

Great Moonstone Oxen of the First and Forsaken Oceans

Taylor McKinney	Matt Stringer	Cesar Cantu
Spencer Reynoso	Ben Bowley-Bryant	David Denton

August 7, 2013

INTRODUCTION

What is the problem?

It is often easy to forget the many crises occurring throughout the world. World Crisis Database is designed as a framework to effectively gather data related to recent crises. WCD then presents the crisis data, including victims, organizations, and political players involved, in a user friendly, digitally logged format.

What are the use cases?

This site makes use of worldwide internet access by bringing news regarding crises and by promoting help for victims. This site can be a effective resource for anyone who wants to be informed on current events. Furthermore, individuals who have a desire to help can easily find contact information for people and organizations involved and from there find the best ways to get involved, including information on how to donate to the charities or volunteer at a local site.

IMPLEMENTATION

Source Code

Most of the code, including the *importScript* and *export* scripts, is written in Python using Django web framework. The base level CSS and javascript are built off of Twitter's Bootstrap front end framework (found at the following website <http://twitter.github.io/bootstrap/>). The framework provides templates for buttons, navbars, and icons along with some javascript functionality. The data is stored to a MySQL server on campus and hosted on Z at the following URL <http://zweb.cs.utexas.edu/users/cs373/benbb/>.

Due to the fact that groups pooled data in Phase 1, the production database already holds most of the data including our own.

Folder Structure

The model of the data structure can be seen below.

```
wcdb/  
  crises/  
  scripts/  
  static/  
    css/  
    html/  
    img/  
    js/  
    templates/  
    xml/
```

Figure 1. The Folder Layout

The root folder, named *wcdb*, must be renamed to *wcdb* when downloading from Github. Within the root folder, there are three subdirectories:

crises, *scripts*, and *static*.

The crisis folder contains the crisis application and the standard Django files:

- *models.py* - a detail of the table structure
- *tests.py* - the unit tests and single acceptance as a catch for any bugs in the program.

The scripts folder contains the scripts *importScript* and *export*. The deviation in convention to *importScript* is to avoid conflicts with the Python keyword *import*. In phase 3, both scripts *export* will be renamed to *importXML* and *exportXML* in order to have a similar convention and more appropriately describe their function.

Inside the static folder, there are the following folders: *css*, *html*, *img*, *js*, *templates*, and *xml*. The *css* folder contains the CSS files associated with Twitter's Bootstrap.

The *js* folder holds javascript files from Twitter's Bootstrap. Twitter's Bootstrap has provided the following functionality to the website:

- animating the navbar, along with drop down menu, for each item when clicked
- capturing text for searches
- highlighting the row when a link is hovered over
- providing an image carousel for each crisis

The templates folder holds just a single HTML file for now that is not in use. Later, this folder will be used to hold templated HTML data that will be used recurringly throughout the site. The *xml* folder currently holds several XML files that were used during development. The only important file is *WorldCrisis.xsd.xml*. This file is the schema our XML data tests against in our *importScript* and *export*.

The *html* folder contains all of our content pages. The file *base.html* is similar to a template. It contains most of the HTML data that can be

seen on the site, with several blocks filled in with content from other pages. The other pages, excluding *export.html* and *export.xml* all extend *base.html*. When Django is rendering a page, it will start by rendering data from the page it extends. Since all of our pages extend *base.html*, all of our pages use the html found there. When Django is filling in content from *base.html* and encounters a block, it will fill in this block with data from the page it was first passed.

Data Model

The core data model starts with the following tables: *Crisis*, *Organization*, *Person*. All three tables are connected via a many-to-many relationships with a *Common* table to hold data that all three models share. In the common table, Crisis, Organization, and Person can have 0 or 1 Common objects. There is an abstract model, *AbstractListType*. The database will never write an AbstractListType, but abstract types are useful when scripting. The types *CommonListType* and *CrisisListType* both inherit from AbstractListType. CrisisListType will hold data for Crisis objects, and CommonListType will hold data for Common.

