# Final Project

## Narrative of application and data model

The proposed application for this dataset is a website that will let users explore topics discussed in Congress over the last year of Obama's administration and the first year of Trump's. The main page displays passages from speeches on a given topic, ranked and filtered by date, party affiliation, and state represented by the speaker. For example, if a user selects the topic "immigration" from a filter menu on the left, they're shown passages from speeches where keywords and synonyms for that topic appear. Additionally, users can limit results by setting a date range or selecting a party affiliation, a house, or a state. The documents are ranked by relevance, where statements that are out of the ordinary for their speakers would be more relevant than statements that are typical for their speakers.

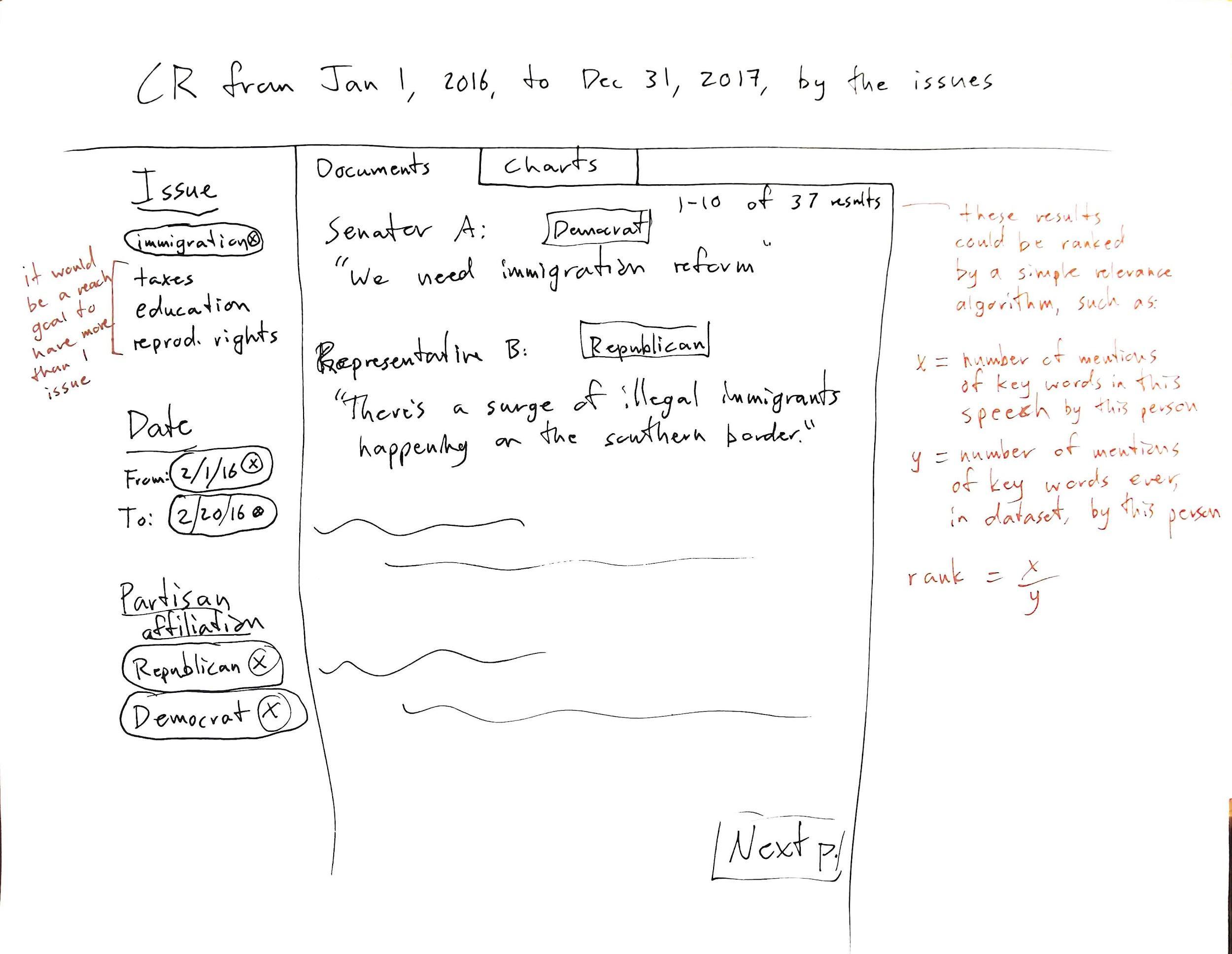


Figure 1. Wireframe of user interface.

