Crear módulos de datos utilizando archivos RAML

# Instalar bb-convert y generar módulos de datos mockup

**Correr:**

npm install -g @bb-cli/bb-convert

**Verifique que la herramienta esté instalada:**

bb-convert --help

# Convierte RAML en módulos de datos

**Recuerde la estructura del directorio del proyecto.**

└─ statics  
 ├── collection-myportal  
 │   ├── pom.xml  
 │   └── src  
 ├── package.json  
 └── pom.xml

**Los archivos necesarios se encuentran en la siguiente ruta compartida:** <https://drive.google.com/open?id=1DyLVZ5fdjJ1TFkvxOWPDVz86NotSpOAZ>

Extraiga el archivo raml.zip que se encuentra en la carpeta **BB-Training/exercise** en la ruta raíz del proyecto **statics**

***Después de descomprimir el archivo obtendrá tres rutas de archivos raml:***

***- raml/advisors/advisors.raml***

***- raml/appointments/appointments.raml***

***- raml/customers/customers.raml***

Dentro de la carpeta **statics** ejecutar los siguientes comandos:

Use **bb-convert** para generar los módulos de datos:

**RAML a MOCKUP**

bb-convert raml **name-folder-raml**/**file.raml** --template mock-ng --o **name-folder-you-collection** /src -y

**Nota:** Cuando se convierte RAML a Mockup, con la instrucción, se crea el folder: mock.data-bb-**name-folder**-http-ng en la ruta que se especifica con **–output** y el **–y** es para reemplazar el folder si ya existe, cuando convierte RAML a Servicios es: data-bb-**name-folder**-http-ng, estos serán los nombre de las carpetas para ser importados en el modelo del widgets.

**Convirtiendo archivos RAML a Mockup**

bb-convert raml raml/advisors/advisors.raml --template mock-ng --output collection-training/src -y

bb-convert raml raml/appointments/appointments.raml --template mock-ng --output collection-training/src -y

bb-convert raml raml/customers/customers.raml --template mock-ng --output collection-training/src -y

# Inyectar el Modulo de Datos al Modelo del Widget

Abrir el archive de la ruta **statics/collection-training/src/model-training-appointments-ng/scripts/index.js**

Buscar:

**import** Model from './appointments';

Debajo de esa línea, agregar lo siguiente:

**import** dataProviderAdvisorsKey, { advisorsDataKey } from "mock.data-bb-advisors-http-ng";

**import** dataProviderAdvisorAppointmentsKey,{ advisor\_AppointmentsDataKey } from "mock.data-bb-advisor\_appointments-http-ng";

**import** dataProviderCustomersKey, { customersDataKey } from "mock.data-bb-customers-http-ng";

**Comentario importante:** La carpeta **mock.data-bb-advisors-http-ng**/**scripts/index.js** existe la siguiente instrucción de exportación:

**export const** advisorsDataKey = 'data-bb-advisors-http-ng:advisorsData';

Como se observa la llave es **advisorsDataKey**, que es la constante que será importada como se observa.

**import** dataProviderAdvisorsKey, { **advisorsDataKey** } from "mock.data-bb-advisors-http-ng";

Una vez importadas todos los mockup buscar:

.module(moduleKey, [

])

**Reemplazar** por:

.module(moduleKey, [

dataProviderAdvisorsKey,

dataProviderAdvisorAppointmentsKey,

dataProviderCustomersKey,

])

Buscar:

'$q'

Reemplazar por:

'$q',

advisorsDataKey,

advisor\_AppointmentsDataKey,

customersDataKey,

### Agregar las dependencias

Abrir la siguiente carpeta: **statics/collection-training/src/model-training-appointments-ng/scripts/appointments.js**

Buscar:

**export** **default** **function** appointmentsModel(Promise) {

**Reemplazar** por:

**export** **default** **function** appointmentsModel(Promise, AdvisorsData, AdvisorAppointmentsData, CustomersData) {

## Implementar los métodos en el Modelo

**Reemplazar el contenido** dentro de la función de **appointmentsModel por**:

**Comentario: las funciones dependerá del mockup generado anteriormente por ejemplo: mock.data-bb-advisors-http-ng\scripts\data-bb-advisors-http.js, solo retorna getAdvisors, esta función se llamara desde el controlador del widgets.**

**function** getAdvisors() {

**return** AdvisorsData

.**getAdvisors**()

.then(**function** (response) {

**return** response.data;

});

}

**function** getCustomers() {

**return** CustomersData

.getCustomers()

.then(**function** (response) {

**return** response.data;

});

}

**function** getAdvisorAppointments() {

**return** AdvisorAppointmentsData

.getAdvisorAppointments()

.then(**function** (response) {

**return** response.data;

});

}

**return** {

getAdvisors,

getCustomers,

getAdvisorAppointments,

};

# Cargar las funciones dentro del Controlador de appointments

Abrir el archivo de la ruta **statics/collection-training/src/widget-training-appointments-ng/scripts/controller.js**

**Remover** el siguiente fragmento de código:

model.load()

