KRAKEN.JS

The kraken suite is a secure and scalable layer that extends Express by providing structure and convention.

STACK

- generator-kraken
- kraken-js
- shortstop
- enrouten
- lusca

OTHER SEA MONSTERS

DUST.JS (LINKEDIN)

- Adaro
- Makara
- findatag

KAPPA

- Making npm effective.

STRUCTURE OF A KRAKENJS PROJECT

KRAKEN CONVENTIONS

PROJECT STUCTURE

```
/config
Application and middleware configuration

/controllers
Routes and logic

/lib
Custom developer libraries and other code

/locales
Language specific content bundles
```

KRAKEN CONVENTIONS

PROJECT STUCTURE

```
/models
Models

/public
Web resources that are publicly available

/public/templates
Server and browser-side templates

/tests
Unit and functional test cases

index.js
Application entry point
```

SIMPLE APPLICATION INDEX.JS

```
use strict';

var kraken = require('kraken-js'),
    app = {};

// Fired when an app configures itself
app.configure = function (nconf, next) {
    next(null);
};

// Fired at the beginning of an incoming request
app.requestStart = function (server) { };

// Fired before routing occurs
app.requestBeforeRoute = function (server) { };

// Fired after routing occurs
app.requestAfterRoute = function requestAfterRoute(server) { };

kraken.create(app).listen(function (err) {
    if (err) { console.error(err); }
});
```

CONFIGURATION

Kraken has available two application configuration files, one for production and another for development

```
$ tree config/
config/
├── app-development.json
├── app.json
└── middleware.json
```

To change the behavior of the application according to the environment, simply change the NODE_ENV env variable.

```
export NODE_ENV=production
export NODE_ENV=development
```

LUSCA: SECURITY

Security is provided by the Lusca module

Provides:

- Cross Site Request Forgery (CSRF) headers.
- Content Security Policy (CSP) headers.
- Platform for Privacy Preferences Project (P3P) headers.
- X-FRAME-OPTIONS to prevent clickjacking

ENROUTEN: ROUTES

Routing logic is contained inside the **controllers** folder

```
'use strict';
module.exports = function (server) {
  server.get('/', function (req, res) {
    var model = { name: 'Ash Williams' };
    res.render('index', model);
  });
};
```

ADARO: TEMPLATES

Adaro module is responsible for rendering LinkedIn's Dust.js templates

```
<h1>Hello {name}!</h1>
```

Templates can be found at **public/templates**

MAKARA: LOCALIZATION

Makara provides to Kraken a easy way for Kraken to adapt for internationalization

This is done by adding context to the response

```
var customerLocality = 'es_ES';
res.locals.context = {locality: customerLocality};
var model = {name: 'Antonio Banderas'};
res.render('index',model);
```

And updating our template

```
<h1>{@pre type="content" key="index.greeting"/}</h1>
```

KRAKEN SHOPPING CART

pregenerated cart found in `kraken-js/src/cart`
generated using `yo kraken`

SHOPPING CART: EXISTING FILES

Controllers / Models

```
cart/controllers/cart.js
cart/controllers/index.js
cart/controllers/products.js
cart/models/productModel.js
```

Wiring

```
cart/lib/database.js
cart/lib/language.js
```

Localization

```
cart/locales/ES/es/index.properties
cart/locales/ES/es/layouts/master.properties
cart/locales/US/en/index.properties
cart/locales/US/en/layouts/master.properties
```

View related

```
cart/public/css/app.less
cart/public/css/elements.less
cart/public/img/*.png
cart/public/templates/cart.dust
cart/public/templates/index.dust
cart/public/templates/layouts/master.dust
cart/public/templates/products.dust
cart/public/templates/result.dust
```

KRAKEN SHOPPING CART

Installing dependencies

```
$ cd cart
$ npm install
$ bower install
```

KRAKEN SHOPPING CART

Install the `mongoose` and `mockgoose` modules as dependencies of your project

```
$ npm install --save mockgoose mongoose
```

Add a `databaseConfig` to the root of the `cart/config/app.json` configuration file

```
"databaseConfig": {
  "host": "localhost",
  "database": "test"
}
```

This configuration will be parsed by the application on startup using nconf.

