Gummy Pandas

Members:

Arturo Lemus
Russell Jahn
Honghui Choi
Young Seo
Hoang Pham
Zhen Yang

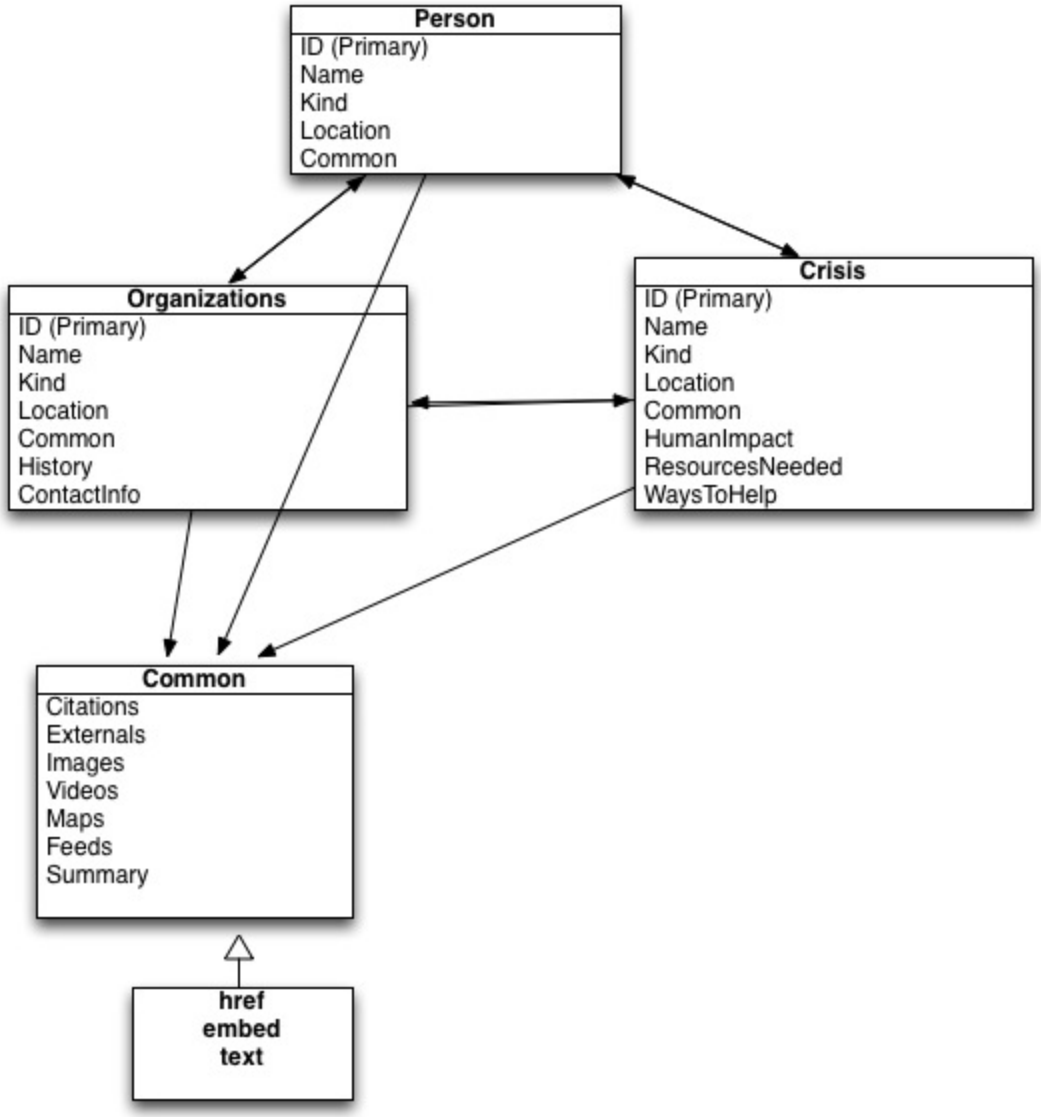
Introduction:

If all the tragedies can stop for one day, that day would be the best day on Earth. But unfortunately, that is impossible. To help us understand the impacts and consequences of the crises are happening around the world, our website was created. Hopefully it will be used to help our generation get a sense of the horrible things that mankind have been through.