.then(loaded => {

$ctrl.items **=** hooks.itemsFromModel(loaded);

}) .**catch**(error => {

$ctrl.error **=** errorMessage(error.code);

bus.publish('widget-training-appointments-ng.load.failed', { error });

})

.then(() => { $ctrl.isLoading **=** **false**; });

Buscar:

$ctrl.isLoading **=** **true**;

Debajo de la línea **agregar** **las funciones que fueron declaradas en el Modelo.**

$ctrl.appointments **=** [];

model

.getCustomers()

.then(**function** (customers) {

$ctrl.customers **=** customers;

});

model

.getAdvisors()

.then(**function** (advisors) {

$ctrl.advisors **=** advisors;

});

model

.getAdvisorAppointments()

.then(appointments => {

$ctrl.appointments **=** appointments;

})

.**catch**(error => {

$ctrl.error **=** errorMessage(error.code);

bus.publish('widget-training-appointments-ng.load.failed',{ error });

})

.then(() => { $ctrl.isLoading **=** **false**; });

# Crear la vista del widget appointments

Abrir **statics/collection-training/src/ext-training-appointments-ng/templates/template.ng.html**

**Reemplazar** el contenido por el siguiente código:

<section class**=**"ext-training-appointments-ng" ng-controller**=**"AppointmentsController as $ctrl">

<header ng-if**=**"$ctrl.appointments.length === 0">

No appointments available.

</header>

<ul>

<li ng-repeat**=**"item in $ctrl.appointments">

<section>

<div>

{{item.date | date:'dd/MM/yyyy'}}

</div>

<div>

{{item.date | date:'HH:mm'}}

</div>

<div>

{{item.subject}}

</div>

<ul>

<li ng-repeat**=**"guest in item.guests">

<span>{{guest.personalData.fullname}}</span>{{guest.personalData.email}}

</li>

</ul>

</section>

</li>

</ul>

</section>

# El código anterior, se puede modificar para que el widget sea responsivo

Utilizando las clases de Bootstrap para que sea **responsive** y usuando **Font Awesome icons**

Reemplace el contenido **statics/collection-training/src/ext-training-appointments-ng/templates/template.ng.html** por lo siguiente:

<section class**=**"ext-training-appointments-ng" ng-controller**=**"AppointmentsController as $ctrl">

<header ng-if**=**"$ctrl.appointments.length === 0">

No appointments available.

</header>

<ul class**=**"list-group">

<li class**=**"list-group-item" ng-repeat**=**"item in $ctrl.appointments">

<section class**=**"row row-eq-height">

<div class**=**"col-xs-5 col-sm-4 col-md-3 col-lg-2 right-border">

<div class**=**"text-center col-xs-3 col-sm-4 col-md-4">

<span class**=**"fa fa-calendar fa-2x"></span>

</div>

<div class**=**"col-xs-9 col-sm-8 col-md-8 col-lg-8">

<div class**=**"text-left">

{{item.date | date:'dd/MM/yyyy'}}

</div>

<div class**=**"text-left">

{{item.date | date:'HH:mm'}}

</div>

</div>

</div>

<div class**=**"col-xs-7 col-sm-8 col-md-9 col-lg-10 overflow-hidden-on-xs">

<h5 ng-bind**=**"item.subject"></h5>

<div class**=**"col-xs-12 col-sm-9">

<ul>

<li ng-repeat**=**"guest in item.guests">

<h6>

<span ng-bind**=**"guest.personalData.fullname"></span>

<a href**=**"mailto:{{guest.personalData.email}}"

class**=**"hidden-xs">&lt; {{guest.personalData.email}} &gt;</a>

</h6>

</li>

</ul>

</div>

</div>

</section>

</li>

</ul>

</section>

# Probar la funcionalidad del widget antes de impórtalos al CXP-Manager

Primeramente debes tener el archivo **.bbconfig**, con el mismo puerto del cxp-manager en que corre y el local para testear los widgets antes de empaquetarlos e impórtalos.

Correr:

