EnForm

Energy Information Portal

“We envisioned a platform that collects, synthesizes and visualizes structured and unstructured data sources to support integrated research and analysis.”

Shaun Giudici
“The Judge”

Front-end development, UX design, user research, information visualization

Hassan Jannah
“The Rainman”

Back-end systems, data & system architecture, NLP, information visualization

Anthony Suen
“The Philosopher”

Energy and environmental expertise, UI/UX, information visualization, data analysis

Advisor: Professor Marti Hearst

Live: http://enform.ddns.net Source: https://github.com/jannah/EnForm

EnForm is a UC Berkeley School of Information Master’s final project (2015)

#UserCenteredDesign #SystemDesign #Usability

#NLP #InfoViz

#InformationOrganization #UXDesign #SearchInterfaces

Introduction

We present a tool that pre-indexes text from the web and then employs a number of powerful javascript libraries to slice, dice and serve dimensions derived from them alongside reference tables of raw data collected from open APIs. Multiple evolutions of this platform have served as the more tangible of a larger set of ideas we had for how a research experience could be improved when accompanied by these contextual methods for interacting with a corpus.

Many search engines today serve as vehicles that send users off to various places in the vast world wide web; and user experiences vary greatly upon arrival. EnForm, or [En]ergy In[Form]ation Portal for short, is a search engine that withholds that fundamental ability (find and go), but also offers a different approach that we will explore in this paper. By maintaining a collection of scraped web resources and pre-processing them with NLP techniques, such as NER (Named Entity Recognition), we can extract a number of new dimensions from the corpus of the results and use them to offer new perspectives on search results.

We envisioned a platform that collects, synthesizes and visualizes structured and unstructured data sources to support integrated research and analysis. Ultimately we aim to provide an enhanced contextual toolset to improve both the efficiency and thoroughness of answering a question.

Any question? Well yes, ideally. But in the span of one semester we realize that scope must be limited.

In the energy industry, the range of data sources and disjointed access to them is especially acute. EnForm is our attempt at addressing these problems by providing a web platform that allows for seamless search and analysis of both structured (we use the EIA's open API as a POC) and unstructured data (we pull publications from the Oil & Gas Journal as well as online news articles from reputable sources like New York Times and Reuters). Our goal is to empower environmental activists and investigative journalists with a new type of search engine that brings disparate data sources into an interactive visual interface.

By giving users the tools to flexibly navigate, sort and filter their information, we hope to help people uncover relevant data more quickly and simplify the task of collecting and sharing findings as a cohesive story.

Part I: User Research

User Interviews

In the initial phases of the project we conducted over 15 interviews with a range of experts in the energy field including academic researchers, business leaders, journalists, environmentalists and students with interests or ties to the energy industry. We captured and later transcribed audio recordings of each interview as a supplement to the notes we took during the session. We quickly saw the varying perspectives of each profession in terms of their needs and pain points when looking for data and conducting analysis. Beyond that we saw that people wanted to build a narratives around existing information, and present new facts. We wanted to know whether users wanted to get deeper into the story, how they find the questions to ask, and how that information impacts their decision making.

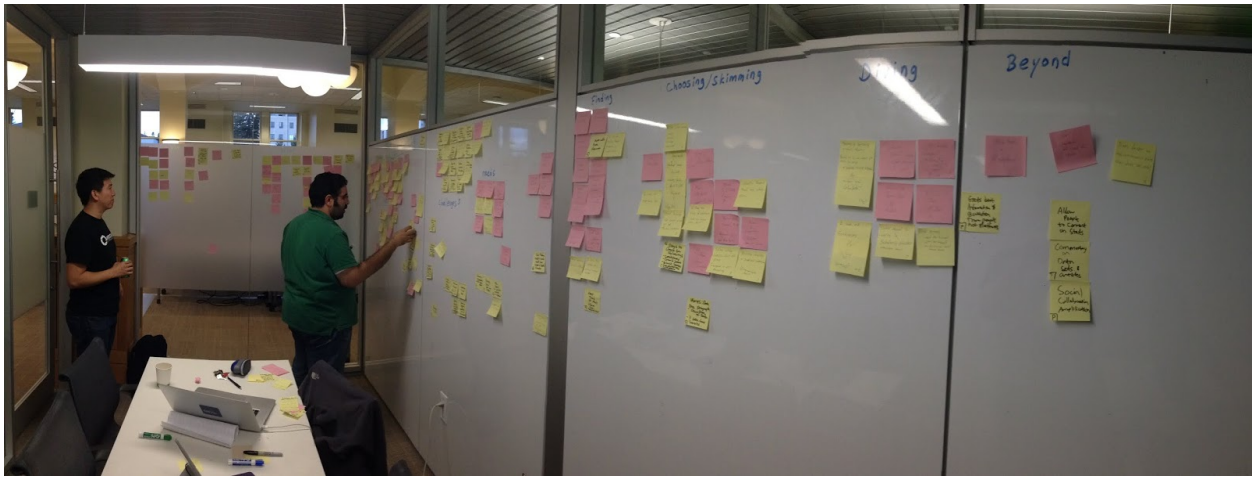
Affinity Diagrams

To aid with the analysis of the interviews, we conducted multiple affinity mapping sessions. In the first session we transferred all pertinent items from our interview notes and transcriptions onto post-it notes. We then proceeded to arrange them loosely into categories, often creating new categories as the need for them arose; through this process we began to see themes emerging. In our second session we digitized our post-it note groups using the handy post-it capture app from 3M. We transferred our insights into a digital canvas where we then took a finer tooth comb to the data. We arranged, combined and prioritized the insights into six main categories.

We found that many of our interviewees still rely on numerous online platforms that do not integrate the NY Times, FT, WSJ, and other energy journals with data sources like Bloomberg, EIA, and World Bank. It was also apparent that these people use a variety of documents ranging from PDF, Word, Excel, and R tables to view and manipulate reports and data sets. Specific pain points that stood out include difficulty in organizing related yet separate databases, getting quickly to the context of an issue they are unfamiliar with, especially in regards to relations between people and organizations. These insights and many more provided early hints about how to think about the value that could be added by an integrated data, analysis, and semantic information platform.

One feature which seems an obvious need by many is the ability to quickly access the deeper meaning of the article, and allow them to see the context of the reported sources.

Many non-academics wanted to be able to create a narrative based on the research they were performing. Use cases range from making production plans, creating scenarios scenarios, writing an exposé or creating price structures which relied on reference material from economic and energy data sets along with written analysis from research reports and news articles.



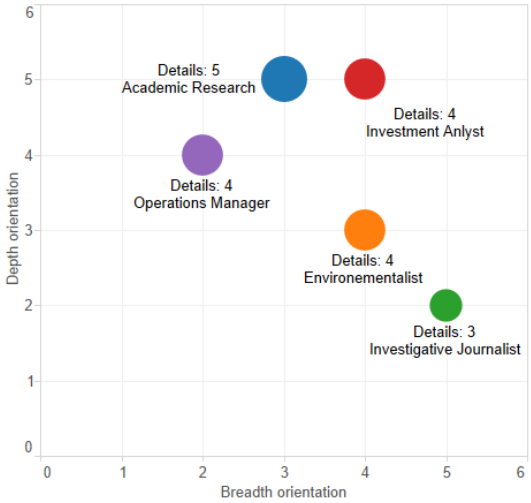
Affinity Diagram Categories

Why	What	How	Needs	Challenges
• Understand • Build • Decide • Inform	• Prices • Supply and Demand • People • Resource Data • Market Intelligence	• Data: Newspapers, Expert Reports, Publications, Academia, Datasets • Tools: Search (Pull), Acquisition (Push), Analysis, Visualizations, Notes	• Query • Data • People • Platforms • Analysis	• People • Query • Data Sources • Data Formats • Data Quality

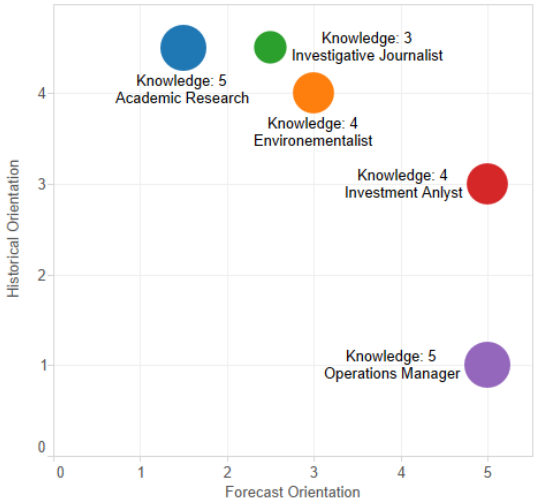
Personas

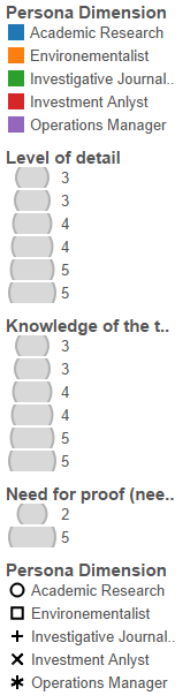
We created six personas to get a picture of what our users wanted. They came from a range of backgrounds from environmental analysts, operation managers in oil companies, research scientists, energy financial analysts, journalists, and policy makers. We mapped out their daily usage habits and needs, and then we created scores for each one of them; finally visualizing the figures. These factors range from expertise, team size, speed, resources, technical know how, bread of knowledge, passive, collaboration, research goals, data processing procedures, responsibilities, time horizon orientation, method of data collection and consumption, preferred sources of data. After we completed these personas, we quickly realized that no one tool can satisfy all six groups. However, the tool we envisioned most closely approximate meets the needs of two archetypes which are the orange and green on this chart, the environmentalist and journalist.