The resulting config will be used in `cart/lib/database.js`

KRAKEN SHOPPING CART

DATABASE CONNECTOR

cart/lib/database.js

```
'use strict';
var mongoose = require('mongoose');
var mockgoose = require('mockgoose');
mockgoose(mongoose);

module.exports = {
  config: function(conf) {

    mongoose.connect('mongodb://' + conf.host + '/' + conf.database);

    var db = mongoose.connection;

    db.on('error', function error(err) {
      console.log('connection error', err)
      process.exit(1);
    });

    db.once('open', function callback() {
      console.log('db connection open');
    });
  }
};
```

KRAKEN INITIALIZATION

Initialization can be performed at four different points:

- During configuration
- At the beginning of a request
- Before Routing
- After Routing

KRAKEN INITIALIZATION

cart/index.js

```
/// Configuration hooks
app.configure = function configure(nconf, next) {
  // Fired when an app configures itself
  next(null);
};

app.requestStart = function requestStart(server) {
  // Fired at the beginning of an incoming request
};

app.requestBeforeRoute = function requestBeforeRoute(server) {
  // Fired before routing occurs
};

app.requestAfterRoute = function requestAfterRoute(server) {
  // Fired after routing occurs
};
```

KRAKEN SHOPPING CART

DATABASE CONFIGURATION HOOK

Add these lines to `cart/index.js`

01_index.js

```
/// Set up the db to be used on ./index.js
var db = require('./lib/database');

app.configure = function configure(nconf, next) {
  // Fired when an app configures itself
  //Configure the database
  db.config(nconf.get('databaseConfig'));
  next(null);
};
```

Let's test the db connection

```
$ npm start
Listening on 8000
db connection open
```

KRAKEN SHOPPING CART

USING CUSTOM MIDDLEWARE

Determining user's preferred language

cart/lib/language.js

```
'use strict';
module.exports = function language(req, res, next) {
  //Pick up the language cookie.
  var language = req.cookies.language;

  //Set the locality for this response. The template will pick the appro
  if (language) {
    res.locals.context = res.locals.context || {};
    res.locals.context.locality = language;
  }
  next();
};
```

KRAKEN SHOPPING CART

USING CUSTOM MIDDLEWARE - DETERMINING USER'S PREFERRED LANGUAGE PART II

Add the following lines into `cart/index.js`

02_index.js

```
/// require our language library
var language = require('./lib/language');
```

02_index.js

```
/// set up the server so it's used before a route
app.requestBeforeRoute = function requestBeforeRoute(server) {
  // Fired before routing occurs
  server.use(language);
};
```

KRAKEN SHOPPING CART

USING CUSTOM MIDDLEWARE - DETERMINING USER'S PREFERRED LANGUAGE PART III

```
$ yo kraken:controller setLanguage
[?] Respond to XHR requests? No
   create controllers/setLanguage.js
```

Add the following to `cart/controllers/setLanguage.js`

05_setLanguage.js

```
/// set up the route to take the lang parameter
server.get('/setlanguage/:lang', function (req, res) {
  res.cookie('language', req.param('lang'));
  res.redirect('/');
});
```

KRAKEN SHOPPING CART

ADD LANGUAGE LINKS

Add these lines into `cart/public/layouts/master.dust` after `</nav>`

```
<div class="lang">
    <ul class="nm-np inline">
        <li><a href="/setLanguage/en-us"><img src="/img/us.png" alt="E
        <li><a href="/setLanguage/es-es"><img src="/img/es.png" alt="S
    </ul>
</div>
```

KRAKEN SHOPPING CART

USING CONTENT BUNDLES

Existing bundles in the cart application

```
locales/US/en/layouts/master.properties
locales/US/en/index.properties
locales/ES/es/layouts/master.properties
locales/ES/es/index.properties
```

Accessing properties in these files using dust:

```
{@pre type="content" key="<PROPERTY_NAME>"/}
```

KRAKEN SHOPPING CART

ADDING A NEW LANGUAGE

```
$ yo kraken:locale index PT pt
   create locales/PT/pt/index.properties
```

Translators can now translate the shopping cart to portuguese by editing this file

