử dụng add() và remove().

var document = new Document();

var documentView = new DocumentView({ collection: document });

$('#new-rectangle').click(function() {

// khi có sự kiện click vào nút có id=”#new-rectangle'” thì Collection sẽ thêm một Model Shape() để hiển thị trên màn hình một hình vuông có kích thước đã được defaults

     document.add(new Shape());

});

$('#new-circle').click(function() {

//,Tương tự,khi có sự kiện click vào nút có id=”#new-circle” thì sẽ hiển thị lên màn hình.

     document.add(new Shape({ circle: true }));

***4.Router***

Route trong Backbone được sử dụng để quán lí trạng thái và các sự kiện của trang web thông qua Url.

Cách định nghĩa Router và ví dụ:

<script type='text/javascript'>

      (function () {

        var ProductRouter = Backbone.Router.extend({

          routes: {

            '': 'index',

            'product/:id': 'viewProduct',

            'product/:id/edit': 'editProduct'

          },

          index: function () {

            $(document.body).append('Index route has been called..');

          },

          viewProduct: function(id) {

            $(document.body).append('<p>you are viewing product : ' + id + '</p>')

            this.navigate('product/' + id + '/edit', {trigger: true});

          },

          editProduct: function(id) {

            $(document.body).append('<p>you are editting product : ' + id + '</p>')

// khi đường dẫn là index.html#product/2/edit Router nhận được đó là phương thức editProduct nhờ routes.

          }

        });

        var app = new ProductRouter;

        app.on('route:viewProduct', function() {

          alert('dfsghjh')

        })

        Backbone.history.start();

      })();

    </script>

Router có ứng dụng rất tuyệt cho những trang web one page hoặc landing page giống như trên giao diện trên trang chủ của <https://backbonejs.org/> .Khi ấn vào một mục bất kì trên thanh menu Url sẽ thay đổi và nhảy đúng đến phần có nội dung tương ứng

***5.Ví dụ đã thực hành***

$(function() {

var Shape = Backbone.Model.extend({

    //khởi tạo model với màu và chiều dài chiều rộng,vị trí x,y

    defaults: { x:50, y:50, width:150, height:150, color:'black' },

    setTopLeft: function(x,y) { this.set({ x:x, y:y }); },

    setDim: function(w,h) { this.set({ width:w, height:h }); },

    isCircle: function() { return !!this.get('circle'); }

});

var Document = Backbone.Collection.extend({ model: Shape });

var DocumentView =  Backbone.View.extend({

    id: 'page',

    views: {},

    initialize: function() {

        this.collection.bind('add', function(model) {

            this.views[model.cid] = new ShapeView({ model: model, id:'view\_' + model.cid }).render();

            console.log(this.collection);

        }, this);

        this.collection.bind('remove', function(model) {

            this.views[model.cid].remove();

            delete this.views[model.cid];

        }, this);

    },

    render: function() {

        return this;

        //hàm trả lại đối tượng DocumentView

    }

});

var ShapeView = Backbone.View.extend({

    initialize: function() {

        $('#page').mousemove(this, this.mousemove).mouseup(this, this.mouseup);

        this.model.bind('change', this.updateView, this);

        // console.log(this.model);

    },

    render: function() {

        $('#page').append(this.el);

        $(this.el).html('<div class="shape"/>'

                  + '<div class="control delete hide"/>'

                  + '<div class="control change-color hide"/>'

                  + '<div class="control resize hide"/>')

            .css({ position: 'absolute', padding: '10px' });

        if (this.model.isCircle()) {

            this.$('.shape').addClass('circle');

        }

        this.updateView();

        return this;

    },

    updateView: function() {

        $(this.el).css({

            left:       this.model.get('x'),

            top:        this.model.get('y'),

            width:      this.model.get('width') - 10,

            height:     this.model.get('height') - 10 });

        this.$('.shape').css({ background: this.model.get('color') });

    },

    events: {

        'mouseenter .shape': 'hoveringStart',

        'mouseleave': 'hoveringEnd',

        'mousedown .shape': 'draggingStart',

        'mousedown .resize': 'resizingStart',

        'mousedown .change-color': 'changeColor',

        'mousedown .delete': 'deleting',

    },

    hoveringStart: function (e) {

        this.$('.control').removeClass('hide');

    },

    hoveringEnd: function (e) {

        this.$('.control').addClass('hide');

    },

    draggingStart: function (e) {

        this.dragging = true;

        this.initialX = e.pageX - this.model.get('x');

        this.initialY = e.pageY - this.model.get('y');

        return false; // prevents text selection

    },

    resizingStart: function(e) {

        this.resizing = true;

        return false;

    },

    changeColor: function(e) {

        this.model.set({ color: prompt('Enter color value', this.model.get('color')) });

    },

    deleting: function(e) {

        this.model.collection.remove(this.model);