Persona Preferences: Breadth, Depth and Level of Detail

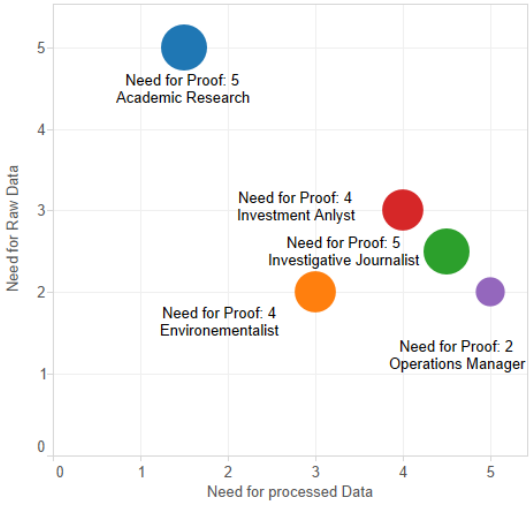


Persona Preferences: Historical orientation, Forecast orientation, and Knowldge of Topic

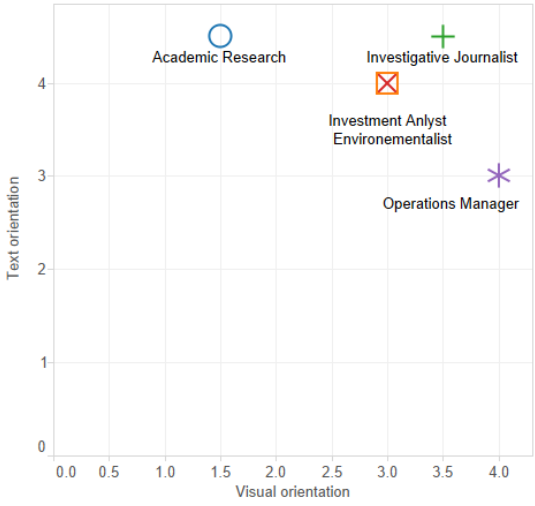




Persona Preferences: Raw v.s Processed Data and Need for references



Persona Preferences: Visual Orientation vs. Text Orientation



Estimated persona preferences based on interviews (1 Low - 5 High)

Catgory	Question	Academic Research	Environementa..	Investigative Journalist	Investment Anlyst	Operations Manager
Getting Data	Access to Specialized Data	4	4	4	4	2
	Fast Queries	3	2	5	3	4
	Need for Specialized Resources	4	3	4	4	2
	News Preference	1	3	5	2	2
	Publication/Report Preference	5	5	3	5	5
	Push orientation	4	4	4	4	5
	Search (pull) oriented	2	4	4	4	2
	Variety in Source	3	4	3	4	3
Processing Data	Breadth orientation	3	4	5	4	2
	Depth orientation	5	3	2	5	4
	Forecast Orientation	2	3	2	5	5
	Historical Orientation	4	4	4	3	1
	Level of detail	5	4	3	4	4
	Need for Human Input	3	4	5	3	3
	Need for processed Data	2	3	4	4	5
	Need for proof (need for reference)	5	4	5	4	2
	Need for Raw Data	5	2	2	3	2
	Text orientation	5	4	5	4	3
	Visual orientation	2	3	3	3	4
Resources	Knowledge of the topic	5	4	3	4	5
	Need for People	3	4	3	4	3
	Organizational resources	3	2	1	5	4
	Team Size	2	2		3	5
	Technical savviness	3	3	5	2	1
Sharing	How to you share data (own category)	2	3	3	5	
	Need for Feedback	2	4	3	3	2

Marcus - The Sophisticated Online Magazine Journalist

Marcus is a global energy and technology journalist for an online publication called Quartz. He has a constant urgency to keep up with macroeconomic, environmental, and technological trends and has written award winning investigative pieces on climate change negotiations and the future of the energy policy. He often chats with folks like Elon Musk to get a sense of what is really happening. Given his broad interests in fields and need to understand the inter connections between disciplines, he is constantly looking at websites or news article paper researchers who provide access to the dataset. Not only is he a great writer, he and Quartz are also praised for their infographics.

Michele - The Environmental Activist

Michele is an up and coming star in the environmental movement who is working on domestic fracking issues across the US. She is passionate about reducing impact on communities and educating the public on the risk of these industries. Michele is very curious about the latest wave of health related complaints and the congressional legislation dealing with the industry. With her team, she is constantly reading the latest scientific reports along with industry publications to find trends that the public usually does not see. She prefers to get raw data and analyze it themselves, using her models that they created. Since data accuracy is essential for her, the access to a variety of government agencies datasets and reports are critical. They measurement success by how many people are seeing their reports which are targeted at the general public, political leaders, and funders They all tend to feature excellent infographic that highlight research from science journals, local news reports, legal opinions, alongside easy to understand charts concerning the fracking industry.

Scenario 1: Breaking News in Energy

There is a new development in regards to the oil markets. An unexpected boom in US oil production has led several OPEC to flood the markets with oil in order to avoid losing market share. This has depreciated prices which puts a huge strain on the entire renewable energy sector across the world. But there might an innovative strategy that governments and cleantech startups are converging on the survive this geopolitical chess game. Time is not on Marcus' side though. Rival publications like the WSJ might scoop him on this idea.

While he has contacts in various fields to contact, he needs to quickly generate and present an interactive infographic for the website but the digital graphics' team says that it will take half a day to do a D3 visualization. He also needs to work with several partners to verify the information on this graphic. Marcus really wishes he has a quick visualization solution that can integrate the articles, feedback, and data without the graphics team getting in his way. Also without a complete framework of all the data, he feels like he will be spending a lot of time explaining this topic during interviews he is still collecting. Traveling out office to meet with these sources means he doesn't even have time to draw up something in PowerPoint. Marcus feels like there are just too many reports and data sets to reconcile and not enough time or expertise to pull this report off.

Scenario 2: Pushing Environmental Policy

Michele is concerned over the impact on local communities that number of fracking projects present. She constantly has news alerts on local regions based on the Pennsylvania region. Because it is important for her to catch any new potential issues coming from the oil fields along with the latest developments in state and federal policy, she has 20 tabs open at once and spends a good chunk of the day downloading data sets and reports in order to write her weekly updates. But keeping track of all the email newsletters and news articles reminders feels hugely distracting and inefficient.

Given that congressional legislation in the state is days away from a decisive vote, she has a deep sense of urgency to present the best possible case that fracking must be restricted in its expansion plans. Several congressman and allies in the NGO community are looking for information release regarding the potential threats if the legislation to expand fracking is passed. They are also asking for a focus public marketing push which she has not had time to prep for. How can she create a big information campaign on such short notice? There was the added burden of having the documentation and visualization vetted by all the members of coalition which would take a long time to merge if it was over PowerPoint. Difficult task of combining several different points of data to several very different stakeholders.

Features Design

Based on the scenario we developed, we came up with some of these core features for the Enform platform:

Skim View

Quickly analyze reports and graphs without leaving the page.

Time Filter

Control view for changes over time, whether it's the past or future projections.

Semantic Information

Ability to see connections between countries, organizations, and individuals

Graphs

Ability to see shifts in real data and article counts over time.

Dive View

Deeper understanding of data from the article with complete text to review.

Stories Feature

Researchers and journalists indicated they wanted to save stories and quotes.

Annotation

Commentary was emphasized by the need to share knowledge among team.

Data Research

We talked with many energy experts and they all agreed the EIA is the best source of energy data. We also considered databases from the World Bank, but quickly realized that the EIA's 1.3 million data sets was more than sufficient since it also covered numerous economic and

demographic figures for the US. For written reports, we will relying on industry publication from oilprice.com along with major, Google News items, and New York Times, and EIA's energy reports.

We also eliminate paid subscriptions to processed data sets and publications due to their proprietary nature and high reproduction cost. We also wanted to limit scope of the project to the US since global statistics and individual project datasets would have taken too much time and resources to acquire.