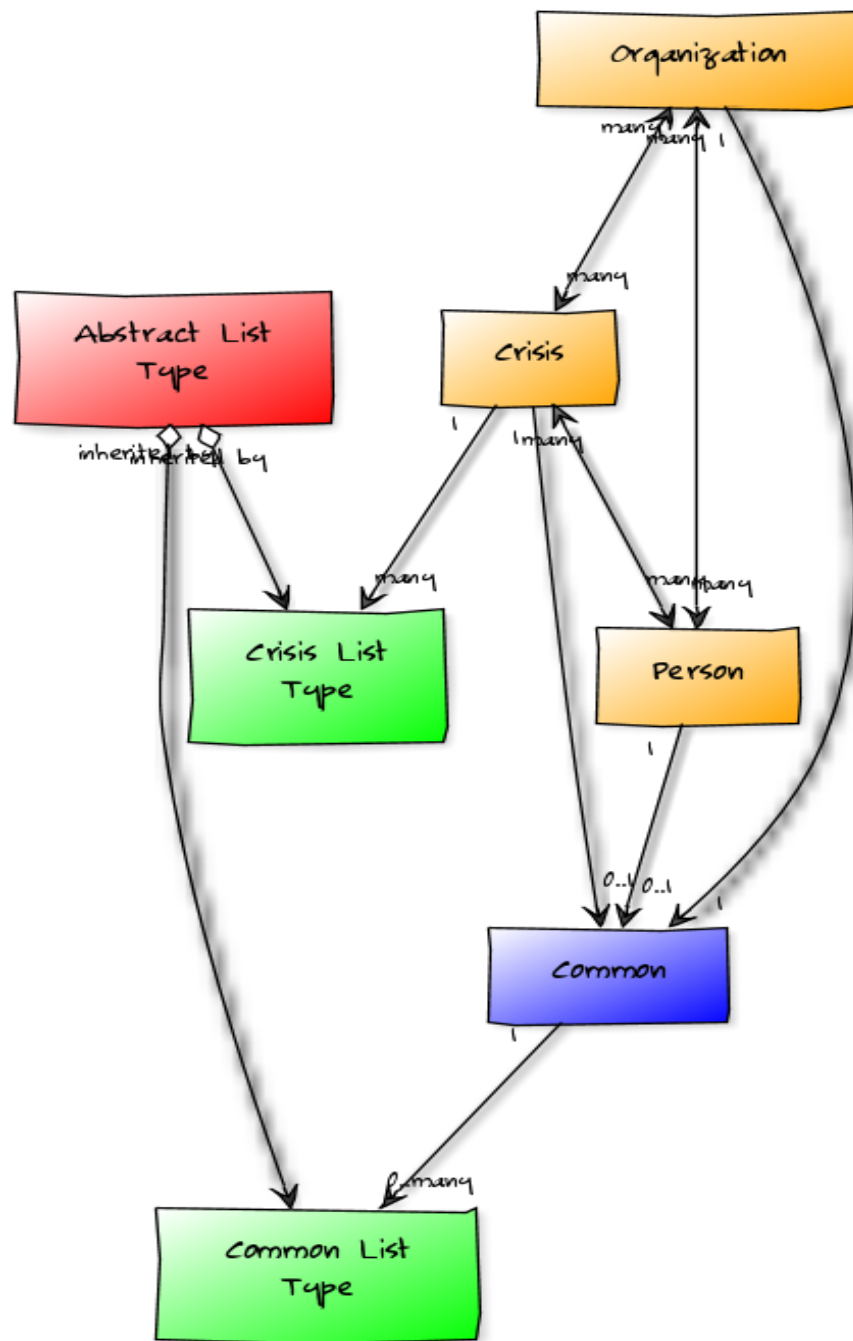


Figure 2. A diagram of our data model

XML Importation

The XML importation functionality is implemented as a Python script named *importScript.py* and is located in the *scripts* folder. The file name was chosen specifically to avoid conflicts with Python’s import keyword. It reads in an XML file as input, parses the information, and stores the applicable data into the database using Django’s model system.

