"team" Technical Report for World Crisis Website

Group name: team

Group members:

- Stephen Chiang
- Aaron Stacy
- Blake Johnson
- Jason Brown
- David Coon
- Qunvar Arora

Group URL: http://tranquil-springs-4182.herokuapp.com

Introduction

People who are interested in learning about world crises do not have a central repository of world crisis information. The average person would have to conduct a series of online searches in an patchwork attempt to gather related information. This approach is time consuming, inefficient, and its expensive time commitment detracts others from gaining awareness into major world issues. An ideal solution involves a single website that can provide easy-to-access information on world crises. This website would act as a hub of information for the person to learn about influential people, organizations and media related to the crisis of interest.

The Use Cases

A person might want to gain information on a particular world crisis event, such as the human impact, economic impact, resources used to address the issue, and ways to help those affected by the crisis. They might want to find out which influential people are involved in the event To learn about the organizations involved with the event To access media resources related to the event, people, and organizations. They may also want to see media resources include online articles, blogs, images, videos, social networks, and maps.

One of the difficulties would be identifying links between the data. It's relatively easy to find a list of people associated with any given event, but then given any one person, finding other events to which they are connected could prove more challenging.

Two primary roles emerge:

1. An author, who knows about the information and wants to share it.
2. A researcher or casual user who merely wishes to consume the information.

These roles may of course overlap in the same person, but they provide a basis for building a system to help gather and amalgamate information on world crises.

Our task

We were tasked with creating a data format that described world crises, researching a number of crises along with associated people and organizations, and making a web application capable of importing data about crises in the designated format, displaying the data, and exporting the data back into the format.

Our End Goal

As the public becomes more educated on this issues, they are more likely to take constructive action to address and/or resolve these issues (e.g. elect representatives, form grassroots support campaigns, volunteer, donate money)

Design

We created the data exchange format in XML. The main benefit of XML in this situation is it provides very powerful capabilities to specify and validate the data format. In order for information systems to transfer data among themselves, they need a common format that is both flexible and well defined. If the data is not flexible, it will not be able to capture all of the information surrounding a topic. On the other hand if it is not rigorously defined it will be too difficult to design systems that can share the data.

One of our first tasks was rallying support around a shared XML schema. We started with an initial schema authored by Vineet Keshari. We then built consensus among other teams in the class by discussing changes in GitHub issues. In the end Vineet was responsible for most of the schema desgin, and the groups participating felt comfortable moving forward with it. In order to make it as easy as possible to drive adoption, Stephen Chiang made a shared database website. The website allowed groups to easily enter their information and get an XML dump that validated against the shared schema. It also provided a really easy way to coordinate XML ID's, hopefully making our jobs easier in future projects.

Implementation

Once we had fleshed out the initial pass at the schema, we implemented [a converter from database info to XML][to_xml]. This was an excellent opportunity to make use of object-oriented design. The three different data types (crises, people, and organizations) all had a lot in common, however they had a few key differences. We used inheritance to define a base set of functionality, and then we over-rode it and extended it where necessary.

We also implemented a means to import data from XML into the database. This was an opportunity to make use of the dynamic nature of Python -- since the algorithm essentially walked through a set of elements, it was easy to define a base class that dynamically dispatched handler methods based on the tag it encountered.

Testing

Unit tests were written to test all the functionality of our JSON creating code, our XML parsing and interpreting and validating code, our database reading code, and our import/export code for interfacting with our database.

The first suite of tests is for our JSON code. The first set checks for proper creation of parent nodes that allows for correct reading of dictionaried data, and served to familiarize us with calling conventions. Next, we test adding children to a parent node and similar calling conventions for getting the data back out of those nodes. Finally, we tested the make_json function and its compatibility with the json.loads command from the json library.

The second suite of tests is for our XML interpreting and validating code. First set tests if we're using the right data (sanity check) and then we see if we're getting the right tag from using the .strip() class function. Second set tests for validation on the code - as in it's still valid XML after being parsed several times.

The third suite of tests is for our database reading code, which only goes in and tests that we are correctly pulling XML from the DB and able to interpret it like we expect it to behave.

The last suite of tests is for our import/export code. The first test checks for a proper redirect response code to send user to login. The second test checks that we can get the correct response code and then in the same declaration that we can get a different response code (that's still correct!).

Our test suite uses a sqlite3 backend instead of MySQL, which is able to run the tests 5 to 10 times faster than MySQL since it does not persist the results to disk. Since our codebase follows good design principles of dependency injection, we are able to swap the database engine out and still get good test coverage over our codebase.