npm start

# Nota: La instrucción de npm start debe estar declarada en el archivo package.json, para su funcionamiento.

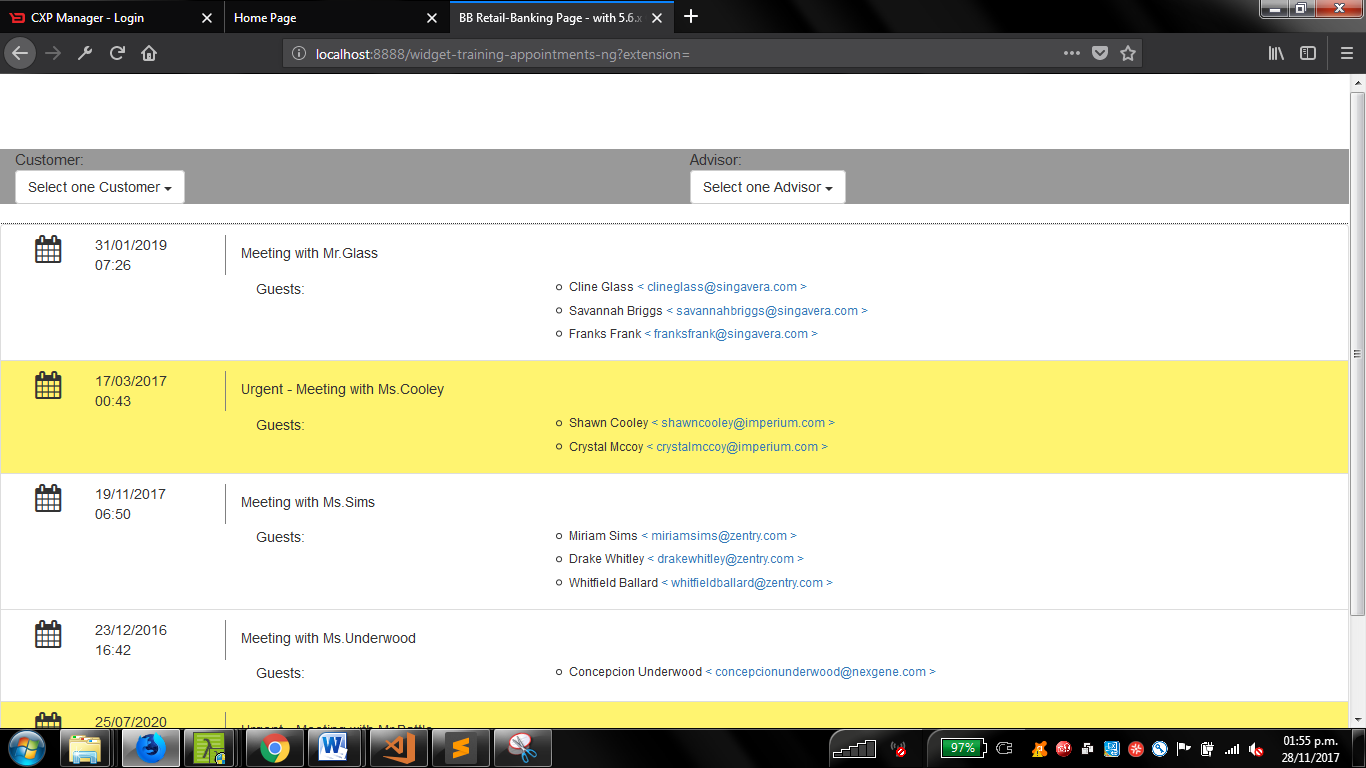
Abrir el browser y escribir en la ruta de direcciones: <http://localhost:8888/> y visualizara los widgets que se haya creado.

"**start**": "cross-env LOG\_LEVEL=verbose COLOR=true NODE\_ENV=development bb-serve project",



Como se puede observar los del apartado **Custom Widgets** son los que hemos desarrollado y los del **Collection Widgets** son del **collection-retail-banking-2.10.0** que se crearon cuando se creó la carpeta contenedora de los widgets **statics**

A cualquier widget creado podemos visualizarlo dando click en **LAUNCH**



**Nota:** Recuerda crear un theme si no lo tienes, ya que será importante para que desde el ambiente de desarrollo se pueda visualizar los **Styles** de los widgets, además al archivo **.bbconfig** es necesario agregarle la siguiente instrucción después de la última línea del “**serve”: {..**

"project": {

"plugins": "sass,uglify",

"prebuilt": "collection-myportal/prebuilt",

"template": "target/bb-serve-template-blank/package/index.hbs"

}

# Importar widgets a CXP Manager

Antes de empaquetar e importar los widgets, asegúrate que este levantado el servidor del CXP-Manager

# Dentro de la carpeta statics ejecutar los siguientes comandos:

# Para generar el zip: bb-package

bb-package **folder-you-collection** /src --prebuilt=**folder-you-collection** /prebuilt -b sass --exclude-defaults

Par la importacion al CXP-Manager: **bb-import**

bb-import --portal-context=/ package package.zip

Para no estar copiando en las instrucciones se pueden agregar al archivo **package.json**

"**package-ng**": "bb-package collection-myportal/src --prebuilt=collection-myportal/prebuilt -b sass --exclude-defaults",

