Technical Report –
Design and Implementation of the World Crisis Database

- Group name: team
- Group members:
    - Stephen Chiang
    - Aaron Stacy
    - Blake Johnson
    - Jason Brown
    - David Coon
    - Qunvar Arora
- Group URL: http://cs373-wcdb.herokuapp.com

Introduction

Problem

People 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 a 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.

Use Cases

Readers can gain information on a particular world crisis event, such as human impact, economic impact, resources used to address the issue, and ways to help those affected by the crisis. They can learn about influential people involved in the event, organizations involved with the event, and related media resources. Media resources include online articles, blogs, images, videos, social networks, and maps.

Readers can do the following: select an event and see the related people and organizations, select a person and see the related events and organizations for which that person has an influence upon, select an organization and see related events and people.

For those people interested in contributing information to the website, an administrator page is available for them to share information.

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).

How To Use The Site

Terminal Command (in root project folder)	Resulting Action
make run	Runs the server
make test	Runs the unit test
make clear-cache	Clears the cache

Design

Directory Structure

<tbd: Blake>

Git Repo Structure

cs373-wcdb/ : Git Repo

- assets/ :Various Project and Website Artifacts
  - log/ : Time Log
  - report/ : Technical Report
  - tests/ : Public Project Unit Tests (see make file to run)
  - ui/ : Website Artifact for Updating Icon
  - xml/ : Public Project XML Schema and Instance
- crises/ : Django Project Folder
- crises_app/ : Django App Folder
  - converters/ : Converters to Help Move Info To and From DB
    - prune_xml.py :Gets Models from Class DB
    - to_db.py :Converts Xml to Model Form for DB
    - to_xml.py :Converts Models from DB to Xml
    - url_to_embed.py :Follows URL and Outputs Embed
  - fixtures/ : Various Fixtures for Early Development
  - management/ : New Manage.py Commands
  - static/ : Static Files (css/img/js)
  - templates/ : Django Templates
  - Various Django modules : Modules to Support the Django Framework
- doc/ : Pydoc Documentation
- Procfile : Heroku Configuration
- README : Project Readme File
- makefile : Project Makefile (commands in tech report)
- manage.py :Django Manage Module
- requirements.txt : Heroku Required Python Modules

Differences between crisis directory and crisis_app directory

The project folder (“crisis”) contains site-wide data. It contains error handling pages (404/500 server errors) and the base site (background, header, footer, etc.)

The app folder (“crisis_app”) contains specific site data. It contains content pages, the search feature, and pretty much everything else that isn't site-wide.

XML Schema

<tbd: Aaron>

The design section needs to explain the XML structure in English

Django Models

<tbd: Dave>

Stephen says the models are equiv to the UML, SQL dump

Get Stephen's comments

Event Model

The event model stores information about events which led to world crises. The primary key uses an ID field (an integer which has no character limit, practically speaking) instead of an XML ID due to the six-character limitation invoked by XML ID ([A-Z]{6}). Django TextFields are limited to 4096 characters to prevent abuse of server resources. Events contain a name identifier corresponding to the event's name (e.g. "Hurricane Sandy"), which provides a natural way to refer to them. Event objects contain fields for the event's name, type, location of occurrence, date/time, human impact, economic impact, resources used, and aid provided.

Person Model

The person model contains information about influential persons' responses to events which led to world crises. The primary key uses an ID field like in the event model. Each person object contains a name identifier corresponding to the person's name (e.g. "Barack Obama"). Person objects contain fields for the person's name, role, primary location, and events with which they are associated.

Organization Model

The organization model contains information about organizations which were influential during events which led to world crises. The primary key uses an ID field like in the event model. Django TextFields are limited to 4096 characters to prevent abuse of server resources. Each organization object contains a name identifier corresponding to the organization's name (e.g. "World Maritime Organization"). Organization objects contain fields for the organization's name, type, location of operation, contact, history, associated events and associated people.

Embed Model

The embed model contains all information not directly hosted by the server. URL inputs are limited to 255 characters because Django does not allow CharFields or TextFields over 255 characters to be unique. Each embed object contains a name identifier corresponding to its URL. Embed objects contain the URL to various artifacts that help describe an event, person or organization (e.g. images, videos and maps).

