Lab 4 (DH2642 only): Dynamic data

Goals

- work with real data
- use REST APIs to access data
- process asynchronous requests (AJAX)
- insure good interaction (delay and error handling)

Assignment

In this lab your task is to extend your prototype implementation of dinner planning service from Lab 3 to include the data from a real data source. To achieve that you will be using BigOven REST web service that provides a database of recipes.

Step 0. Setting up and exploring the data

Most web services that provide Application Programming Interfaces (APIs) to their data require you to acquire a API key (or set of keys) that will uniquely identify you towards the web service. They do this so that they can insure security (e.g. blocking requests that are identified to be malicious) but also to be able to provide various access levels, giving only some data/features to free accounts and more to the paying users. Luckily the BigOven provides enough of information for our labs in their free version.

To get access to your API key you need to:

1. Follow the required steps to register at BigOven
2. Get the API Key from Developer console (when asked to provide website address you can put http://YOUR_KTH_USERNAME.kth.se)

After having your API key you can start exploring the data. In general it is good to read the documentation and try test queries before you start implementing it in your code. You might even need to reorganize your code or do it differently than planned depending on how the data is structured and API architected.

The documentations of APIs (as most documentation) is often not complete or perfectly clear. This is the case with BigOven API as well. That is why you will need to test and explore the APIs. The easiest way to explore an API is to use an API browser like apigee. In such browsers you can setup all the needed parameters and headers and see the results that are returned from the web service.

To explore BigOven API data you will need to:

1. Find the endpoint (address) of the data you want to explore. For example, for Recipe search the endpoint is http://api.bigoven.com/recipes?
2. Add the address to the API browser
3. If you click Send now (try it!) you will get the results, but at the bottom of the result you will notice an XML message saying *Invalid API Client*. There are several issues we need to correct here
4. To get the messages in JSON (JavaScript Object Notation) which is easier for us to use in JavaScript (the dishes in previous labs were in that format) instead of XML, according to documentation we need to also pass *Accept* header with *application/json* value. You can API browser you should see the Headers option just under the Resource address. Add the needed header/value combination and lets' see what we get.
5. We are still getting *Invalid API Client*, but at least in JSON format. To fix the error, we need to pass the api_key you got earlier in the exercise. Add api_key=YOUR_KEY_VALUE to the and of the endpoint address, i.e. http://api.bigoven.com/recipes?api_key=YOUR_KEY_VALUE **Note:** It takes up to 1 hour from signing up before your API key will be active. If you were really fast and you are still getting an error Invalid API Client error, you might need to wait a bit more.
6. Done? No. There should be a new error message showing in the results: *Search query not completed. Paging parameters required ('pg', 'rpp')*. Even though the Recipe search didn't specify that these parameters are required, if we check Paging documentation, we see this indeed to be true. If you add for example pg=1&rpp=10 to your endpoint address you should finally start getting some real recipes in results :) Now you can play with other parameters from search documentation (like title_kw and any_kw) to see how you can influence the results.

Now that you got the results, you can explore how it is organized. Compare it to dishes data you had in the previous labs. Are they organized in the same way? Do you have all the information you need? It is likely you will need to change some of your code to fit the BigOven data.

Consider if there is any other data you will need and for your application and check the documentation to see how you can access it.

Step 1. Getting the data in your code

In previous labs you used getAllDishes function to find the dishes you needed. That function was getting the dishes from a static source -> the dishes array in your model. Now it is time to use the BigOven data.

As we want our application to be modern, web 2.0 application, we do not want our code to block the application while it gets the data from the web service. Therefore, we will use AJAX to issue asynchronous calls. Check out the lecture notes to reminder yourself about asynchronous calls.

You should update the getAllDishes and getDish function to use BigOven Recipe search

API. You can check the documentation for the example of how to use jQuery ajax method to get the data. For now do not worry about your interface being broken, instead use debugging console and play with your model code until you get the results from the dishes as you expected.