To navigate through the website, you can click on each of the different Crisis, Organization, and Person links from the home page. Each of these pages will contain information in a list format. Typically, someone will visit the site for academic research, to get involved to help with a crisis, or to inform themselves out of personal interest.

Design:

The XML Schema only requires an ID and a name for every crisis, organization, and person. Other than that, everything else is optional. There is also a complex type called common that includes general information like citations, external links, images, videos, maps, feeds, and summary that you can add into the big 3 (crisis, organization, person). For crisis, you can have the IDs of people and organizations that are involved in it. Then you can add in date, time, and the locations of the crisis. Furthermore, there are human impact, economic impact, resources needed and ways to help field that you can fill to give the reader a deeper understanding of the crisis. For organization, the optional elements are crises that it is involved with and the people that are in it. Plus you can fill in in more information about the organization including its type, locations, history, and contact info. Finally, we have person type that have elements like crises and organizations that the person gave a hand to. Additionally, the person's job and location are there if you feel like you want to tell everyone what this person is up to. To make sure that each elements from the big 3 are unique, we provided 3 simple id types called CrisisIDType, PersonIDType, and OrgIDType. These 3 id types will require either ORG_, PER_, or CRI_ in front and then the unique ID of organization, people, and crisis at the end.



Implementation:

To import the xml files, first we use elementTreeWrapper.getTree() to get the elementtree object. Then we find the elements with tag Crisis and do the following:

1. Get ID and name by making a dictionary, then get all the items of the element then put it into that array. Finally, save the ID and name from that dictionary into new variables to access them later.
2. Find elements with tag Organization and People. Since a Crisis can have many people and organizations, we use a while loop to get every people and organization that are involved in the crisis append them into a list.

3. Find elements with tag Kind, Date, and Time then save them into their own String varialbes
4. Find elements with tag HumanImpact, EconomicImpact, Locations, ResourcesNeeded, WaysToHelp. Since there can be elements in the above tags that are surrounded by <li>, we make sure to get all of them by using a while loop and append them into their own arrays.

For People and Organization, we do the same 4 steps above but with different tags. In the schema we use, we have a complex element called Common that includes some common information. Since the big 3 elements all have Common we decided to make a function to extract the information that are needed for Common and call it at the end of the big 3 elements. With in the Common function, we did the same thing we did in step 4 but with tags: Citations, ExternalLinks, Images, Videos, Maps, Feeds, Summary.

Now that we have all the information we need from the xml file, we need to put all that into our django models. To do that, we make an empty list called models. At the end of each of the big 3 elements, we append (Crisis or Person or Organization (data we have)) into the models list. Finally, we go through our models list using a for loop and call save() on the Crisis, Person, and Organization objects.

To export our Django models to a xml file, first we make our root called “WorldCrises” by using ET.Element(“WorldCrises”). Then we get all the objects in our Crisis Table. For each of the object, we make it the child of “WorldCrises” thanks to the ET.SubElement command and make it called Crisis. Then we add other information that we have on Crisis as children of Crisis. After that, we do the same thing for Person and Organization. Next, we indent the file to make it look pretty and make it into a tree by using ET.ElementTree. The reason we do this is because ElementTree has a function called write that can write out the xml file with the given tree.

Testing:

At the end of the project, we created some units for the functions that we wrote in xmlParser.py. This file is the crucial part of the project. It extracts the data from the XML and store that into database. The functions involved in accomplishing this job in our xmlParser file are 'getTextAndAttributes', 'getCommonData', 'elementTreeToModels', 'modelsToDjango'.

For 'getTextAndAttributes' function, we created a short sample XML, and casted it into a tree and passed it to 'getTextAndAttributes' function as an argument, which returns a dictionary. Then, we called assertions to check if the contents called by the keys match what we expected.

For 'testgetCommonData,' we again created a short sample XML and turned it into a tree,

which returns a list of dictionaries. We compare the expected output to the actual output for each content of the keys and the array itself that contains dictionaries.

For 'indent' function, we used a sample XML code, and called the fuction on that. Then, we compared the output of the function to the expected output.

For 'elementTreeToModels', we created a short XML. Then create an element tree for the XML and pass this tree to this function. The output is a list of models. We compared the output of the function with the output we expected.

For 'isNotDuplicate', we create a dictionary and pass this dictionary and a value to the function. If the the value is already in the dictionary, the function would return false, otherwise return true.