UML

<tbd: Qunvar, Dave>

This is the big picture view of the models

UI

<tbd: Stephen>

Introduction

The World Crisis Database strives to provide users an easy and intuitive interface for browsing the content of the site. A well-designed structure either separates or interleaves the navigation from the main content as appropriate. A good user interface should also provide a safety net for bad data. Because this site will need to import information from external sources uncontrollable by the authors of the application, invalid data may arise. The ordinary user should see a page which adapts to the content, regardless of the validity of the data.

Navigation via Links

The home page greets users with a full-width splash screen that allows browsing of some of the most popular events, people, and organizations of the time. The splash screen is interactive, which means that users may choose to focus on one of the specific categories of the site (events, people, or organizations), or focus on a specific event, person, or organization. Users interested can pursue further reading of a given event, person, or organization with a link to that topic's content page.

Three separate pages accessible from the navigation bar anywhere in the site provide the database's complete selection of events, people, or organizations. This information is displayed in a tiled format, with focus on displaying a general overview of each tile by means of a short description and (if applicable) an image. Activating a tile by means of clicking on the one of interest takes users to a content page with more information regarding that topic.

Content pages are the most descriptive method of display. They provide users with a complete overview of any one topic, with information categorized for ease of access. If readers want to view more about a particular topic, they can access the topic's related events, people, and/or organizations. They also can open links to external sources given in online articles, images, videos, and social media sites that provide the latest news feeds on a given topic. To maintain consistency, navigation away from the main site will result in opening a new window or tab, whereas navigation within the site will open the page in the same window or tab.

Navigation via Search

The website also provides an intuitive search interface. On any page, users have immediate access to a search area in the navigation bar. When conducting a search, the user may choose to search through all events, people, or organizations. The user can filter the post-search results into any of the three categories. The search tool detects the active category filter and defaults to this category on subsequent user searches. For example, if People page is active and the user types in a word, the search tool will look for matches in the People category. S

Searches will redirect the user to a search result page with tabbed results, where each tab corresponds to one of the main categories of the site (e.g. event, people, and organizations). Each search result is presented as a tile with the name of the result, a briefing of the topic, and further description which is limited in length to prevent excessively large tiles. At the moment, an image is not provided since not all results will have images, and this inconsistency blemishes the aesthetics of the search results page.

Within each tab of the search, the user may also consider sorting the category in a specific way. By default, each category is sorted in the way that seems most natural (events are sorted by their latest occurrence, people and organizations are sorted by name descending). The user has the ability to change the sorting method by means of a button to the right of the tabs.

<tbd> Stephen has new feature to add

Implementation

Import/export facility

<tbd: Aaron>

include links and explanations of the import/export modules you've developed.
Stephen says: for design, discuss the piazza discussions; for imp, talk about how you coded it

Testing unittest

<tbd: Dave>

Description

The Unit tests for this project cover all the functionality of the JSON creating code, XML parsing and interpreting and validating code, database reading code, and import/export code for interfacing with the database.

The first suite of tests covers JSON code. The first set checks for proper creation of parent nodes that allows for correct reading of dictionary-type data. Other set tests adding children to a parent node and similar calling conventions for getting the data back out of those nodes. The final set tests the make_json function and its compatibility with the json.loads command from the json library.

The second suite of tests covers XML interpreting and validating code. The first set verifies that the correct data is being used and that the .strip() class function (which removes trailing and leading whitespace) results in the correct tag. The second set verifies that the XML remains valid after multiple parse iterations.

The third suite of tests handles database reading code. These tests verify that the website correctly pulls and interprets the XML from the database.

The last suite of tests is for the import/export code. The first test checks for a proper redirect response code to send the user during login attempts. The second test verifies correct response codes, including different response codes in the same declaration.

The project test suite uses a sqlite3 backend instead of MySQL for performance reasons. Sqlite3 runs the tests 5 to 10 times faster than MySQL because it does not save the results to disk. The project codebase follows good design principles of dependency injection, which makes it possible to swap the database engine out and still achieve good test coverage.

How to run the unittests

To run the unit tests - execute the following command in the main directory:

make test

The makefile invokes the manage.py file that is located in the root project directory.

<Issues To Address>

- documenting the results of those decisions
- A technical report is meant to explain your design to someone who is unfamiliar with the project
- <tbd: Create diagrams with captions>