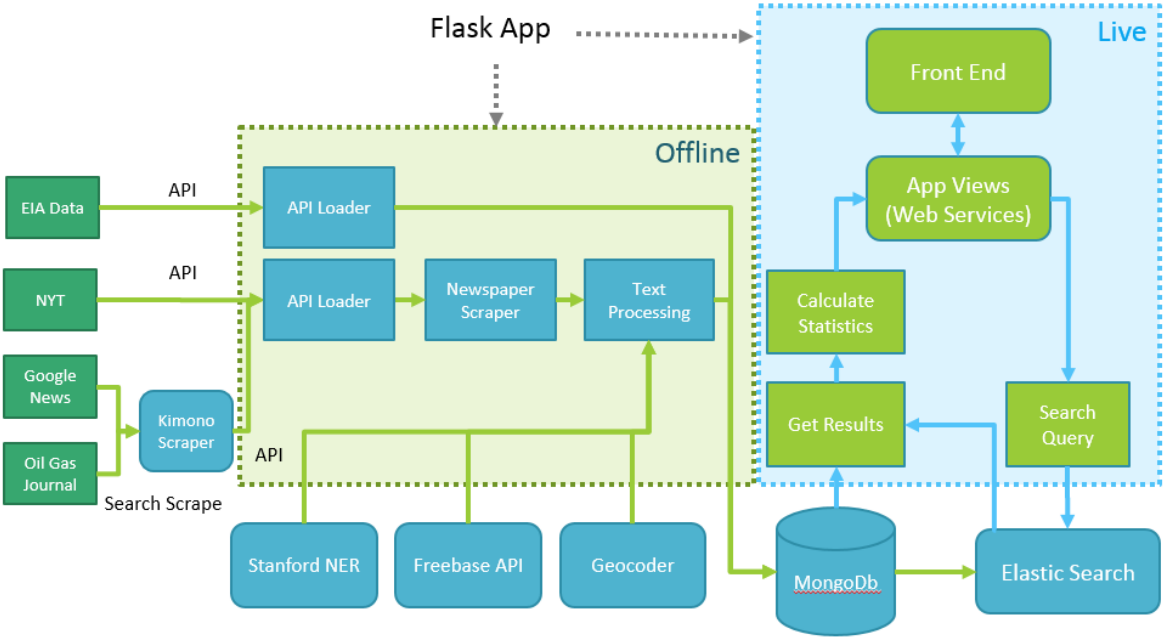
Part II: Backend Design and Implementation

We envisioned building a platform that is data-source agnostic; we could change the information feed and the system would still work. To do this, we needed to design and implement Enform using system design techniques. While this increases the scale and complexity of the tool, it provided us with a flexible and extensible platform that we could build incrementally in modules. It enabled us to add new features and debug issues more easily.

System Architecture

The below diagram shows the major components that make up the architecture of the portal. The Architecture shows three parts. The edges are the external tools and services and repositories we used to load, process, and store information. The green box shows the modules that are run "offline" when articles are loaded. It is usually run in large batches. In the green box are the modules that are used in real-time when users are actively using the platform.

System Architecture and Data Flow



Application Backend using the Python's Flask Framework

Due to the extensive use of NLP and document processing, we needed to use a python backend to leverage the available libraries required for such operations. Furthermore, we wanted a backend that can support parallel processing to improve performance. We also wanted to have a backend framework that can support frontend functionality. Thus, we chose the Flask web-framework for python. It is well documented and supported. It has a good templating engine (Jinja) for HTML page parts which allows HTML content generation on the server side to minimize the the load on the browser. The templates would also have access to python functionality for faster, more integrated manipulation of HTML content which might not be possible on the client-side (JavaScript).

Generally, the backend is divided into four components:

1. **The The API Loader:** a multi-purpose, configurable, and parallelizable API loading interface. The interface can be configured to map how the API requests should be formatted and how the results can be interpreted. This loader was strictly used to load all articles into the database.
2. **The Article Processing Pipeline:** A complete pipeline that takes in Article URLs and runs a series of operations starting with web scraping and ending with a full processed article object.
3. **Search Interface:** A collection of modules that are used to search the articles and dataset collections and format the output for consumption by frontend components.

4. **Frontend Views and Templates:** A collection of API-like web interfaces that return search database objects as JSON or partially rendered HTML pages.

Database Storage: MongoDB

Based on our initial research, most APIs and scraping tools use JSON to store and exchange data. In addition, we didn't have a clear internal data structure for our internal data. Thus, we saw a need for a data repository that can allow flexible data structure, preferably as JSON objects. Because we expected to utilize different APIs with varying data structures, the flexibility is critical to reduce the overhead of organizing data. Most SQL-based repositories require strict data structures which could create major overheads in managing the backend database. We also needed an engine that has a well-supported python interface and a good ORM (Object Relational Model) library to manage DB operations and manipulate documents. MongoDB provided us with all of the above[1]. Because none of us had prior knowledge of running and using MongoDB, it was important to experiment with MongoDB very early in the project before committing to using it.

Search Engine: Elasticsearch

After reading about and experimenting with MongoDB, we found that it does not provide good full-text search support on multiple fields as it can only index one text field per document (e.g. either title or text). This would slow down the search response time for users. We needed an additional layer on top of the database to facilitate search. We went with ElasticSearch (ES). ES can index multiple text fields in the same document. It also provides very fast and and highly configurable search. We could use some NLP techniques to analyze and search terms (e.g. remove stop words and morph plural words) and also tune the result scoring mechanism to boost certain results (e.g. increase score if search terms occur in the title). Finally, ES has several well-documented python interfaces[2]. The only drawback is that it didn't support MongoDB out of the box. We needed to use a special plugin to connect ES to MongoDB to allow ES to "suck in" the content of certain MongoDB collections[3].

Frontend: JavaScript, HTML5

The front end is built in pure JavaScript using a variety of libraries for DOM manipulation, data processing, information visualization, and styling.

Main Libraries:

- jQuery: DOM manipulation and interactions
- DC.js + Crossfilter: Integrated, interactive visualizations built on top of D3.js

---

[1] http://namlook.github.io/mongokit/

[2] http://elasticsearch-dsl.readthedocs.org/en/latest/search_dsl.html?highlight=nested

[3] https://github.com/richardwilly98/elasticsearch-river-mongodb

- lodash: utilities and client side templates
- Jinja: client side templates
- Boostrap: UI elements and layout
- Sass: Syntactically awesome style sheets

External Components

To be able to implement our vision for the application given its scale and complexity, we needed to outsource as much functionality as possible. That was a strategic decision we made very early in the project.

Newspaper.py: A generic news scraping library

The Newspaper.py[4] library is one the most critical components of the backend. It's a general purpose web scraping API that is optimized for scraping news content. It provides several functionalities that are core the application out of the box which saved as a lot of time. Without it, we would have had to write customized scrapers for each page or invest more time to develop a multi-purpose scraper.

It uses an internal voting algorithm to decide which parts of the HTML document make up the primary content of the page. That content is then extracted and stored as clear text. The library also extracts image links. In addition, the library extracts metadata from the HTML page (e.g. tags, keywords, authors' names, publication dates...etc). Furthermore, the library uses NLP methods to generate an automated summary of the text.

Because of our high dependence on the library, we decided to use the internal structure of its article object as the bases for our internal data structure for articles.

Backup Selenium Scraper

Some websites use javascript to render content on the client side. Such websites cannot be scraped using traditional methods, including the Newspaper library. The Selenium scraping library for python scrapes websites using an actual browser client to load pages fully before scraping. The downside is that scraping is slower, more CPU and memory intensive, and more prone to getting stuck causing the code to pause indefinitely. Thus, this method is only used as a backup when the newspaper API fails to extract content. Furthermore, the scraping script must be monitored regularly to check that the script is running properly.

[4] https://github.com/codelucas/newspaper

Named Entity Recognition (NER)[5]: Stanford NER Library

One of the key functionalities we wanted to implement is to extract named entities from article text. These entities, we believed, could help add more context to the sub-corpus of search results and also to individual articles. Furthemore, wanted to have several classes of entities (e.g. people, countries...etc.) instead of a single list of all entities.

We had three options for implementing NER:

1. *Build our own NER algorithm using NLTK*: We had a skeleton available from previous work. However, it needed some significant improvements. It also had too much manual configuration to be scalable to a large corpus. Finally, it was not able to properly organize different entities into different classes.
2. *Use the NER module of NLTK*: The NLTK library comes with an NER module. The module is ready to use and has good performance. However, the quality of results was not always good.
3. Or, *use the Stanford Core NLP NER component*. The Stanford library is one of the best NLP libraries readily available. It also has a good NER component that can classify results into different classes. The biggest drawbacks were that the library was slow and it is written in Java.

After some research, we found that the Stanford library could be run as an HTTP server on the local machine which can be queries by any platform. We also found a python wrapper that can easily query this HTTP server. Furthermore, we could extract more entities and classify them into 7 classes right out of the box[6] (People, Locations, Organizations, Dates, Time, Percent, Money)[7]. We did some performance and quality testing. Comparing the results between Stanford NER and NLTK NER showed the quality of results in the Stanford library was better (see Appendix 1). Furthermore, the performance was excellent as it took a few seconds, and sometimes under 1 second, to process a full article. Running the library as a server allowed us to run multiple articles at a time which significantly reduced the processing time of the full corpus.