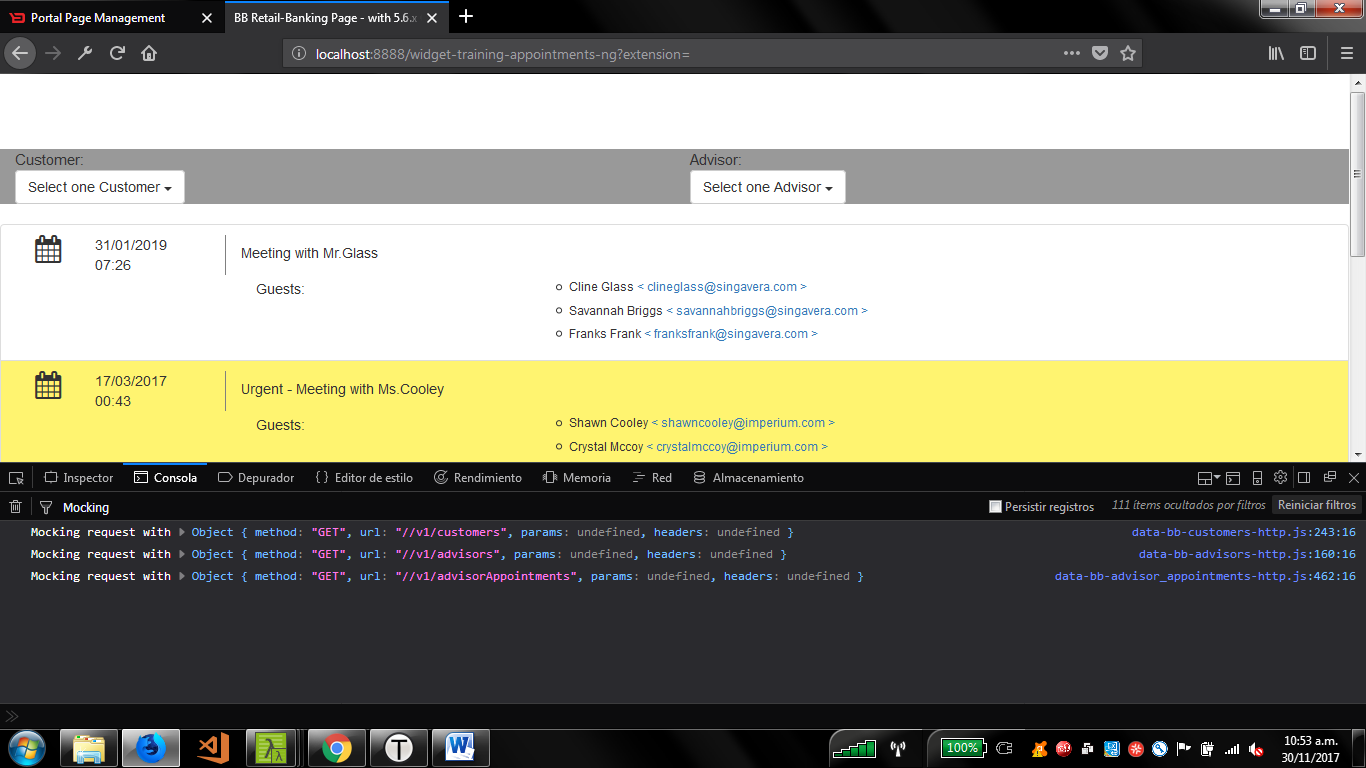
"**import-ng**": "bb-import --portal-context=/ package package.zip",

Posteriormente ejecutar unas instrucciones para generar el zip e importarlo al cxp-manager

**npm run package-ng**

**npm run import-ng**

# Con lo anterior es para probarlo desde un portal existente en el CXP-Manager, pero también se puede probar de forma local.



# Convierte RAML en Servicios de datos

Una vez que los desarrolladores de Back hayan realizado un war con los servicios para este ejemplo, proceder con lo siguiente:

Descargar el war del siguiente enlace <https://drive.google.com/open?id=1DyLVZ5fdjJ1TFkvxOWPDVz86NotSpOAZ> y colocarlo en la carpeta de **war**

Ir a la carpeta de **cxp-demo** y crear un nuevo archivo **bat**  denominado **run-services.bat**  con el siguiente contenido.

java -Xmx512m -XX:MaxPermSize=256m -XX:+UseConcMarkSweepGC -jar jetty-runner-9.3.9.jar --lib lib --config jetty-9.3.9.xml --port 7777 --path / ../war/DemoLoginPortalServices-editorial.war --path /contentservices ../war/contentservices-webapp-5.7.0.0.war --path /orch ../war/orchestrator-webapp-5.7.0.0.war --path /solr ../war/solr-4.10.4.war

Antes de ejecutar run-services.bat es necesario detener el cxp-manager de la consola que ejecutaron, para detenerlo presionar **Ctrl + C** y teclear **Y** y Enter.

Posteriormente hay que cambiar la configuración de archivo **cxp-demo/backbase/backbase.properties** agregándole las siguientes líneas al principio del archivo:

demologin-host=localhost

demologin-port=3000

demologin-context=

Después buscar la línea

#foundation.csrf.protection.enabled=false

Quitar el comentario, en caso de que el valor sea **true** cambiar a **false**.

Luego ejecutar el archivo **cxp-demo/run-services.bat** que levantara el cxp-manager junto con los servicios necesario para la prueba.