The XML importation functionality is on a page on the website. This page is only accessible to site administrators and those with sufficient privileges. A link to the page itself is found under the Administrative drop down menu, and is labeled “Import”. From the page, an administrator may select a file and upload the information. This file must conform to the schema as described in *WorldCrisis.xsd.xml*; importation will fail otherwise. Any XML containing *Document Type Definitions* (DTD) is rejected as a security risk.

For testing purposes the password to gain access to the XML import functionality is currently **downing**. Users retain permissions to access the page through a cookie. This cookie will expire five minutes after logging in or immediately after closing the browser.

Detailed Look

validateXML

This function is what parseXML uses to validate the XML. It uses minixsv to validate the XML given in the parameter and returns false if an error is thrown. The XML Schema used **must** be in the same folder as the script, and must be named “WorldCrisis.xsd.xml”. This function will not work otherwise.

parseXML

Parses the file given in the parameter and returns an *xml.etree.ElementTree* Object.

xmlToModels

Converts an *xml.etree.ElementTree* to Django database models, as well as stores them in the database. The database must be properly configured for this function to properly store and create the models.

link_up_dict - The keys for this dictionary are always strings whose contents are the IDs of either crises or organizations. The corresponding value for any given key is a list of IDs from crises, persons or organizations that reference the key.

reference_dict - The keys for this dictionary are the IDs of crises, persons or organizations stored as strings. The value for the key is the model the ID references.

addReferences

Fills *link_up_dict* as outlined in **xmlToModels**.

linkUpModels

Associates the models stored in *reference_dict* with the models in *link_up_dict* and writes them to the database.

XML Exportation

As with the importation functionality, the XML exportation functionality is implemented as a Python script in the *scripts* folder and is named *export.py*. The script pulls the data stored in the database and then writes it to a new XML file using *xml.dom.minidom*. The user is then prompted to download the file.

This file should always validate against the schema outlined in the Design section. Currently, the user must manually verify that the file generated validates. This will be fixed in subsequent versions of the website.

Export does not require administrator access. Any user can export this database by clicking the Export link from the navigation bar.

Detailed Look

exportXML

This function is what is run from *views.py* to generate the XML. It calls the various *parse* functions found in *export.py* to pull data from the database. It returns an *xml.etree.ElementTree* node with the tag “WorldCrises”.

get_all_elements

This function is essentially the inverse of *linkUpModels* from above. It pulls the relevant data from the database, using the parameters specified. It then creates XML nodes containing the IDs of all objects related to it. Finally, it adds those XML nodes to the parent node passed in as *parent*.

DESIGN

XML Schema

Although our class has been divided into separate groups for this project, our group agreed to share an XML schema between groups. The latest version of the schema can be found at this url.

<https://github.com/aaronj1335/cs373-wcdb1-schema/blob/master/WorldCrises.xsd.xml>

The entire XML file is wrapped in a *WorldCrises* object. Within the World-Crisis object, there are one or many *Crisis*, *Organization*, and *Person* elements. Each of those elements have type key types, named *CrisisKey*, *OrgKey*, and *PersonKey*. There are wrappers for containers of these objects that can contain a list, i.e., a Crises container for multiple Crisis objects, etc. Furthermore, there are type definitions for each type: *CrisisType*, *PersonType*, *OrgType*.

CommonType defines data related to Crisis, Person, and Organization, such as links, images, and videos. There is a *ListType* definition that contains a list of XML tokens to facilitate objects with lists of data.