---

[5] Paper: Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370. http://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf

Software: http://nlp.stanford.edu/software/CRF-NER.shtml

[6] Using the english.all.7class.distsim.crf.ser.gz module

[7] NLTK NER also classifies named entities but using a slightly different taxonomy.

Data Source Libraries

From previous work, we realized that doing NLP on text in real time can be impractical for large documents. We needed to pre-process all the text and store the results in the database. Thus, we needed to pre-fetch the full text of news articles offline using news or scraping APIs.

Source Specific APIs (e.g. New York Times)

Some news sources provide search APIs that return article links and, in some cases, full text. For example, the New York Times provides full access to its content through API calls. We used their APIs as an example of an openly available data source that can be used in enform. While some full text was available through the API, we decided to just use the search API to get the search results and additional meta data. The URLs were then passed to the standard scraping and processing pipeline.

Generic Search Scraper: Kimono Scraping API

While some of the sources we decided to include provide a full API implementation that includes the article links or full text, others don't. Furthermore, several users mentioned Google News as an important source. Unfortunately, the Google News API was deprecated. To be able to include Google News and other search based news sources, we needed to scrape those pages.

In line with our plan to outsource as many functionalities as possible, we decided to use the Kimono Scraping API. Kimono provides a gui scraping interface of any website and then avails the results as JSON which can be retrieved through an API. Kimono also runs its scraper using multiple machines which reduces the probability of anti-blocking mechanism preventing web scraping. Using Kimono allowed us to use a generic scraping tools that can be used on many sources with minimum changes to our backend.

We used Kimono to scrape search results, not full articles. We pass the URLs of scraped results to the newspaper library to scrape the actual text. We used Kimono to scrape Google News, EIA Reports, Oil & Gas Journal, and OilPrice.com.

EIA Datasets API

The EIA energy datasets are widely used by our target audience. EIA provides a very good and standardized APIs to load part or all of its datasets (>1.2 million). All datasets (called series) are uniformly structured and come with very useful meta data (.e.g measurement units, start and end dates, time frequency...etc.). We decided to use the EIA series data structure as a the bases for our internal data structure for datasets.

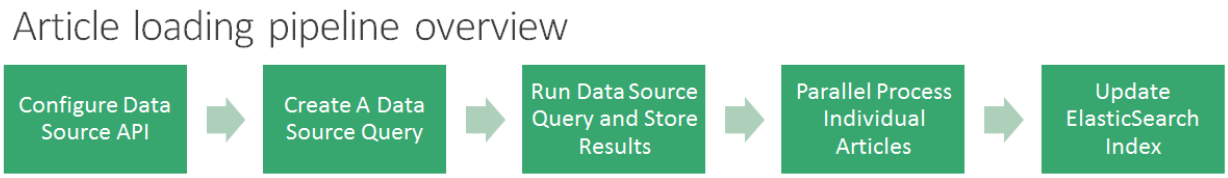
Named Entity Decoder: Freebase API[8]

The Freebase API (now migrated to the WikiData project) provides a large corpus of named entities. We used the their API to: 1) create more uniformity for extracted entities, and 2) get more metadata about those entities.

Geolocation Decoders

To provide better geographic context for the corpus, we used external geocoding APIs (e.g. Bing, Google...etc) to get more information about extracted locations. Primarily, we wanted to get country, state, and city information for each location where applicable. The information will be used to 1) disambiguate locations across the corpus, 2) improve location based search results, and 3) provide a location contextual filter in the user interface.

Backend Implementation



The API Loader

The API Loader is configured for each data source. There are three main configurations:

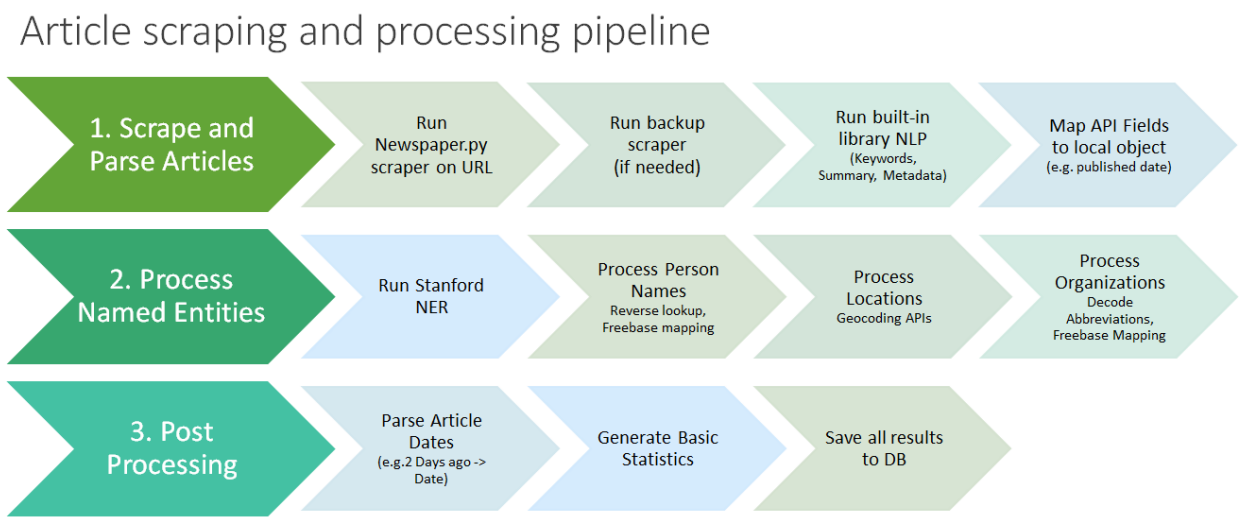
1. Request URL structure: The API root URL and URL format using placeholders for URL parameters (e.g. {root_url}{version}/{api_id}?{parameters}). There are special parameter names reserved for the search terms and date ranges.
2. URL Request parameter values: Values for each parameter in the URL such as API key, search term, date ranges, etc. These also include special query parameters for result paging.
3. Result field mapping: These mapping help the loader know how to handle returned results. There two types of mappings: meta data (e.g. total hits), and individual result mappings (e.g. title, url, date).

After a datasource is created, it can be used to create multiple search queries. For example, we created three queries for the NY Times (Oil Natural Gas, and Shale). The queries are then executed to retrieve results from the API and store them in the database

[8] http://www.freebase.com/

Article Processing Pipeline

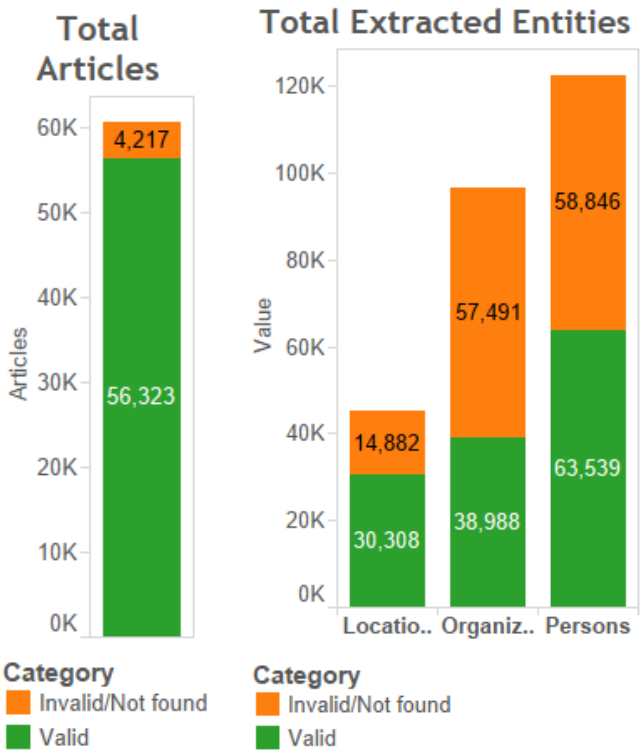
The flowchart below shows the key steps involved in acquiring and processing an article from a web resource.



Scrape and Parse Articles

Run the newspaper library extractor on the article's URL to retrieve the HTML and detect the main content. If no content is detected (e.g. due to javascript page rendering), the backup selenium is used to render the page fully, extract the HTML, and store in the newspaper article object.

Metadata from the page is extracted from the text using the internal library functions. Information extracted includes: keywords, authors, source name. The library also runs an NLP algorithm to generate a summary of the text.