Since AJAX is asynchronous, we can't just update the getAllDishes/getDish methods and use them in the same way as before. Why? What would be the problem? If you are not sure you know the answer think about when you do you expect the data from methods (as soon as you call them) and when would you actually get it from AJAX call (some time later - in a callback function). There are several ways we can address this. To be consistent with the rest of the application the best way to solve this problem is using the Observer pattern once again. Therefore, in the "success" callback of the AJAX call we want to call "notifyObservers" so that the views can update themselves with the correct data. Even if you didn't pass any arguments to notifyObservers in the previous labs, now it is time to do so. We want to pass the data we received from the service.

In your views that display search results you need to check when are you calling getAllDishes method and make sure that you change your "update" method so it handles the dishes returned from the web service.

Remember that the data in BigOven is organized differently than dish data was before (e.g. "Title" instead "name" for the dish/recipe name) so your interface will probably not work immediately. Therefore, try first using the debugging console to log your results.

Note: Since BigOven search API requires paid features to use the category filter, you will need to use the category name (i.e. appetizer, main dish, dessert) in your keyword search instead. Do not worry if you are not getting perfect results.

Once you made sure that you are getting results you wanted in your getAllDishes function and getting it in your interface you will need check and slightly modify the rest of the code so that everything is working as it did before.

Note: Since BigOven does not the price for the ingredients, we will "pretend" that the price of each ingredient is 1$/SEK (whatever currency you used) per unit. That means if you need 1 egg and 10 ml of oil the total price will be 1x1 + 1x10 = 11.

Here is a bit of help to understand how your application should work:

1. User enters text into search
2. Your controller reacts to event and calls "getAllDishes" method passing the search parameters
3. getAllDishes makes the AJAX call to BigOven
4. In your success callback (that is triggered when the data is returned by the service) you call "notifyObservers" in your model and pass the result data as the parameter
5. in your search result view's update method you interpret the results and display them to the user

You will have similar steps when you are getting the details of the recipe (i.e. when

updating the getDishes code).

Step 2. Insuring good interaction

Having real data in the application is a nice improvement, however there are certain issues with the interaction. The search results are not instantaneous as they used to be. Depending how many results (the rpp parameter) you get from the BigOven, and your internet connection, it might be few seconds before the results are in and the view updates. In good and reactive web application we need to take care of such cases so that the users know what is happening and feel good using our application.

In this step you will need to implement some form of indication to the user in the cases when your application is still waiting for the data. How you present this information to the user is left to your choice. For example, you can show a message (searching, loading, ...) in the place where the result is expected, have a indicator (like spinning circle), or any other way you know or can come up with.

If we consider the 5 points your application goes through from the previous step you will need to modify:

- step 2 to show the indication to the user
- step 5 to hide the indication

Another important thing to consider is what if there is some error. The simples case can be if the web service is not available (internet connection breaks), but your users could also enter wrong data (that is not allowed by the web service) or there could be any other reason. For good interaction it is important that you let the users know that something an error has happened and, if possible, guide them on how to fix it.

For our application, you can simulate failure by either turning off the internet on your computer or using the Chrome Developer Tools in Device mode and choosing the "Offline" option in the network settings. In such cases, when a web service is not available or it returns an error, the AJAX call will not end up calling the success callback you specified before. Instead you need to create a "error" callback. In this callback you want to also call notifyObservers, but instead of passing the results you want to somehow indicate that you have received an error, so that your view knows it should display an error message.

What to deliver on 6th Mar

When you have finished with developing, just push your changes to GitHub and present your results to assistants in one of the Lab sessions.

Visa tidigare händelser (7) ›

Erik Forsberg har tagit bort sin kommentar

... eller skriv ett nytt inlägg

Alla användare med KTH-konto får läsa.

Senast ändrad: 2015-02-17 15:11. Visa versioner

Taggar: Saknas än så länge.

Följ denna sida Anmäl missbruk

http://www.kth.se/