**Crear el Widget Login**

Posicionarse en la ruta **statics/collection-myportal/src y ejecutar lo siguiente.**

**bb-generate** widget ng

Responder las instrucciones del generador del widget

**Name** == > **widget-training-login-ng**

**Title** - --- Accept default value ---

**Description** - --- Accept default value ---

**Version** - --- Accept default value ---

**Do you want to create an extension now?** == > ---Yes ---

**Do you want to create a model now,** == > --- Yes---

# Para convierte RAML en Servicios de datos seguir los siguientes pasos:

Extraiga el DemoLogin-RAML-API.zip que se encuentra en la ruta: <https://drive.google.com/open?id=1DyLVZ5fdjJ1TFkvxOWPDVz86NotSpOAZ> en la ruta raíz del proyecto **statics**

Con el **raml** que se tiene, posicionarte en la carpeta raíz **statics** y ejecutar:

**Recuerda la instrucción:**

**RAML a Model Services**

bb-convert raml **name-folder-raml**/**file.raml** --o **name-folder-you-collection**/src –y

**Convirtiendo archivos RAML al Modelo de Servicios**

bb-convert raml DemoLogin-RAML-API/login.raml --output collection-myportal/src -y

# Una vez creado el modelo de raml en nuestro proyecto, es necesario:

Instalar la siguiente dependencia:

npm install -g raml-1-mockup

Abrir la carpeta **statics/ DemoLogin-RAML-API** y ejecutar en consola el siguiente codigo para simular las peticiones GET, POST...