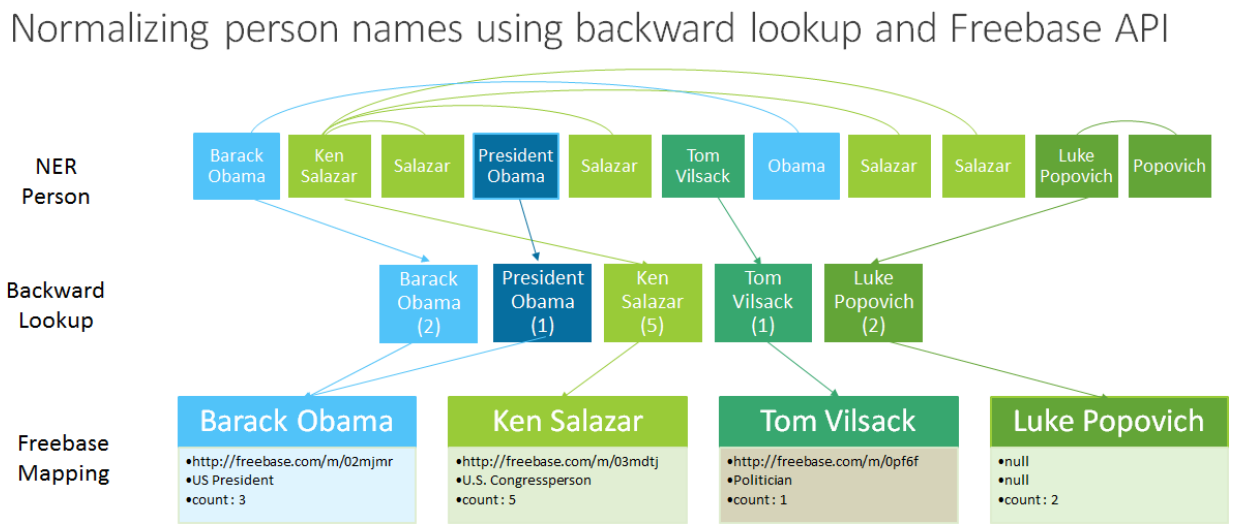


Process Named Entities

NER using Stanford NLP Library

The full text from each article is passed to the Stanford NER module, which is running as a server, to extract entities and classify into 7 classes (People, Locations, Organizations, Dates, Time, Percent, Money). The returned JSON result is stored in its raw format inside the article object in the database.

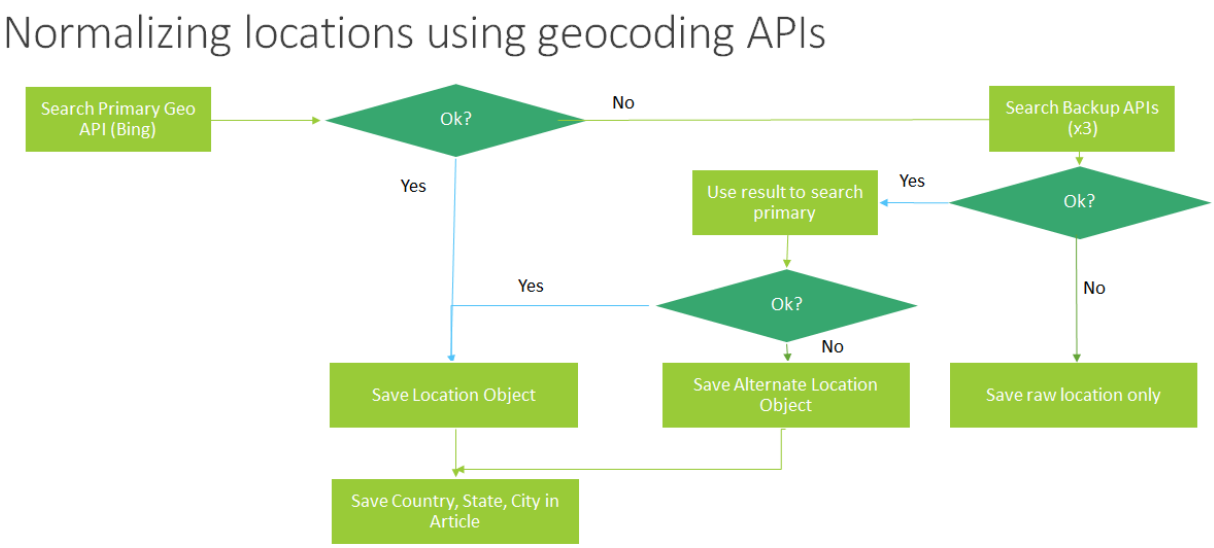
People Processing



One of the classic challenges in NLP is dealing with people's names in text. Within an given article, an individual could be referred to in multiple ways. For example, an article that mentions President Barack Obama could use his full title, full name, or just last name within the same article. To resolve this ambiguity within an article we used backwards name lookup. This techniques relits on the assumption that writers use the full name of a person when they first mention them and then use partial names to refer to them. Luckily, the Stanford library returns results in their original order in the text. Thus, for each name in the array, we could look up whether that partial name is part of an earlier mention (.eg. Obama is a substring of Barack Obama). We conducted several tests on random article and we were able to resolve a big portion of the ambiguity within a given article.

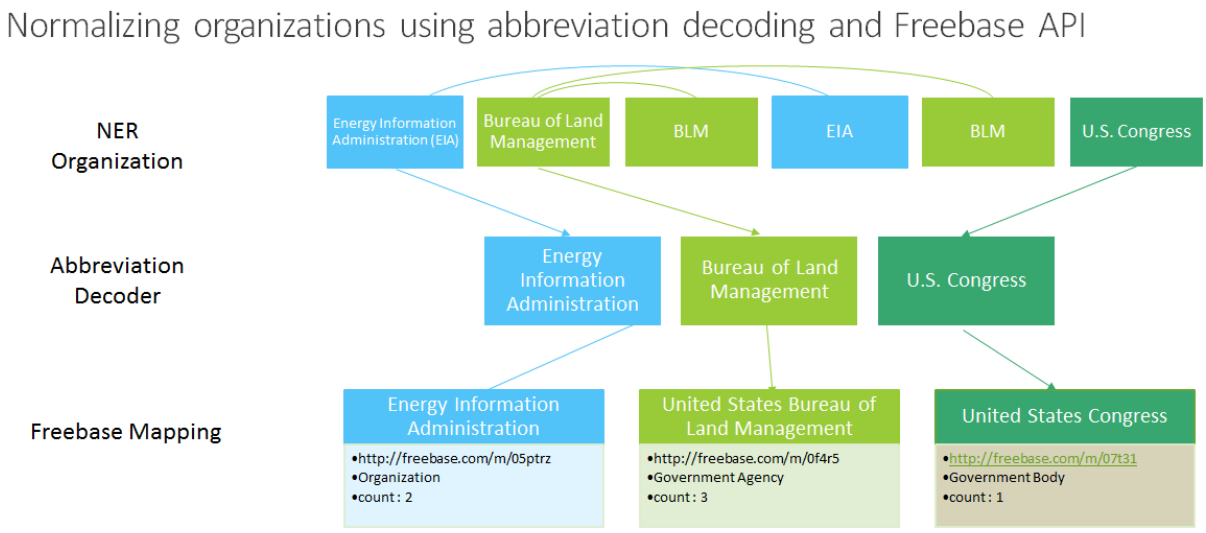
However, For Enform to be able to generate useful contextual information from people's names and filter them properly, this ambiguity must be cleared for a given entity across the entire corpus, not just an individual article. Given the size of our corpus, and the variety of text, doing this manually was prohibitive. Instead, we used the Freebase Entity API to search for any names and retrieve a more semantic representation of that name. We also retrieve additional metadata about the person (e.g. Role) and a URL reference to their page on Freebase. This technique was very useful for to resolve the ambiguity of many notable names within the corpus. To improve the performance, we maintained a list of aliases of most entities to be able to look them up locally instead of using the API.

Location Processing



We used external geo-decoding services to create uniformity among the locations in the corpus and provide more geographic context for a search results. Each extracted location is passed to the primary geo decoding service. We used Bing as a primary service because it had no API call limits. If the a location is found, that location is stored in the database and the key location levels (country, state, city,and full address). To facilitate geographic search and better retrieval performance, these key levels are stored with the article object with a reference to the full object. If the primary lookup fails, the a series of backup services are used (Google, Yahoo, OSM) until result if found. If a result is found, it is passed back to the primary service in order to maintain the consistency of the data structure of location objects. If that fails, the retrieved object is saved. If all attempts fail, only the raw location is stored. To improve processing performance and reduce delays from API calls, we maintained a database of address aliases that can be looked from the the database before making API calls.

Organization Processing



To normalized extracted organization names, we also used the freebase API. However, organizations extracted from text are different than names. The main challenge is mapping abbreviated names back to full name. In such case, plain reverse lookup won't work. Instead, we used the Schwartz and Hearst simple abbreviation detection algorithm to match abbreviated names with the full organization names[9]. The original implementation was written in Java. We found a python implementation for the same algorithm[10]. We just repackaged the source code to fit our needs. After the names are mapped, the full name is passed to the Freebase API.