    },

    mouseup: function (e) {

        if (!e || !e.data) return;

        var self = e.data;

        self.dragging = self.resizing = false;

    },

    mousemove: function(e) {

        if (!e || !e.data) return;

        var self = e.data;

        if (self.dragging) {

            self.model.setTopLeft(e.pageX - self.initialX, e.pageY - self.initialY);

        } else if (self.resizing) {

            self.model.setDim(e.pageX - self.model.get('x'), e.pageY - self.model.get('y'));

        }

    }

});

var document = new Document();//tạo một collection

var documentView = new DocumentView({ collection: document });//khởi tạo 1 view có key là collectionvà value là đối tượng document

    console.log(document);

$('#new-rectangle').click(function() {

    document.add(new Shape());

    //khi click vào id #new-rectangle thì Model Shape() được khởi tạo và thêm vào collection:document

});

$('#new-circle').click(function() {

    document.add(new Shape({ circle: true }));

});

});

ví du 2:

var collection = new Backbone.Collection([

    {name: "Tim", age: 5},

    {name: "Ida", age: 26},

    {name: "Rob", age: 55}

  ]);

  alert(JSON.stringify(collection))

/tra ve toan vo key-value khoi tao cua collectioon/

var collection = new Backbone.Collection(

    {name:"chinh"},

    {name:"tien1"}

);

    // {name:"chinh"}

    // );

// console.log(collection);

collection.on("get",function(ship){

    alert("chao "+ ship.get("name")+"!");

});

//on de bat su kien khi collection add them 1 model

collection.add({name:"tien"});

// collection.set({name:"tien"})

console.log(collection.get("c2"));

//lay ra doi tuong model bang id hoac cid trong collection

vi du 3:

var chapter=Backbone.Model;

var chapters=new Backbone.Collection;

chapters.comparator='page';

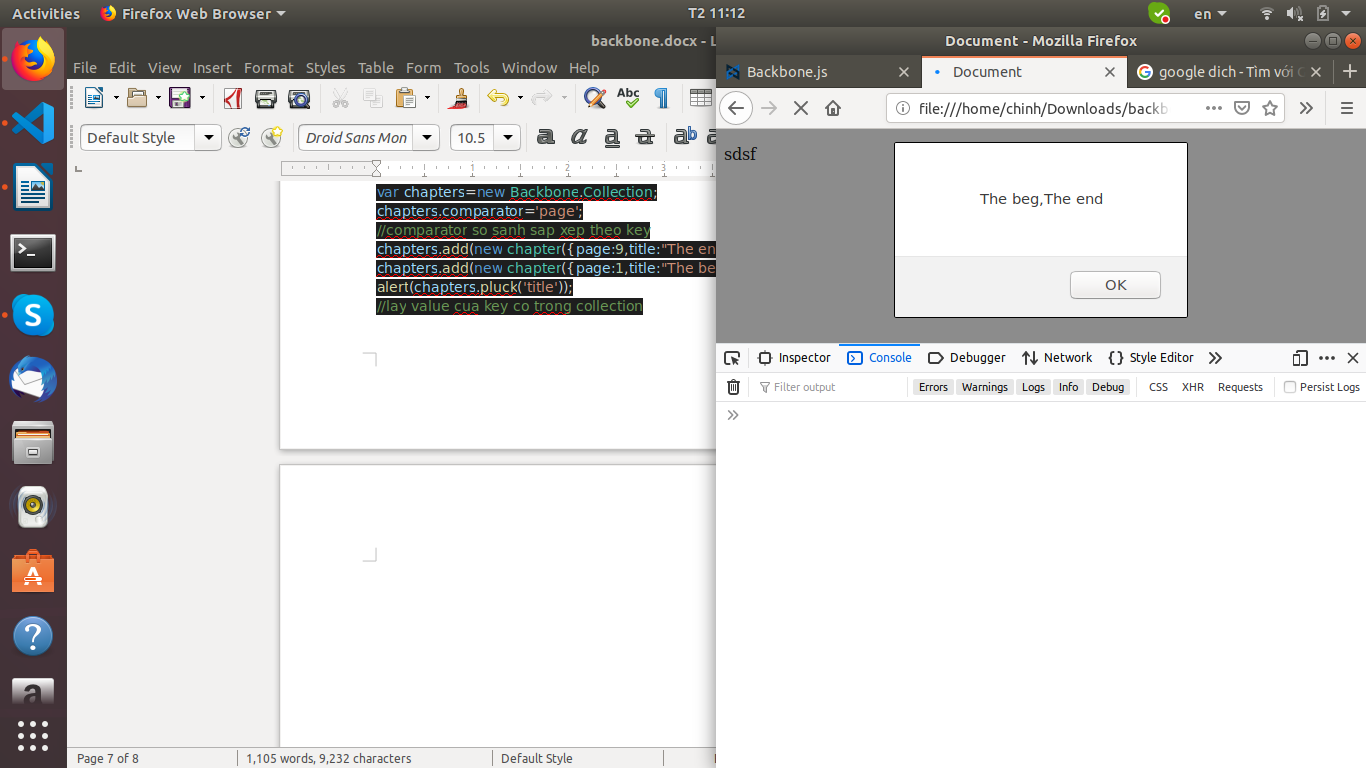
//comparator so sanh sap xep theo key

chapters.add(new chapter({page:9,title:"The end"}));

chapters.add(new chapter({page:1,title:"The beg"}));

alert(chapters.pluck('title'));

