Lab 7: Remote Data

In previous courses you have learned how to consume data on the client side from server-side web services. In this lab you will apply that same concept to fetch, parse, and store data directly from server to server. This practice of transforming data for your own use is extremely prevalent in the business-to-business space and will undoubtedly come in handy in your future work as a server-side developer.

Objectives & Outcomes

Successful completion of this activity will show that you can:

- Fetch serialized data from remote web services using server-side code.
- Deserialize/Parse data using server-side code.
- Implement very simple server-side caching logic.

Level of Effort

This activity should take approximately 4h to complete. It will require:

- 45m Research
- 5m Prep & Delivery
- 3h Work

If you find that this activity takes you significantly less or more time than this estimate, please contact me for guidance.

Instructions

This project should go into a `07_remote` folder. For this lab you will choose *either* PHP or ColdFusion—you don't need to do both.

Development

You're going to build a web service consumer. The basic process will be:

1. Research a 3rd-party web service. This *cannot* be an RSS feed—it needs to be XML or JSON in some other format.

2. Write code to fetch the remote data. Make sure you practice defensive programming and include at least the following sanity checks: HTTP status codes, MIME type, data format, and data content.

3. Write code to deserialize/parse the data. **Do not** parse this data by hand. Use the built-in XML or JSON functions of your chosen language. If the data is malformed and cannot be parsed using the built-in functions you will need to use a different web service.

4. Pick at least 3 data elements that you find useful from the data. Build a database table that can store these elements with some kind of unique ID and a "last fetched" timestamp, for a total of at least 5 columns. For example, maybe the web service provides a list of products, from which you could store a product ID, title, description, thumbnail URL, and fetch date. Insert the parsed data into your table.

5. Refactor your fetch code to be smart enough to *not* re-fetch your data more than once per hour. That is, if the most recent fetch date is less than an hour ago just used the data from the table. *Hint:* look into MySQL's `CURRENT_TIMESTAMP` constant and `NOW()` function.

6. Refactor your fetch code into a Model/Service class. The caching logic should be built into this class.

7. Build a View that shows off the fetched data in some nicely-formatted way.

8. Write a simple Controller that just calls the Model/Service and View.

Above and Beyond

If you implement this lab in both languages you will definitely see the underlying patterns, not just between server-side languages but also betwen server- and client-side languages. You could also try to cache non-string data, such as dates and numbers, or more complex structured data that would require multiple database tables.

Git Commits & Video Reflection

The same rules about Git Commits and Reflection Videos apply for this Lab as for previous labs. Review the Lab 1 documentation to ensure you meet all of the required points. Your required questions are mostly the same, with small differences:

1. What didn't go as well as you had hoped? Where did you run into problems with the lab?
2. What went better or was easier than you expected? What went right?
3. **What are examples of situations where it makes more sense to fetch remote data on the server-side instead of the client-side? What about the reverse?**
4. How do you think what you did in this lab is going to help you in your future school work, career, or life?

Where should you be?

The basic concepts of fetching remote data should be very familiar, given what you've already done in JavaScript and ActionScript. The syntax for server-side code is a little different, but it shouldn't be completely unfamiliar. Don't worry about memorizing the details, but you should remember the basic sanity checks: HTTP status codes, MIME types, content formatting, data validation.

Deliverables

Make sure your source code, assets, and reflection video URL are committed to the repository. Push your changes to the server before the start of the next lecture.