Post Processing

Date Processing

Publication dates are important meta data for each article for search and interactive filtering to work. The dates that are retrieved from the initial API calls are more reliable than the ones generated by the newspaper library. However, some dates mapped from API results cannot be parsed into python dates using the standard date utilities. For example, recent article dates are displayed on Google News results as human readable date references such as "2 days ago" or "6 hours ago". We used a python library to convert these relative dates into actual dates.[11]

---

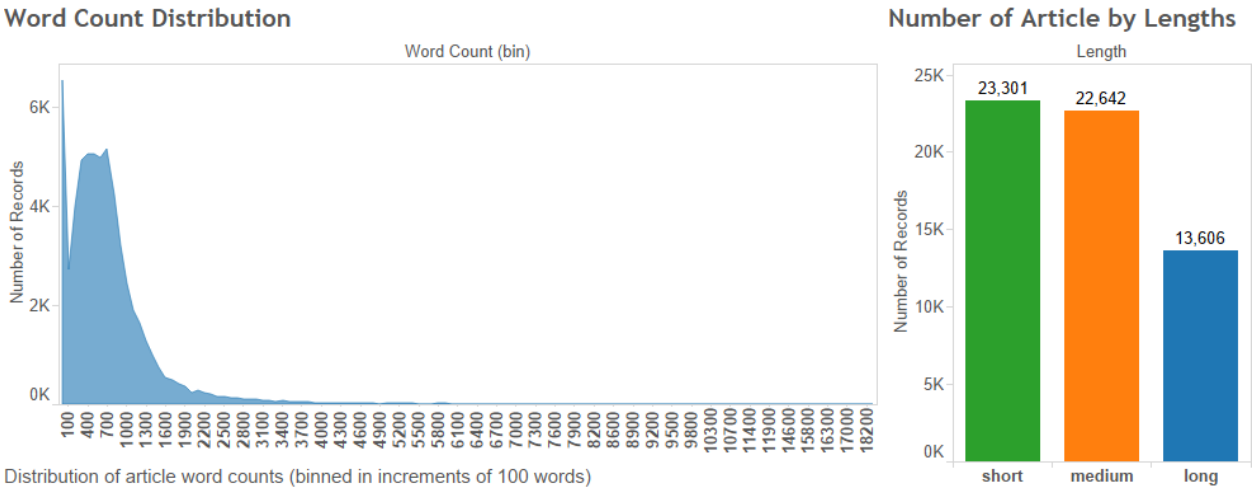
[9] A Simple Algorithm for Identifying Abbreviation Definitions in Biomedical Text, Ariel Schwartz and Marti Hearst, in the proceedings of the Pacific Symposium on Biocomputing (PSB 2003) Kauai, Jan 2003.

[10] http://www.clips.uantwerpen.be/~vincent/scripts/abbreviations.py

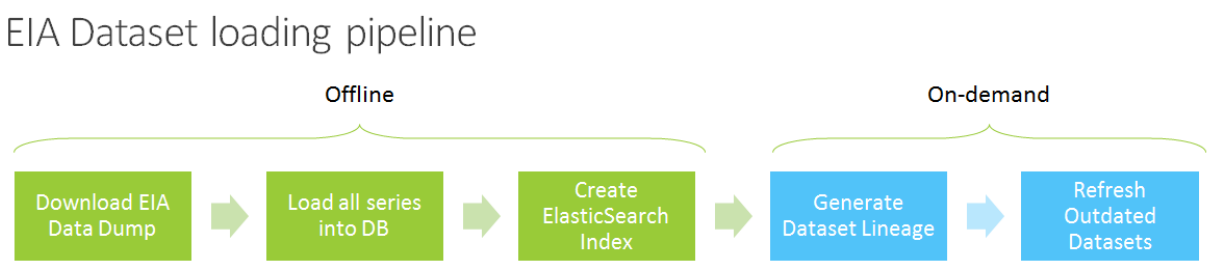
[11] https://github.com/bear/parsedatetime

Basic Statistics

For each article, we generate interesting statistics such as word count, paragraph count, average word length ..etc. The main usage of these statistics is to classify articles by length. We performed some basic analysis of these statistics. Based on the value distribution, we classified articles by length into three classes: 1) Short (<500 word), 2) Medium (<1000 words), and 3) Long (> 1000 words). We also considered estimating article difficulties based on length, unique word count, and average word length. However we didn't pursue the idea further.



Dataset Loading



The EIA provides open access to its entire dataset (+1.2 million series) through an API. It also provides daily data dumps of the data to reduce the load on the API server. The series are organized using a taxonomy of categories which could be used to navigate the collection. The category objects are stored separately from the series with reference to series IDs.

Loading the EIA series involves fiesteps:

1. The full dataset dump (categories and series was downloaded and extracted.
2. The extracted JSON files are loaded to the database as is. The first two steps were only done once.

3. After all datasets are loaded, an elasticsearch index was created to facilitate fast search of datasets using dataset metadata.
4. When a specific dataset is requested and is outdated, the dataset is refreshed with an API call in realtime and the new data is saved in the database.
5. In addition, the lineage of the database it calculated by recursively navigation the the categories collection and then storing the top two levels in the series object. This is done once per series.

After working with the dataset collection for a while, we found that it would be impractical to expose the entire collection. Size was one issue. The other big issue is that several categories include US-wide datasets in addition to state-specific series of the same data. Furthermore, most datasets are available in different time frequencies (daily , weekly, monthly, annually). After several trials, we decided to enable only US-wide datasets in addition to key primary global series (e.g. Oil and Natural Gas prices). We also refined the enabled series to include only the smallest time frequency for any given data point (e.g. keep only daily data where available). The front-end would handle time-aggregations in real time. The final subset contained around 4,615 EIA data series.

Content Search

ElasticSearch (ES) was used to facilitate very fast text search of multiple field in both the article and series collection. The platform provides many knots and bolts to perform search and finetune result scores. We decided to use minimal configuration due the time and scope constraints.

The search process involves two steps:

1. Using the ElasticSearch-MongoDB-river plugin, ES is configured to copy a document collection from MongoDB. The configuration would specify which specific document fields are copied.
2. The actual search is done using ElastichSearch-dsl python library. Each query can be tweaked for the task.

Article Search

Due the size of the article object, only relevant text fields were copied to ES: Full text, keywords, meta_keywords, authors, meta_data, title, publish_date, tags, named_entities, named_entities_raw, and source_url.

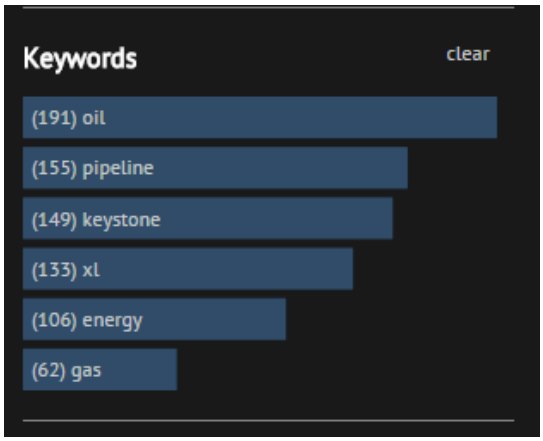
The standard ES key term analyzer was used to clean up the the search terms (e.g. remove stop words and account for plural words). After analyzing search terms, ES would search for all search terms across all fields, not just within individual fields. Finally, some fields were assigned higher scoring weights ("title^3", "meta_keywords^2",keywords^2", "tags^2"). The bases for this boosting is that if search terms were found in keyword or in the title, then the

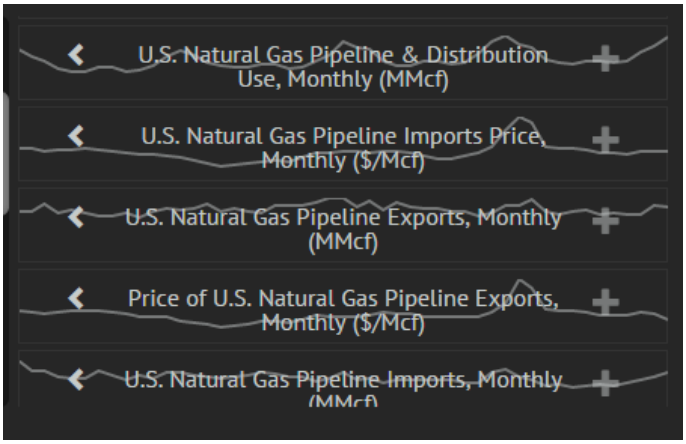
article is probably more relevant. This configuration was a results of testing and experimenting with ES and also some feedback from users.

Dataset Search with Keywords

Searching for relevant datasets was more challenging. Generally, just using search terms to search the metadata - mainly name (^2), description, and geography - of the datasets lends poor or no results. We tried to improve the query by switching to the more inclusive OR operator to search across fields, but the results didn't improve much.