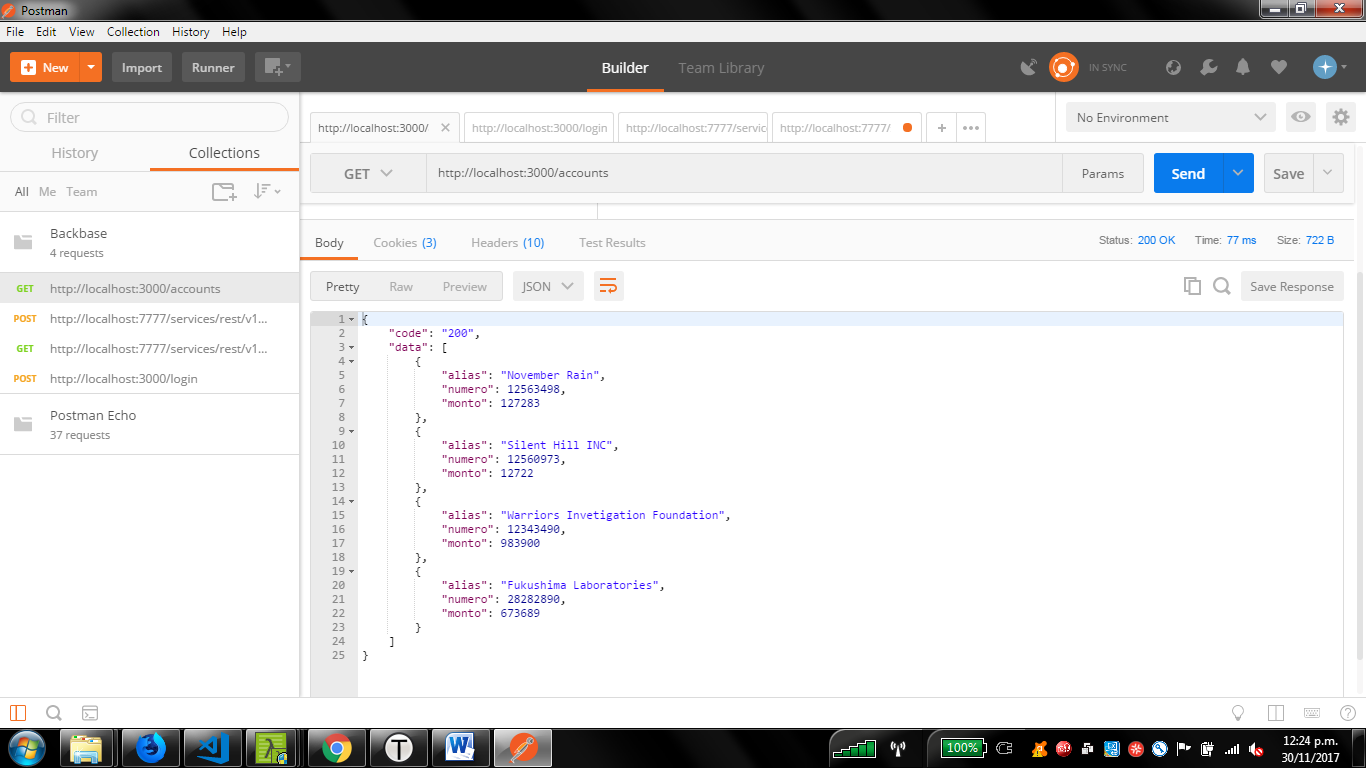
raml-1-mockup login.raml -p 3000

# El comando anterior te mostrara los servicios que tiene el raml y las solicitudes que se estén realizando.

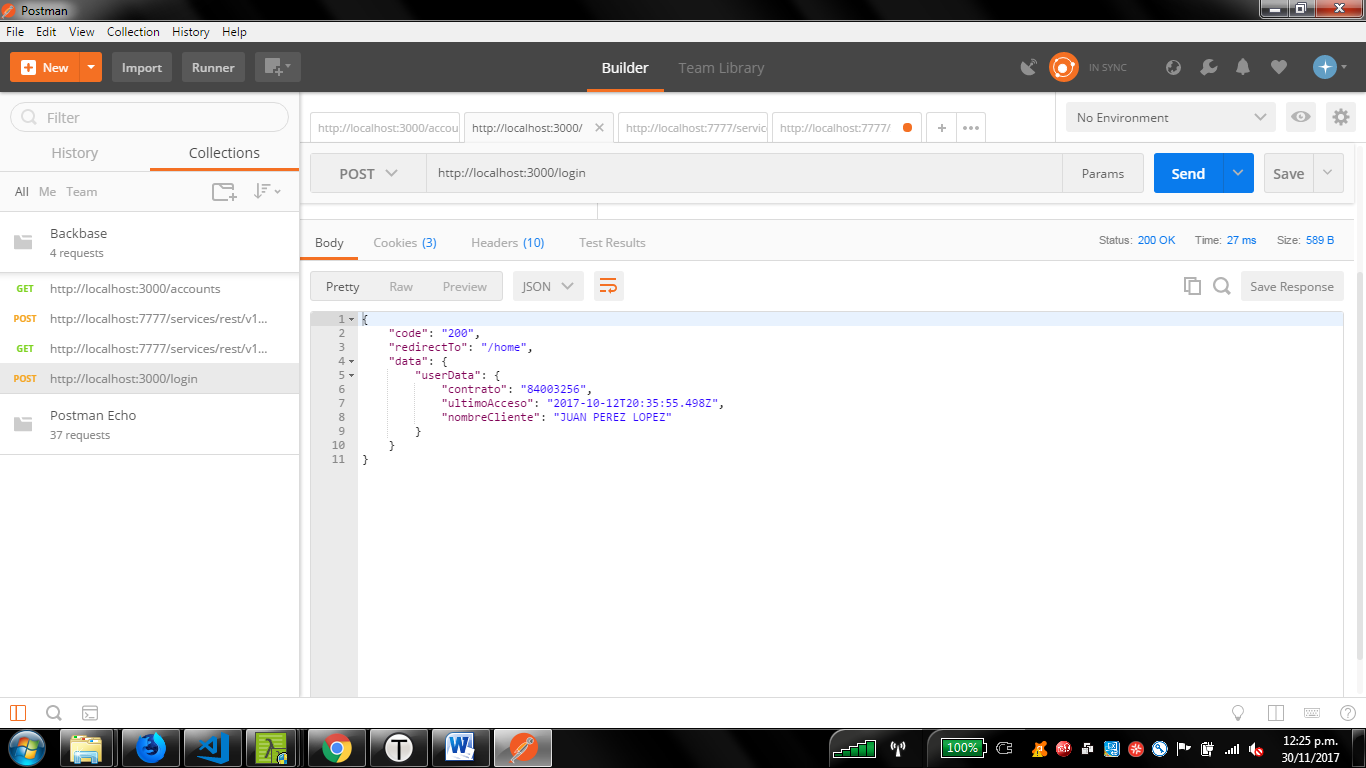
# 

# Para testear el servicio puedes probar con la siguiente rutas en el Postman escribiendo: las siguientes rutas:

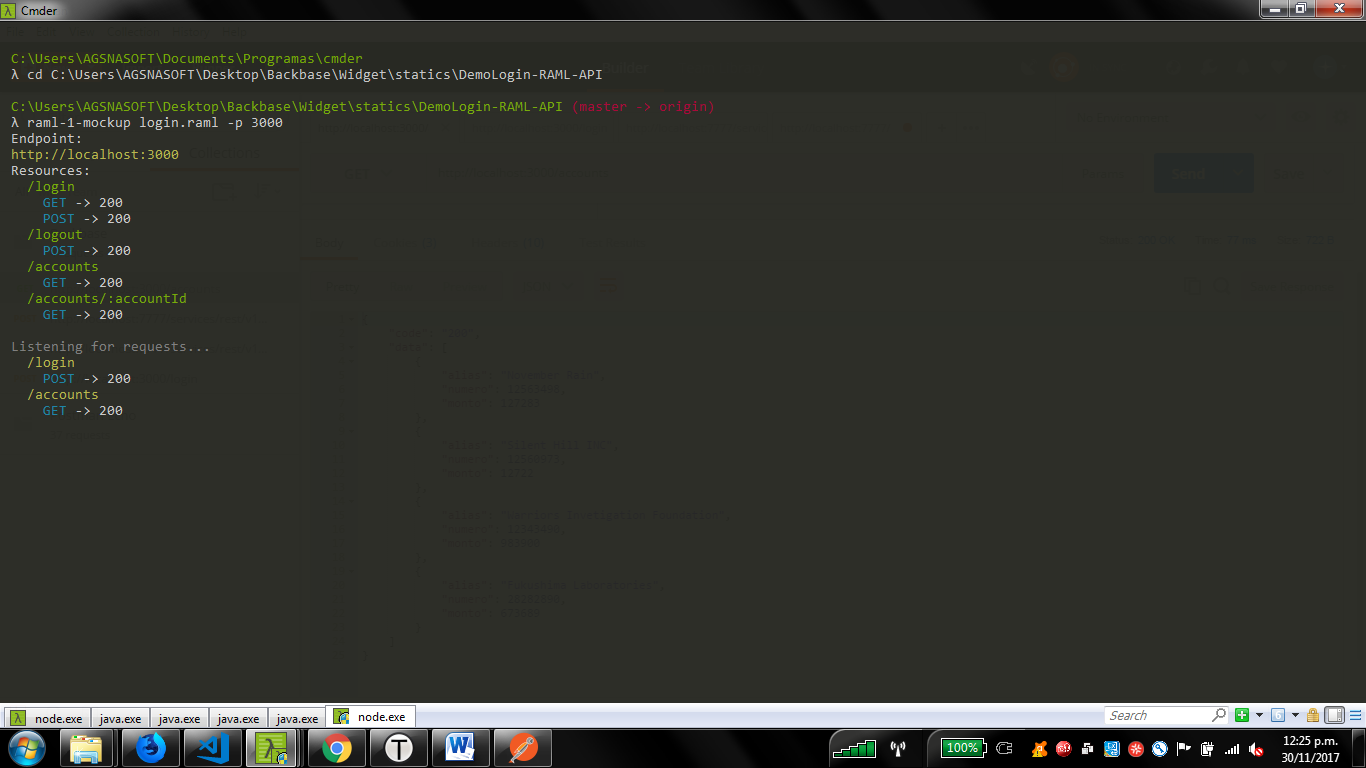
# <http://localhost:3000/accounts>



<http://localhost:3000/login>



Estas pruebas únicamente son para testear las peticiones del raml y como resultado:



Para probar los servicios desde el war las rutas son distintas pero deben funcionar de igual manera.

<http://localhost:7777/services/rest/v1/accounts>

<http://localhost:7777/services/rest/v1/login>

# Cambiar el Modulo de Datos al Modelo de Servicio del Widget

Abrir el archivo de la ruta **statics/collection-myportal/src/widget-training-login-ng/scripts/index.js**

Buscar:

**import** Model **from** './login';

Debajo de la línea agregar:

**import** backBaseDemoLoginDataModuleKey, {backBaseDemoLoginDataKey} **from** "data-bb-backbase-demo-login-http-ng";

Buscar la siguiente línea:

.module(moduleKey, []

Dentro del corchete agregar:

.module(moduleKey, [backBaseDemoLoginDataModuleKey,

Después buscar:

'$q',

Debajo de la línea agregar:

backBaseDemoLoginDataKey,

Abrir el archivo de la ruta **statics/collection-myportal/src/widget-training-login-ng/scripts/login.js**

Cambiar todo el bloque de código del **export default function loginModel** por el siguiente código

**export default function** loginModel(Promise, loginData) {

**function** postLogin(data) {

return loginData

.postLoginRecord(data)

.then(function(response) {

return response.data;

});

}

**function** getLogin() {

return loginData

.getLogin()

.then(function(response) {

return response.data;

});

}

**return** {

postLogin,

getLogin,

};

}

Abrir el archivo **statics/collection-myportal/src/widget-training-login-ng/scripts/controller.js** y agregar la siguiente función.

$ctrl.onPostLogin = ($event) => {

$event.preventDefault();

