Lesson 30: Outbound API and JSON

Created by Charl Viljoen, last modified on Jul 21, 2020

> *The Shop Owners and Farmers want a communal address book that they can contribute to and will include people likely not on the system so as to easily share contact information with each other and recruit new users.*

Lesson Outcomes

By the end of this lesson you should know how to use several forms of the DSL outbound API to perform authorised requests to outside applications including GET, POST, and DELETE.

New & Modified App Files

`./model/objects/AddressBookListing.mez`

`./web-app/lang/en.lang`

`./web-app/presenters/AddressBookMgmt.mez`

`./web-app/views/AddressBookMgmt.vxml`

Helium 2 Service

In this tutorial we are going to use our own Helium 2 service to act as the outside application which will be contacted using the outbound API. Please ensure you have completed the Helium 2 Beginner's Tutorial until at least Part 7, and that your service is running as expected before continuing.

AddressBookMgmt

Essentially this will be a CRUD view for objects that are persisted outside of our Helium 1 database and rather in our Helium 2 database using the API services between these two applications to manage these objects. If you have not yet set up your Helium 2 service, please do so before continuing with this tutorial.

The view we are going to create shouldn't look much different from anything we have covered so far:

AddressBookMgmt.vxml Expand source

Firstly we allow all the users to see this `<menuitem>` so as to create a communally contributed address book tool.
A `<textfield>` is then used to take input from the user and perform a search on that

input.
The `<textfield>` displays all of the listings found based on the user's input along with an `action="deleteListing"` and `action="updateListing"` `<rowAction>`.
Lastly the page has several text inputs and a `<submit>` which will be used to update or create a listing.

We also need a new non-persistent model object to serve as a `<collectionSource>` for our table and temporarily house the results returned from our search:

AddressBookListing.mez Expand source

You should be familiar with the view components used above as well as how to create non-persistent model objects.

Remember to update the **`en.lang`** file with the new translations and see the attached source code for any omitted parts if necessary.

Outbound api:get and AddressBookMgmt

We will now look closer at the functions used in this management view starting with the search function:

presenter snippet (AddressBookMgmt.mez)

```

void searchAddressBook(){
  if (searchParam != null && searchParam != "") {
    foundResults = null;
    foundResults = getAddressBookListings();
  } else{
    Mez:alertError("alert_error.no_input_given");
  }
}

AddressBookListing[] getAddressBookListings() {

    string baseUrl = "http://127.0.0.1:8090/3ec84978-b
    string listingEndpoint = "/1/listing/";
    string url = Strings:concat(baseUrl, listingEndpoi
    string userName = "user1";

    AddressBookListing[] result;

    try {
        MezApiRequest request = MezApiRequest:new();
        request.url = url;
        request.credentials = userName;

        MezApiResponse response = api:get(request);
        int responseCode = response.code;
        string responseMessage = response.message;

        if(responseCode < 200 || responseCode >= 300)
            Mez:alertError("alert_error.listing_get_fa
            return result;
50             }
51
52             result = createAddressBookListingsFromJson(res
53             showResults = true;
54             return result;
55         } catch(ex) {
56             string exceptionMessage = ex.message;
57             Mez:alertError("alert_error.listing_get_except
58             return result;
59         }
    }
```

The `searchAddressBook()` function validates that we have received input before calling any API to ensure that we do not needlessly call the service without the correct parameters. The `getAddressBookListings()` function on line 28 is our first brush with the outbound API.

Helium has two added non-persistent objects used with the outbound API functionality to accomplish these calls. The one, **`MezApiRequest`**, holds the values for your request in its attributes such as the `url` that will be used as the endpoint for the service being contacted, the `body` of the request if needed, which `credentials` should be used during the request (more on this later), and finally any `headers` that might be needed by the service. The second, **`MezApiResponse`**, holds values that can be expected in a standard REST API response such as the `body` of the response, a `code` that represents specified HTTP response codes, a `message` describing the HTTP response code if provided, and lastly a boolean `success` attribute indicating whether the response code is in the 2xx range (succces) or not. These are used along with the following built-in functions that are provided for making outbound API calls. Note the api namespace:

api:get	Performs the outbound call with a GET HTTP method.
api:post	Performs the outbound call with a POST HTTP method.
api:delete	Performs the outbound call with a DELETE HTTP method.
api:put	Performs the outbound call with a PUT HTTP method.

The functions all expect a single argument of type **`MezApiRequest`** and return a single instance of **`MezApiResponse`**.See more about these objects and functions here.

We have a string value that represents the `baseUrl` of the endpoint we will be using and we concatenate that with the endpoint for the specific resource we will be using, `listingEndpoint`, as well as the parameter for the search which will be the user's input in this example, `searchParam`. We alse define which credentials we want to use when making this call, although note that these credentials have to be set up for our application. There are two ways to add credentials to your application, use the core Helium API, or use the user interface provided for the application under the app admin page on the Helium core application. You can read more about the credentials management here. Remeber to add your credentials according to what you have defined during your Helium 2 service setup. The user you will use to make this call will have to have at least `READER` privileges.