Finally, in a moment of inspiration, we experimented with utilizing the top keywords found in the article search as search terms for the datasets. This technique drastically improved the quality (relevance) of the retrieved datasets. For example, searching for "Keystone XL", a proposed shale pipeline, now returns datasets related to pipelines and gas.





Part III: Front End and Iterative User-Centered Design

Through an iterative process we designed and developed a number of novel interface components to support a highly contextual research experience. We moved quickly from very rough sketches/digital mockups to a functional prototype given how advanced we were with the backend. Once we had the functional prototype, we conducted more than 10 user experience interviews from individuals in academia, journalism, industry and design.

Our user experience interviews focused on validating importance or use of elements over styling and specific positioning. We relied on the modularity of interface items in parallel while considering alternative layouts and views. This includes interactive elements for filtering, viewing, collecting and annotating search results.

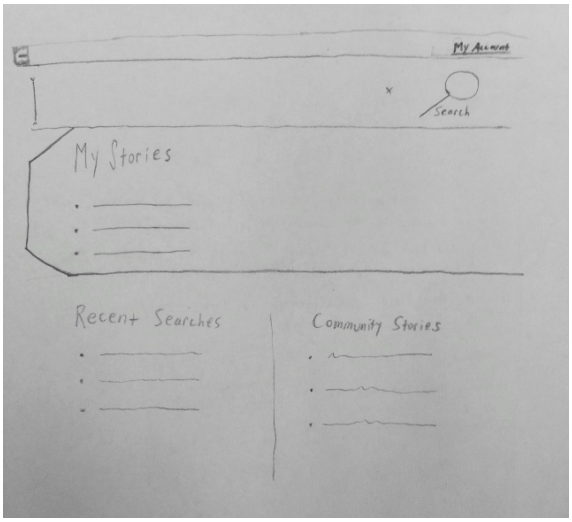
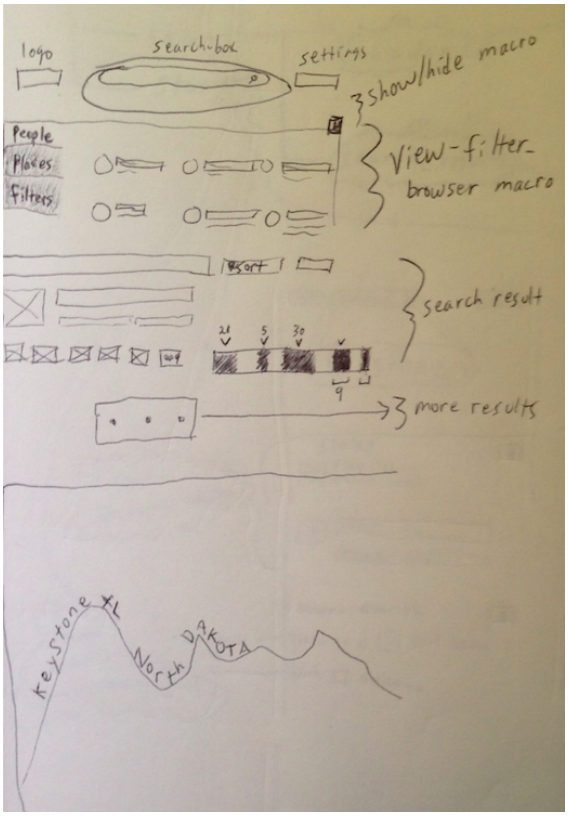
We'll start by describing the progression of the interface design throughout the project. For each phase we will include:

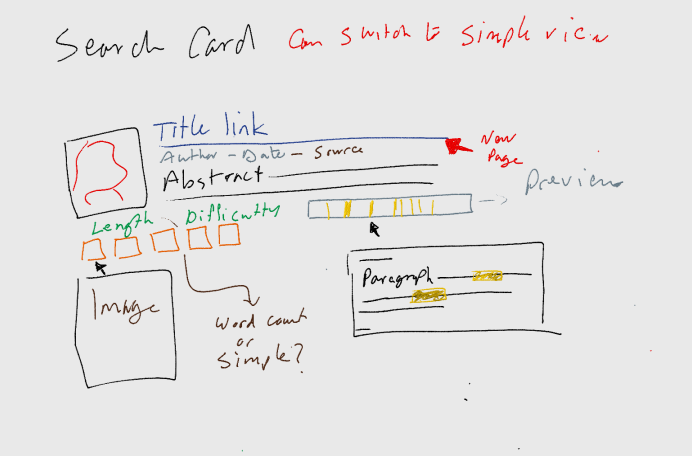
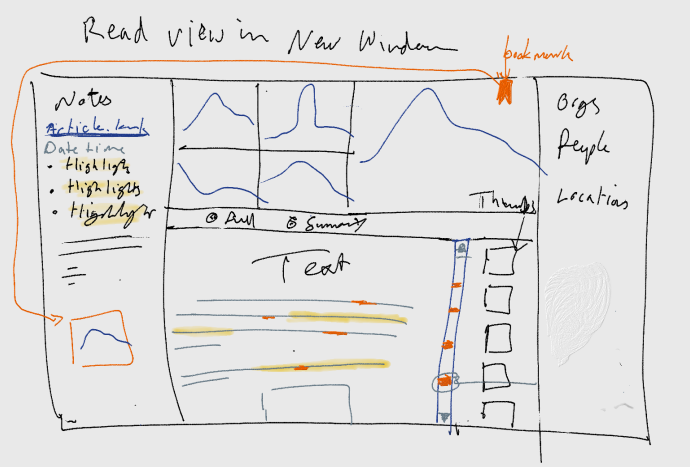
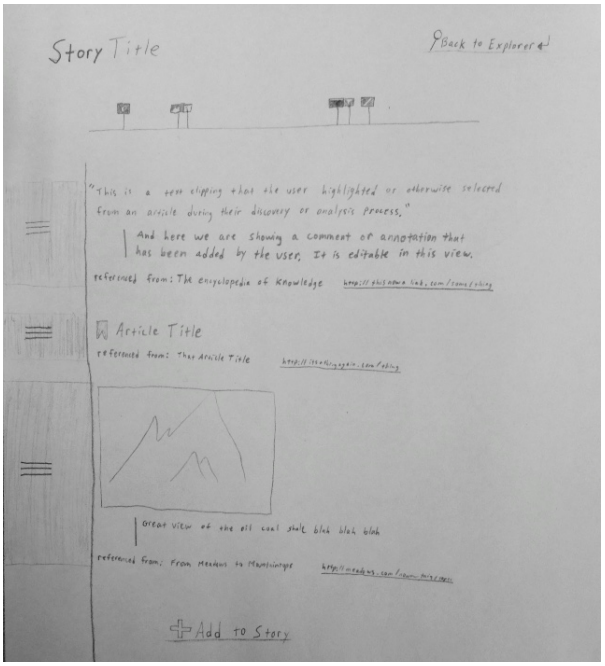
- Explanation of the work done
- User observations and feedback
- Top insights that informed our next design iteration
- Lastly we'll describe in detail the unique interface components that we think add the most value in all of this.

Front-end: Prototype 0 "It started with sketches"

Armed with a better sense of who and what set of scenarios we were designing for, we began the interface design with group brainstorming and sketching sessions. We filled the room with ideas and used whiteboards to mock-up and clarify our meanings. Our initial design was comprised of four pages:

We designed features in rough sketches. deciding what we wanted to tackle initially. This is also where we decided to focus on search results and novel features (pushing the story to later). There were several approaches we took to the projects. Our original designs were in greatly aligned with one another. We had three views initially: home page, skim view, dive view. We can talk through the process of moving towards a more integrated skim + dive view. Visions diverged and converged. User testing helped us verify and blast some of our assumptions.

1) A homepage with search bar and displays of a user's activity in the app.	
2) A search results page with filter and skimming features	

3) A close-up view of an individual search result	
4) A “dive” page, for closer inspection and viewing of a search result without leaving the app	
5) A “story view”, where a user’s bookmarked items would be appear	

Our sketches served as useful conversation aids both within the team and for discussions with our advisor. They helped ground the conversation to what we proposed to build, and paved the way for a more refined product specification.

We spent weeks in this sketch + mockup phase, growing our understanding of the project's scope and magnitude. As we refined our ideas for features that would address the user scenarios we engaged in major prioritization and scoping of the project.

Interface feature scoping

Our research showed that time was a major pain point for users in their typical research practice. Therefore the idea of developing features to skim search results more efficiently emerged as one of our highest priority objectives.

We observed a vast fragmentation of methods for collecting findings from a search. For example, some people worked directly in email, some used tools like Evernote or Pocket, while others built up their own local databases for reference later. In most cases a lot of copy/paste was involved. With these insights in mind, we envisioned a way to integrate the process more seamlessly with the act of reviewing search results. This idea represents the story-building portion of our app. Though the story feature idea is important and interesting, we were advised and subsequently decided to de-prioritize it in favor of putting a full effort towards the novel search + skim features, collection methods and technical implementation.