The data model is composed of three main components, texts, people, and topics. The texts are modeled as issues of the Congressional Record, with speeches as a subclass. A speech is the text of a single speaker, however short or informal, that is recorded as part of the business of congress. Speakers are kept track of with a separate class, and topics a third top-level class. We decided to keep topic a separate class from text since multiple topics might appear in a given speech, and multiple speeches might address a single topic.

## UML Models

The “crIssue” class represents the Congressional Record as a whole. The subclasses and attributes are completely focused on the information that appears in the Congressional Record and there is only one Congressional Record per Congressional session, however, the user’s query can return multiple Congressional Record issues that have been created. The subclasses of the crIssue class only appear once, except for the “issueDoc” subclass, since this subclass represents the multiple documents that compose the Congressional Record. Also, within the issueDoc subclass is the attribute “creator” which is why the “congressPerson” class was created. A shortcoming of this class is this attribute since it is just the identifier for the creator of the document from the API. I was not sure if I should have somehow referenced this attribute to the congressPerson class.

The “topics” class represents the various topics that are discussed in Congress when they are in session. This class is used as the query by the user of our application, so the string that is entered into it is then searched throughout the different Congressional Record issues. A potential shortcoming of this is mapped as an aggregation of the “crIssue” class. My thought when mapping it was it should be an aggregation of the crIssue class since technically the Congressional Record is full of topics. Sure, it can be argued that this class could have been mapped as a composition of the crIssue class. Congress has and will still continue to meet (and thus Congressional Records would be produced) if the members did not discuss anything. But the purpose of our application is the user searching different topics that appear in the Congressional Record.

The “congressPerson” class represents the members of Congress. It is its own class because the members of Congress are independent of the Congressional Record, but they do appear in it. Our reason for creating a class for the Members of Congress is due to all of the information regarding the individuals, how they have unique identifiers in the API, how there could be multiple authors of a document within the Congressional Record, and they could appear in multiple sections of the Congressional Record. Due to these variables, creating a class made more sense. The congressPerson class is mapped as an aggregation of the “crIssue” class because there is a “creator” attribute in that class’s “issueDoc” subclass. As discussed in the paragraph about the crIssue class, the way the congressPerson class is mapped could be a shortcoming of this model’s scope.

Finally, to be honest, as (Maryse) the modeler or this component of the project, I feel I may have not correctly mapped the subclasses and their attributes correctly. My understanding of the “inheritance” relationship (solid black line with white arrow pointed from the attribute to the class/subclass) was the best to use. However, upon second guessing myself, it is possible the “dependency” relationship (dashed black line with a hollow arrow pointed from the attribute to the class/subclass) could have been implemented in some places.

## XML / JSON Serializations

Our example serializations are provided in XML, with RELAX NG files to validate them. We chose to split the serialized data into three files, one for each of the top classes, *crIssue*, *congressPerson*, and *topic*. In each file is a wrapper as the root element. For the RELAX NG validation schemas, we specify that one or more instances of the main element must exist. However, not every *crIssue* needs to have a *document* (speech) instance, since there might not be any speeches in some issues of the *Congressional Record*. We also chose to validate data types, as we have four different XSD data types in our data: *string*, *anyURI*, *float*, and *integer*. One RELAX NG file is provided for each XML file.

## External vocabularies

For the XML serializations, we chose to incorporate bits and pieces of four authoritative external vocabularies. For metadata at the issue level, we decided to keep the MODS terms provided for the *Congressional Record* by the govinfo.gov API, since it may be important for the application to link back to the Congressional Record website. For the speeches we identified within individual issues, we decided to incorporate the basic Dublin Core vocabulary for some of the elements in *crIssues*, as DC provides an interoperable and recognizable solution to common text attributes. There is a limitation here in that these two vocabularies cover the same type of domain, so they may be confusing to be used in close proximity. Finally, we used FOAF to standardize some of the information for congresspeople, since that is the best basic ontology we know of to describe people, and the URI of a Library of Congress Subject Heading as the ID for our example topic, immigration.