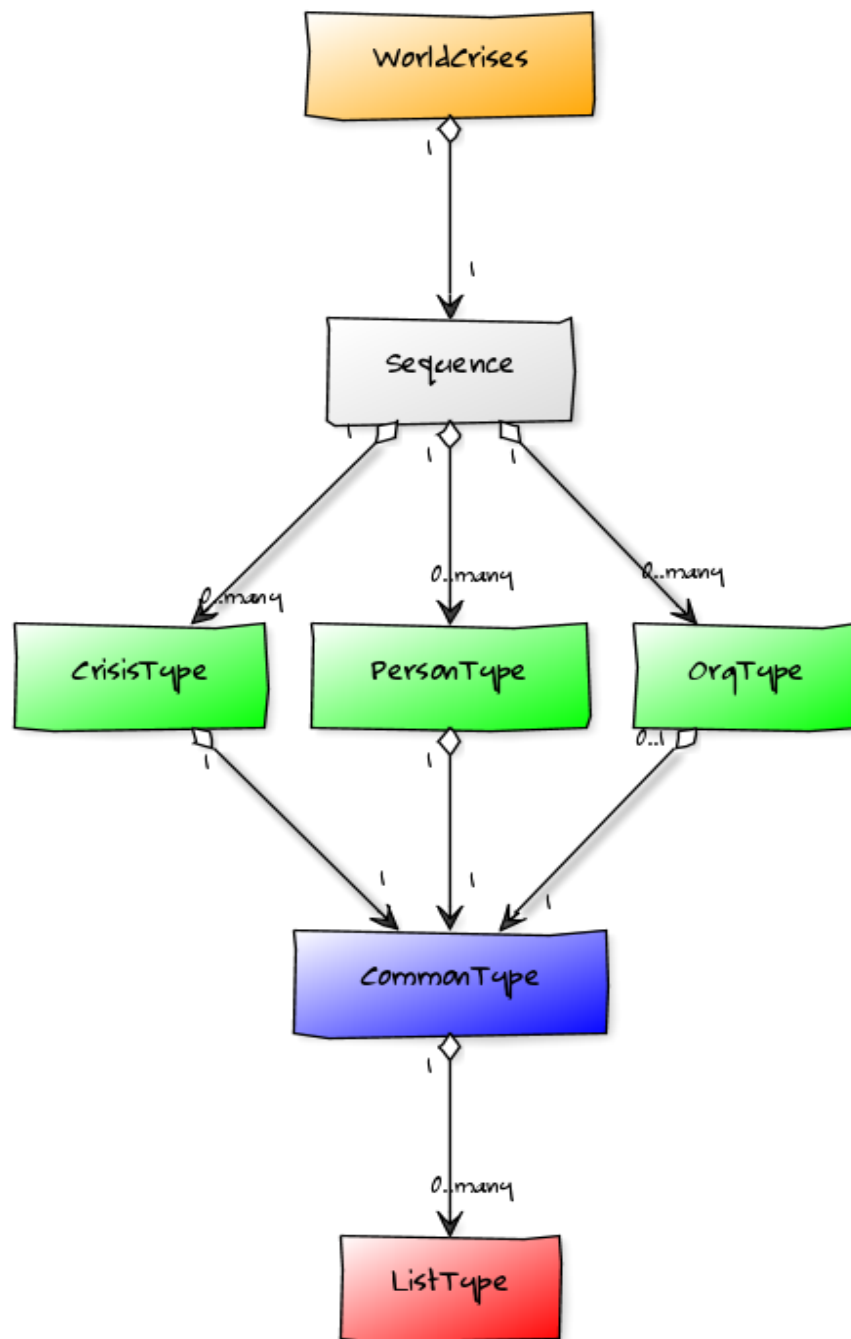


Figure 3. A simplified diagram of our XML Schema

TESTING

Testing is done using tools provided by Django and unittest. To run the tests, run the command **python manage.py test**, which will run the unit tests using MySQL. Initial test used SQLite. Since MySQL is used in production, all tests have been shifted to this database platform in order to ensure the code is running properly on our production environment. The ideal solution is to create two testing environments: one that runs unit tests on SQLite3 for performance purposes, the second runs tests on MySQL before pushing. This allows us to run quick unit tests while working and ensure the site works on our production environment.

Django extends the standard unittesting framework, providing several hundred tests against the back end. In addition, a number of unit tests are added for the scripts: importScript and export.

PAGE DESIGN

The page resembles IMDB in base design. The navigation bar displays across the top, a dark background, and a light, centered foreground. Most of the information to be presented shows in the foreground. The data presents itself in rows and columns laid out in the Bootstrap grid. As of now, all of our pages extend the base.html page, which details the format of our most interesting pages, the crisis, organization, and person pages. There are plans to create a second base, for the simpler pages, that will eliminate the excessive lines and link areas.