const inputsData = {

user: $ctrl.data.usuario,

password: $ctrl.data.password

};

model

.postLogin(inputsData)

// .getLogin()

.then(response => {

console.log("response:", response);

})

.catch(error => {

$ctrl.error = errorMessage(error.code);

bus.publish('widget-training-login-ng.load.failed', {

error

});

})

.then(() => {

$ctrl.isLoading = false;

});

};

# En la vista agregamos los input y al form la función $ctrl.onPostLogin para su funcionamiento:

Abrir el archive **statics/collection-myportal/src/ext-training-login-ng/templates/template.ng.html** y reemplazar el contenido por:

<div class="ext-training-login-ng container" ng-controller="LoginController as $ctrl">

<div class="omb\_login">

    <h3 class="omb\_authTitle">Login</h3>

        <div class="row omb\_row-sm-offset-3">

            <div class="col-xs-12 col-sm-6">

             <form name="loginForm" class="omb\_loginForm" action="" autocomplete="off" ng-submit=**"$ctrl.onPostLogin($event)"**>

                    <div class="input-group">

                        <span class="input-group-addon"><i class="fa fa-user"></i></span>

                        <input type="text" ng-model="$ctrl.data.usuario" class="form-control" name="username" placeholder="Username" required>

                    </div>

                    <span class="help-block"></span>

                    <div class="input-group">

                        <span class="input-group-addon"><i class="fa fa-lock"></i></span>

                        <input type="password" ng-model="$ctrl.data.password" class="form-control" name="password" placeholder="Password" required>

                    </div>

                    <br>

                    <!-- <pre>{{$ctrl.data | json}}</pre> -->

                    <button class="btn btn-lg btn-primary btn-block" type="submit">Login</button>

                </form>

            </div>

    </div>

    </div>

</div>

Antes de exportar nuestro widgets es necesario una configuración adicional pero al Modelo de servicios que se generó con los comandos del raml.