KRAKEN SHOPPING CART

PRODUCT MANAGEMENT

KRAKEN SHOPPING CART

PRODUCT MODEL

cart/models/productModel.js

```
'use strict';
var mongoose = require('mongoose');

// Define a simple schema for our products.
var productSchema = mongoose.Schema({
  name: String,
  price: Number
});

// Verbose toString method
productSchema.methods.whatAmI = function () {
  if (this.name) {
    console.log("Hello, I'm a", this.name, "and I'm worth €" + this.pric
  } else {
    console.log("I don't have a name :(");
  }
};

// Format the price of the product to show a euro sign, and two decimal
productSchema.methods.prettyPrice = function () {
  return '€' + this.price.toFixed(2);
};

module.exports = mongoose.model('Product', productSchema);
```

KRAKEN SHOPPING CART

Product Management: existing products

cart/controllers/products.js

```
/// products controller GET
server.get('/products', function (req, res) {

  Product.find(function (err, prods) {
    if (err) { console.log(err); }
    var model = { products: prods };
    res.render('products', model);
  });
});
```

KRAKEN SHOPPING CART

Product Management: existing products

cart/public/templates/products.dust

```
<!-- /// products listing -->
<div class="products">
  {?products}
    <fieldset>
        <legend>Product List</legend>
        <ul class="nm-np inline">
          {#products}
            <li>
              <form method="POST" action="/products">
                <input type="hidden" name="item_id" value="{.id}">
                <h3 class="nm-np">{.name}</h3>
                <h4 class="nm-np">{.prettyPrice}</h4>
                <h5 class="nm-np tiny">ID: {.id}</h5>
                <input type="submit" value="Delete">
                <!-- If we don't send the Cross-Site Request Forgey
                     this request will be rejected -->
                <input type="hidden" name="_csrf" value="{_csrf}">
                <input type="hidden" name="_method" value="DELETE">
              </form>
            </li>
          {/products}
        </ul>
    </fieldset>
  {:else}
    There are no products :(
  {/products}
</div>
```

KRAKEN SHOPPING CART

Product Management: add new product

cart/controllers/products.js

```
/// products controller POST
server.post('/products', function (req, res) {
  var name = req.body.name && req.body.name.trim();
  var price = parseFloat(req.body.price, 10);

  if (name === '' || isNaN(price)) {
    res.redirect('/products#BadInput');
    return;
  }
  var newProduct = new Product({name: name, price: price});
  newProduct.whatAmI();
  newProduct.save();
  res.redirect('/products');
});
```

KRAKEN SHOPPING CART

Product Management: add new product form

cart/public/templates/products.dust

```
<!-- /// Add a new product -->
<div class="mb2">
  <fieldset>
    <legend>Add a new product</legend>
    <form method="POST" action="/products">
      <input name="name" placeholder="Product Name"><br>
      <input name="price" placeholder="Price"><br>
      <input type="hidden" name="_csrf" value="{_csrf}">
      <input type="submit" value="Save">
    </form>
  </fieldset>
</div>
```

KRAKEN SHOPPING CART

METHODOVERRIDE

Override the http method with a GET/POST `_method` param

Add the following to `cart/index.js`

19_index.js

```
/// require the express module
var express = require('express');
```

19_index.js

```
/// add before route
app.requestBeforeRoute = function requestBeforeRoute(server) {
  // Fired before routing occurs
  server.use(language);
  server.use(express.methodOverride());
};
```

Used in templates

```
<form action="/resource/id" method="POST">
  <input type="hidden" name="_method" value="DELETE" />

  ...

</form>
```

KRAKEN SHOPPING CART

Product Management: remove product

cart/controllers/products.js

```
/// products controller DELETE
server.delete('/products', function (req, res) {
  Product.remove({_id: req.body.item_id}, function (err) {
    if (err) { console.log('Remove error: ', err); }
    res.redirect('/products');
  });
});
```