The Navigation Bar

The base.html file defines the navigation bar across the top of every page. The navigation bar is light colored to contrast the dark background of the site. The button to the very left is the home page button, and links to the index page.

The Crisis dropdown menu presents a list of links for the user. The first option is a link to view a list of all crises pages in the database. The rest of the of links are to the 10 most recent crises objects.

The Organization and Person dropdown menus also present a link to pages that display links to all corresponding entries in the database. Below those links are the first ten entries, currently sorted in alphabetical order.

The navigation bar has a link to a page labelled *Create User*, which links to the create user page. Entering information will add the user to the database without admisistrative priveledges.

The Feeling Sympathetic button will link to a random crisis page for anyone feeling sympathetic, but isn't sure where they would like to start helping.

The login area has two text fields, for username and password. Currently, the buttons have no effect. Future plans are to allow the user to log in without going to another page.

The search bar sits to the right end of the navigation bar. A user may type queries into the search bar, and upon pressing Enter, javascript will capture the text and a search url will be requested. Django will then search through the database for any Crisis, Organization, or Person whose name contains any of the search queries as a substring. The results will then be listed according to their type as links.

The Base Page Content Box Layout

All pages currently extend the layout in the base.html file, which defines the layout of the content. The content box renders in the center of the page in a lighter color than the page background. The box has rounded corners and a “glowing” shadow effect. This gives the content a bit of flair, and draws attention to the main focus of the page.

The layout of the page will later be adapted into two base templates, in order to accomodate for two types of pages. As of now, there is only one base, so the second type of pages will look unfinished, and should be updated before release. The first layout pages consist of the Crisis, Organization, and Person pages. They have plenty of content and a good organization will make them much more readable, giving the site a more professional look. The second layout pages are the index, Import, List, and Search pages. They are considered incomplete as of now, and will be fixed in phase 3.

Row and Columns

They layout is defined recursively on the Bootstrap grid. Rows may be defined within columns, and columns may be defined within rows. The page itself is considered a column, and the first definition is of a row. The rows and columns may be moved depending on the size of the screen viewing the page, allowing users of smaller computers, tablets, and phones to have layouts as friendly as a desktop user.

Both layouts in this site use a single outside row, with a single column named content.

First Layout

The main content, including summary, economic and human impact, ways to help, histories, and kinds are layed out on the left. In crisis pages, the text fields kind, date, time, and location will appear to the right. Images display in a carousel to the right. Links to videos and maps appear below the images. External links and links to related pages show in the bottom of the content box.

Second Layout

Much simpler than the first layout, the second layout is planned to just have the content box. The links may be put in columns to span left and right for convenience as well.

FEATURES

User Accounts

To gain access to the WCD website without admin access, there is a create user page where potential users sign up to the WCD website. Those that sign up to webpage via this portal will be given access to view all crises, organizations, and people entered in database. Admin privileges are defaulted to false, which gives all site functionality except for allowing non admins to import XML in to the database. For those who may need admin privileges, accounts can be manually updated outside of the create users page.

Random Button

Added as the Feeling Sympathetic Button, this button randomly shows an existing crisis, not organization or people.

PHASE 2 GOALS

The primary future goal for phase 2 was to get the site presenting data from the database dynamically. This has been completed.

Also at the end of Phase 1, the site has been modeled after IMBD website with emphasis on adding the specific functionality found in the site. The following list below details the the specifications and their status into the WCD website functionality:

- Each page should have a gallery for its related images (completed)
- A search page, where users can search for any key (completed)
- Organize pages by filters, like date and location (phase 3 issue)
- A short list of recent events on the home page based on how “kind” they are (a measure for total impact) (phase 3 issue)
- Improve the display of data, including charts and graphs (phase 3 issue)

Some other desired features:

- A Kindness Slider, where users can vote and rate organizations and people
- A proper logo for our team and site
- A random crisis action, to show people random crises
- A sympathy button for crises
- User profiles

PHASE 3 GOALS

Under Phase 3, we plan to add the following goals:

- Construct SQL queries to submit to Piazza
- Improve randomize button to pull from non-crises
- Coordinate with other teams to get XML data into a single db
- Generate cd