//lay value cua key co trong collection



vidu 4:

var friends=new Backbone.Collection([

    {name:"Tthos",job:"account"},

    {name:"Porthos",job:"account"},

    {name:"sdf",job:"teacher"}

]);

var mus=friends.where({job:"account"});

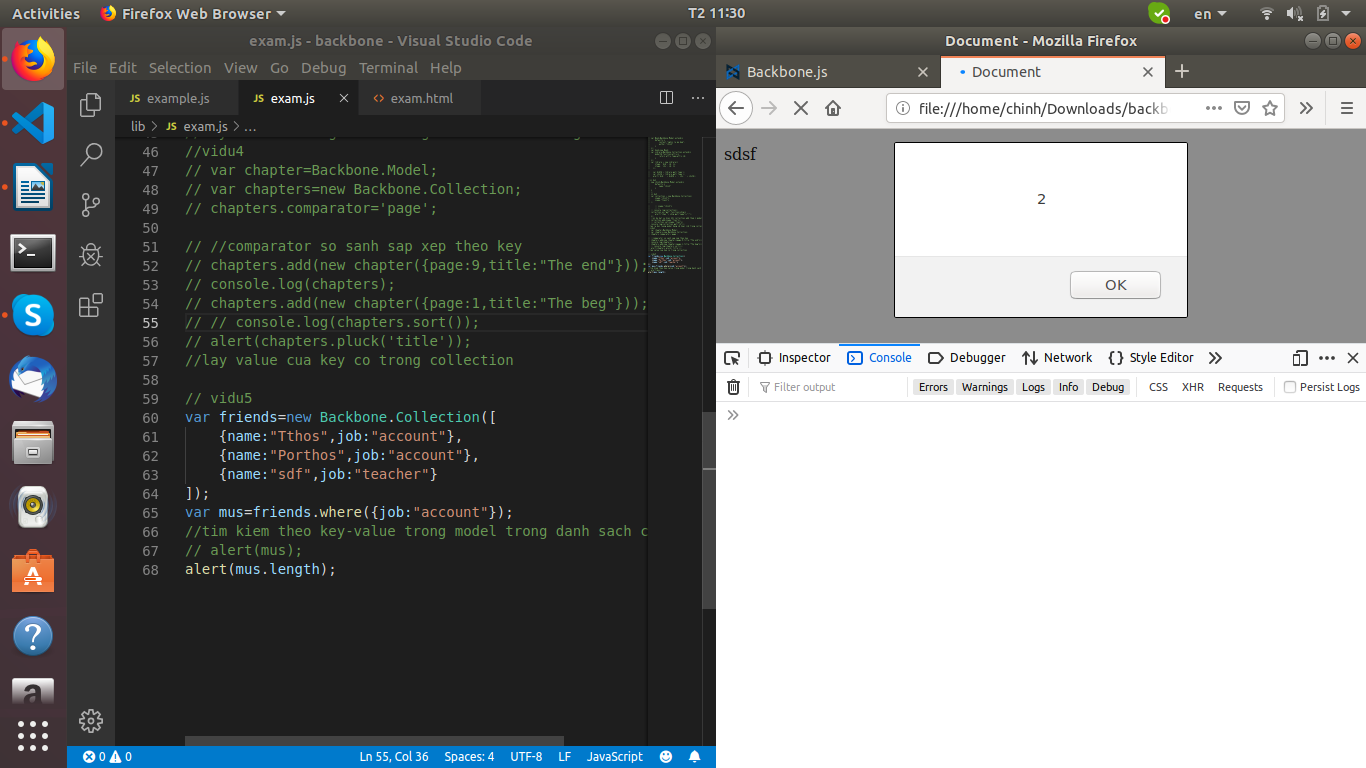
//tim kiem theo key-value trong model trong danh sach collection

var mus=friends.findWhere({job:"account"});

//tra ve model dau tien n tim thay trong collection

alert(mus.length);

collection.clone() : trả về đối tượng collection có cấu trúc dữ liệu y hệt.nhưng thao tác trên collection khi được clone thì sẽ không ảnh hưởng đến collection ban đầu.



vi du 5:

Backbone.sync=function(method,model){

    alert(method +":" +JSON.stringify(model));

    model.set('id',1);

};

var book=new Backbone.Model({

    title:"truyen",

    author:"tien"

});

book.save();

book.save("chinh","FDR",{error:function(){

    console.log(1);

}});