## Ontology

The “crIssues” class consists of 1 subclass and that subclass consists of 6 subclasses. The reason for this format is due to the multiple Congressional Record issues that may be returned from a user’s query. The subclass crIssue and its attribute issueDoc both have the data property “identifier.” crIssue’s data property is expressed as an “anyURI” since it is based on the unique identifier assigned to the Congressional Record issue from the API. issueDoc’s data property is expressed as a string which was taken from the last section of the API with an integer added to it to signify which instance it appears as. For example, the document identifier "CREC-2016-09-08-01" is the first instance in the September 8, 2016 issue where the queried topic appears. The attributes identifier, originInfo, part, titleInfo, and typeOfResource within the subclass crIssue will appear only once for a single document within Congressional Record issue and pertains to its identifying information. The text within the attribute issueDoc is slightly different since it can have multiple paragraphs depending how the text is formatted.

The “topics” class represents the user’s query. Based on our XML serializations, topics have the object property where they will appear in some documents within the Congressional Record (appearsIn some issueDoc). It has the subclasses of documents, expressions, label, and subject which inherit this class’s data property. The attribute “document” of the subclass documents has the data properties: docFreq, docId, and relevance. “docFreq” is expressed as a float because it is based on our algorithm for the platform. “docId” is expressed as a string because it is the identification string for the document in the API as a string. “relevance” is expressed as a float because it is calculated by the algorithm of our platform based on the amount of times the query string appears in the document. The attribute “expression” of the subclass expressions has the data properties: regEx and totalFreq. “regEx” is expressed as a string because it is the user’s query and “totalFreq” is expressed as an integer because both are based on the algorithm. “totalFreq” is calculated based on how many times the search topic appears in all of the returned Congressional Records. The attribute “subject” has a data property “subjectId” that is based on the Library of Congress Subject Heading URI for the topic. For example, if “emigration” or “immigiration” is searched in our platform, the subjectId assigned is “<http://id.loc.gov/authorities/subjects/sh85042782>.”

The “congressPerson” class has 1 subclass that has 5 attributes. The Person subclass has a data property of personId expressed as a string because it is the unique identifier from the API is a string. All of this information is unique to each member of Congress. However the “creator” attribute from the crIssues class is linked to congressPeople through an object property hasAuthor some congressPeople because a document within the Congressional Record can have multiple members of Congress as its author.

## RDF triples

[Do we need the text below? Didn’t delete in case it might be useful.]

* The *crIssue* class is for date and publication information for the Congressional Record issue. Its attributes currently consist of: *crId*, *date*, *volume*, *number*, and *pages*. The pages attribute currently has the children *pageStart* and *pageEnd*. Multiple *crIssue* instances are wrapped with a *crIssues* element.
  + The *document* class appears as a child of *crIssue*. It is for information regarding the speech results returned from the user’s search query. Its attributes are: *docId*, *title*, *subtitle* (<zeroOrMore>), *author* (<oneOrMore>, related to the *congressPerson* class by *personId*), *crSection* (which section in *the Congressional Record* issue this document is located), *relevance*, *type* (which will always be “Speech”), and *text*. The *text* attribute is still being formatted as we consider if we want the data model to reflect the paragraphs in the original document.
* The *congressPerson* class is for information regarding the author(s) of the document class. Its attributes are: *personId, firstName*, *lastName*, *chamber*, *state*, and *partyAffiliation*. Multiple *congressPerson* instances will be wrapped with a *congressPeople* element.
* The *subject* class is to represent the topics addressed throughout *document* instances. Each *topic* will have a *subjectId* and a set of *expressions* and *documents*. Each *expression* element is defined by a regular expression string (*regEx*) and carries a *totalFreq*, the number of times that regular expression matches throughout the dataset. Each *document* element represents one document containing this topic. It links to a *docId* and has *docFreq*, the sum of all topic expression instances matching in the document, and *relevance*, the *docFreq* minus the sum of all *totalFreq* values for the topic expressions.