Pay attention to how the **`MezApiRequest`** is created using the values we have just discussed and then passed as the parameter for the `api:get()` built-in function call made on line 42. On this same line you can also see how the returned object is assigned to a variable of type **`MezApiResponse`**. The response is then used to display a success or error message to the user as feedback on their response. The body of the response now holds the result of our request made to the service in JSON format. In the next section we will look at how we can interpret that JSON body.

jsonGet and AddressBookListing

On line 51 in the API function above, the response body is passed to the `createAddressBookListingFromJson()` function. While the response is a single **`json`** object, we know this reponse will be an array of 0 or more elements so we convert it to a **`jsonarray`** variable before converting it again to an array of **`json`** objects (**`json[]`**). Look at how these variables are implicitly cast, simply by assigning the variables to other variables with different types on lines 72 and 73 below. Also note that casting from **`jsonarray`** to **`json[]`** is a relatively expensive procedure and should be avoided for large arrays.

presenter snippet (AddressBookMgmt.mez)

```
61
62  AddressBookListing createAddressBookListingFromJson(js
63      AddressBookListing listing = AddressBookListing:ne
64      listing.id = listingJson.jsonGet("id");
65      listing.name = listingJson.jsonGet("name");
66      listing.address = listingJson.jsonGet("address");
67      listing.age = listingJson.jsonGet("age");
68      listing.mobileNumber = listingJson.jsonGet("mobile
69      return listing;
70  }
71
72  AddressBookListing[] createAddressBookListingsFromJsor
73      jsonarray listingsJsonArray = listingsJson;
74      json[] listingsJsonCollection = listingsJsonArray;
75
76      AddressBookListing[] result;
77      foreach(json jsonListing: listingsJsonCollection)
78          AddressBookListing newListing = createAddressE
79          result.append(newListing);
80      }
81
        return result;
```

```
}
```

We then just loop over this **`json[]`** and call the function above it on each element in the array which creates a new `AddressBookListing`. The DSL built-in function `jsonGet()` takes a single string parameter which is used to retrieve and return the value for that specific key in the **`json`** object that it is called on. In the example above we can see how it is used to retrieve the attributes for the listing and then assign them to the newly created `AddressBookListing`. Looping over the array in the response body creates our collection source for our table.

Outbound api:post and jsonPut

We can now take any input given to the application by the user and create/update a listing using an outbound API post call.

presenter snippet (AddressBookMgmt.mez)

```

void submitListing(){
  string baseUrl = "http://127.0.0.1:8090/3ec84978-b8
  string listingEndpoint = "/1/listing";
  string url = Strings:concat(baseUrl, listingEndpoir
  string userName = "user2";
  json body = "{}";
  uuid id;

  if (selectedListing == null) {
    AddressBookListing obj = AddressBookListing:new()
    id = obj._id;
  } else {
    id = selectedListing.id;
  }

  body.jsonPut("id", id);
  body.jsonPut("name", name);
  body.jsonPut("address", address);
  body.jsonPut("age", age);
  body.jsonPut("mobileNumber", mobileNumber);

  try {
      MezApiRequest request = MezApiRequest:new();
      request.url = url;
      request.credentials = userName;
      request.body = body;

      MezApiResponse response = api:post(request);
      int responseCode = response.code;
      string responseMessage = response.message;

      if(responseCode < 200 || responseCode >= 300) {
          Mez:alertError("alert_error.listing_get_fai
      } else {
        searchAddressBook();
        Mez:alert("alert.uploaded_data_saved");
      }
  } catch(ex) {
      string exceptionMessage = ex.message;
      Mez:alertError("alert_error.listing_get_excepti
  }
}
```

The API request is built in mostly the same fashion as before with slightly different endpoints, however, we are also now using a different set of credentials that has `WRITER` privileges and we will add a `body` to our request where there was no `body` before. The `body` attribute is of **`json`** type and will contain the attributes for our listing as expected by the service we are calling. The `jsonPut()` built-in function is used to add fields and values to a **`json`** variable which are passed to the function as parameters. On lines 110-114 we can see how the values received from user input with their respective fields are being added to the **`json`** body of our request. This is then added to our **`MezApiRequest`** object which will be used with the `api:post()` built-in function this time.

Note that the `action="updateListing" <rowAction>` only takes the listing selected and populates the input fields used to create a listing. When submitting these values then the lines 102-107 evaluates whether it should create a new **`uuid`** which the service will use to create a new listing, or whether it should use the selected listing's uuid indicating to the service that this listing should be updated with the values in the `body`.

Outbound api:delete

We also have the `api:delete()` built-in function that is called similarly to all of the previous methods only updated to the corresponding REST resource it will utilise and using the credentials of a user that has `WRITER` privileges.

presenter snippet (AddressBookMgmt.mez) Expand source

Further Reading

There are several more advanced concepts regarding both the Outbound API and Helium JSON handling that have not been covered here. It is recommended users read up more about these topics here:

Outbound API

Native JSON Types

Lesson Source Code

lesson_30.zip

No labels