Abrir el archivo **statics/collection-myportal/src/data-bb-backbase-demo-login-http-ng/scripts/index.js**

Buscar el siguiente código:

.provider(backBaseDemoLoginDataKey, [() => {

const config = {

baseUri: '/',

};

**Reemplazar** por el siguiente código:

**.provider**(backBaseDemoLoginDataKey, [() => {

const portal = window.b$ && window.b$.portal;

const serverRoot = (portal.config) ? portal.config.serverRoot : '';

const config = {

baseUri: `${serverRoot}/services/rest`,

headers: '',

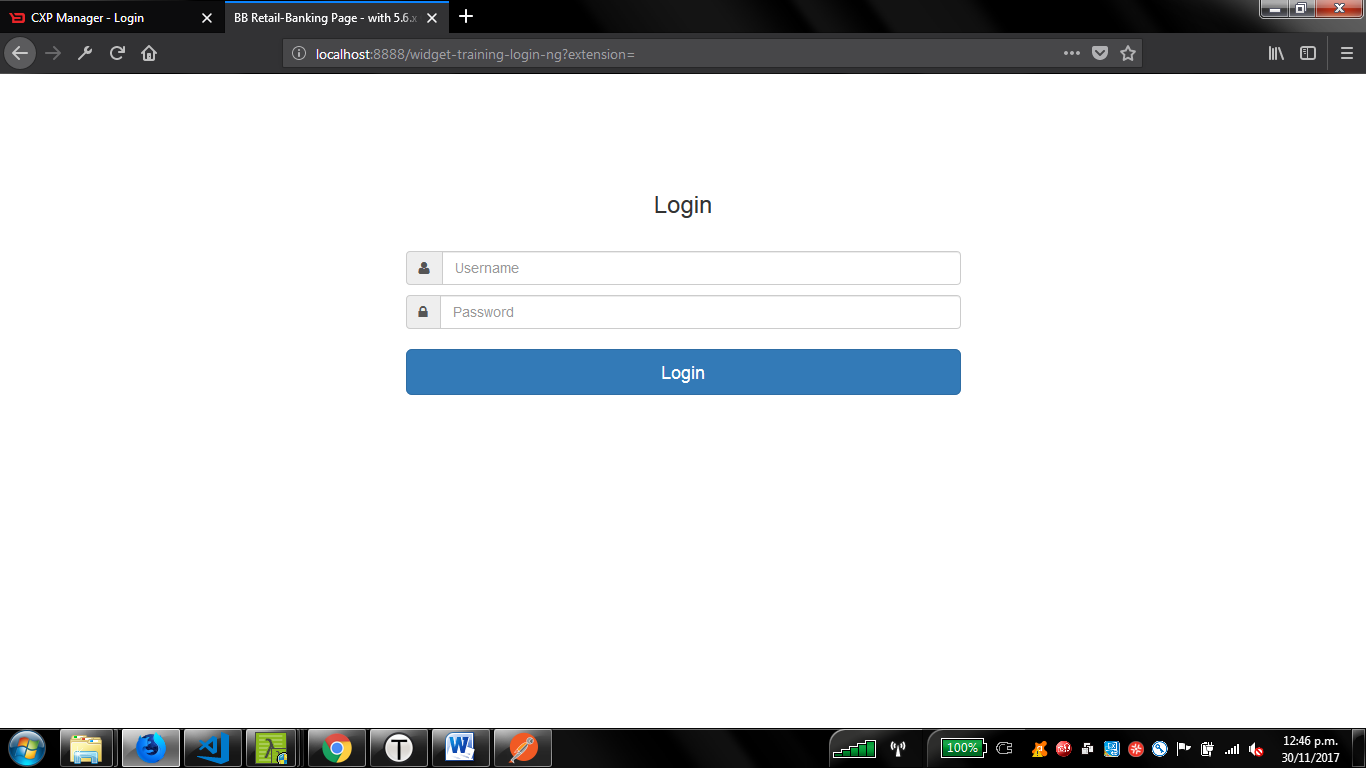
};

Ahora ya tenemos listo el widget para probarlo, esto funciona únicamente exportándolo o al CXP-Manager **NO** de forma local.

**npm run package-ng**

**npm run import-ng**

Creamos una página nueva para arrastrar nuestro widgets de Login y probar su funcionamiento y quedando de la siguiente manera en nuestro portal, recuerda asignarle un **Theme name** a tu página para que tome los estilos diseñados para este widgets.



**Nota:** Recuerda puedes ver en consola del browser la respuesta de la petición del formulario del login y puedes escribir cualquier cosa en usuario y password, ya que no se tiene validación en el servicio.