KRAKEN SHOPPING CART

Product Management: remove product form

cart/public/templates/products.dust

```
<!-- /// products listing -->
<div class="products">
  {?products}
    <fieldset>
        <legend>Product List</legend>
        <ul class="nm-np inline">
          {#products}
            <li>
              <form method="POST" action="/products">
                <input type="hidden" name="item_id" value="{.id}">
                <h3 class="nm-np">{.name}</h3>
                <h4 class="nm-np">{.prettyPrice}</h4>
                <h5 class="nm-np tiny">ID: {.id}</h5>
                <input type="submit" value="Delete">
                <!-- If we don't send the Cross-Site Request Forgey
                     this request will be rejected -->
                <input type="hidden" name="_csrf" value="{_csrf}">
                <input type="hidden" name="_method" value="DELETE">
              </form>
            </li>
          {/products}
        </ul>
    </fieldset>
  {:else}
    There are no products :(
  {/products}
</div>
```

KRAKEN SHOPPING CART

CUSTOMER INTERFACE

Now that we have the ability to manage products, lets let people buy them!

KRAKEN SHOPPING CART

Viewing Products: controller

cart/controllers/index.js

```
/// add the root route
server.get('/', function (req, res) {
  Product.find(function (err, products) {
    if (err) {
      console.log(err);
    }

    res.render('index', {
      products: products
    });
  });
});
```

KRAKEN SHOPPING CART

Viewing Products: template

cart/public/templates/index.dust

```
<!-- /// Customer product display -->
<div class="products">
  <ul class="nm-np inline">
    {#products}
      <li>
        <form method="POST" action="cart">
          <input type="hidden" name="item_id" value="{.id}">

          <h3 class="nm-np">{.name}</h3>
          <h4 class="nm-np">{.prettyPrice}</h4>
          <input type="submit" value="{@pre type="content" key="inde
          <input type="hidden" name="_csrf" value="{_csrf}">
        </form>
      </li>
    {:else}
      <li>There are no products :(<br>You should <a href="/products"
    {/products}
  </ul>
</div>
```

KRAKEN SHOPPING CART

Add a product to the cart with http POST

cart/controllers/cart.js

```
/// Handle POST
server.post('/cart', function (req, res) {
  req.session.cart = req.session.cart || {};
  var cart = req.session.cart;
  var id = req.param('item_id');

  Product.findById(id, function (err, prod) {
    if (err) {
      console.log('Error adding product to cart: ', err);
      return res.redirect('/cart');
    }

    // Add or increase the product quantity in the shopping cart.
    if (cart[id]) {
      cart[id].qty++;
    } else {
      cart[id] = {
        name: prod.name,
        price: prod.price,
        prettyPrice: prod.prettyPrice(),
        qty: 1
      };
    }
    // Display the cart for the user
    res.redirect('/cart');
  });
});
```

KRAKEN SHOPPING CART

Displaying the cart

cart/controllers/cart.js

```
/// Handle GET
server.get('/cart', function (req, res) {
  var cart = req.session.cart,
    displayCart = {items: [], total: 0},
    total = 0;

  if (!cart) {
    res.render('emptyCart', { message: 'Your cart is empty!'});
    return;
  }

  // Ready the products for display
  for (var item in cart) {
    displayCart.items.push(cart[item]);
    total += (cart[item].qty * cart[item].price);
  }
  req.session.total = displayCart.total = total.toFixed(2);
  var model = { cart: displayCart };
  res.render('cart', model);
});
```

KRAKEN SHOPPING CART

Cart templates

cart/public/templates/cart.dust

```
        <!-- /// List products in users cart -->
        <div class="products mb2">
            <h2>Your cart </h2>
            <ul class="nm-np inline">
                {#cart.items}
                    <li>
                        <h3 class="nm-np">{.qty} x {.name}</h3>
                        <h4 class="nm-np">Price: {.prettyPrice} ea.</h4>
                    </li>
                {/cart.items}
            </ul>
        </div>
```

cart/public/templates/emptyCart.dust

```
{>"layouts/master" /}

{<body}
    <main role="main">
        Result: {.message}
        <h3><a href="/">Keep on buying!</a></h3>
    </main>
{/body}
```

KRAKEN SHOPPING CART

Time to test your brand new app with krakenjs!

```
$ npm start
```

SUMMARY

- Kraken.js provides a configurable wrapper around Express
  - Security by default
  - Convention + structure for a unified team workflow
- Generate new components with yeoman