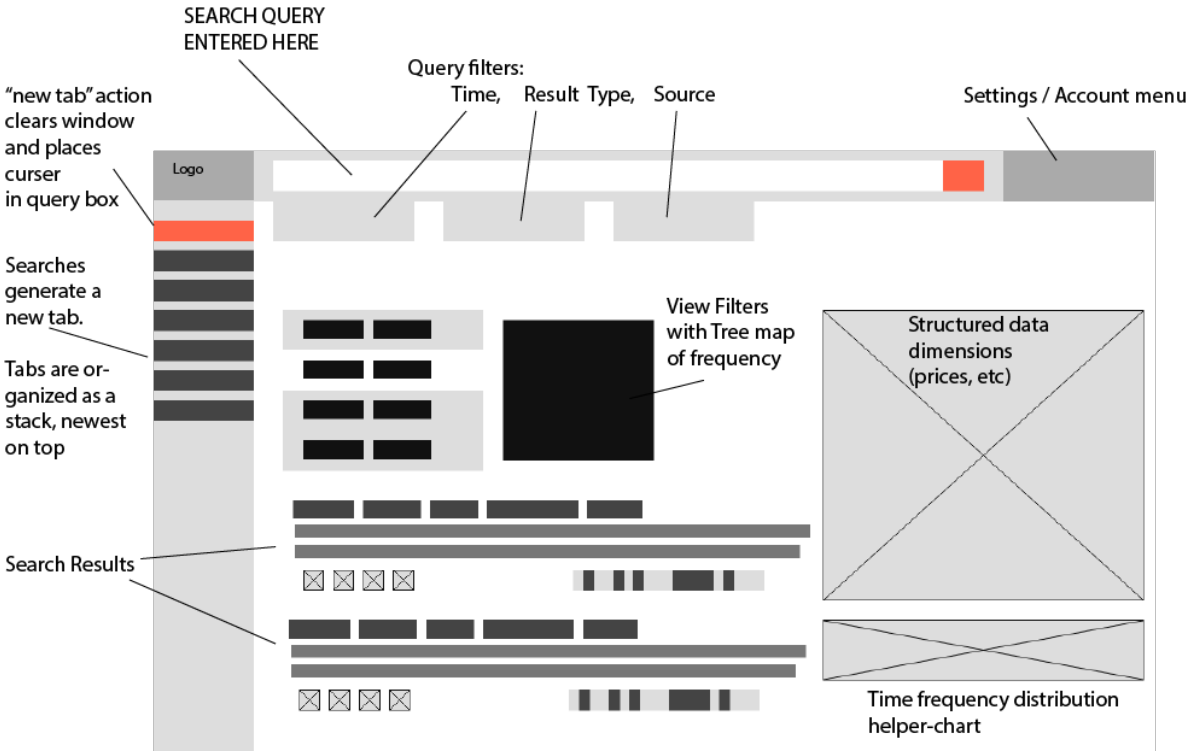
There were a number of other features that our research supported a need for, but in order to maintain focus and reduce to a manageable project scope, we had to make the tough decisions of cutting them.

Here are a few of the features that didn't make it off the drawing board:

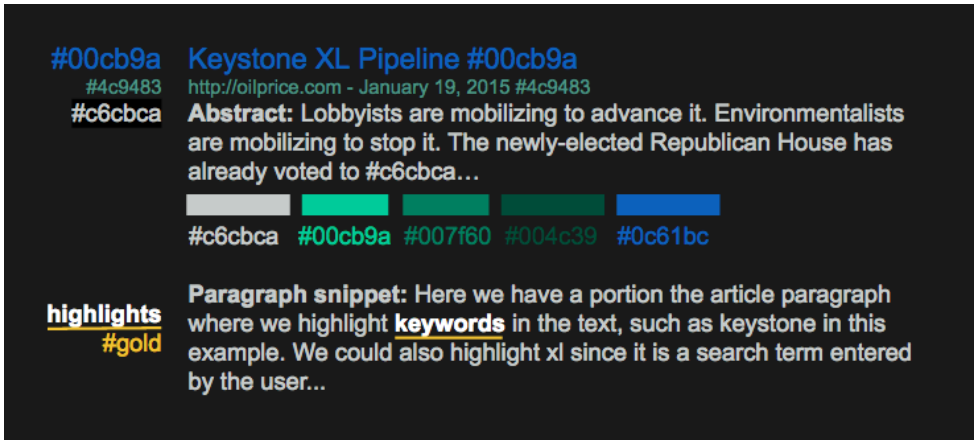
- **PDF scanning**. Potentially useful to users who have non-public content, such as purchased reports. Ultimately removed due to it being a niche feature with high difficulty.
- **Dataset uploads**. Potentially useful to academic researchers who have obtained specialized datasets, or financial analysts who have excel models. Ultimately removed because we chose personas who don't have a strong need for this.
- **Data Downloads**. Potentially useful to energy experts who wanted to be able to download the data from the platform. Ultimately removed as a feature with less priority.
- **Maps.** Potentially useful to users who wanted additional context, but ultimately not executed due to lack of time.

Wireframes and functional prototype specification

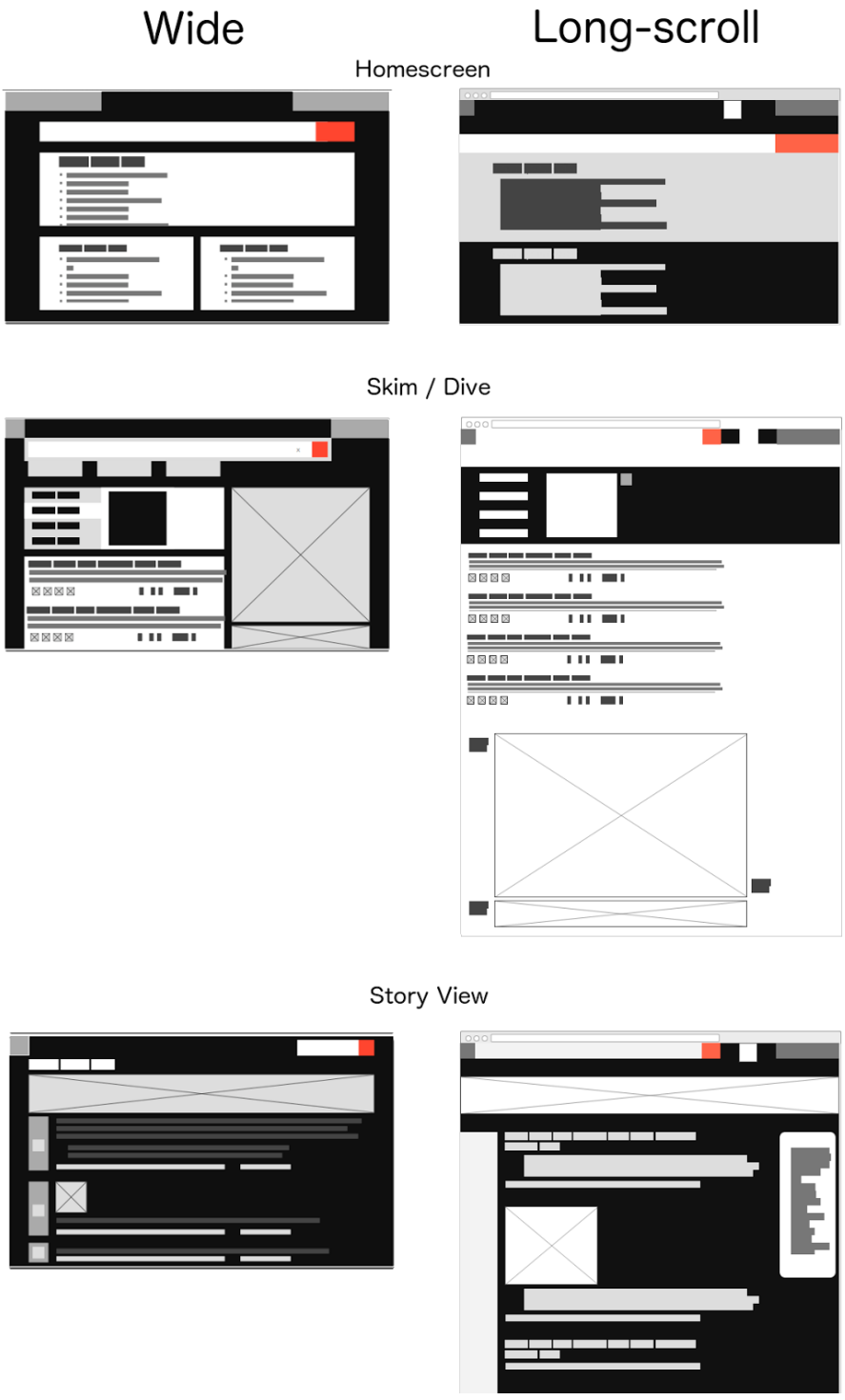
We refined our sketches into more concrete wireframes to serve as design specs for the first prototype. The annotated wireframe (see image below) helps to give more context about what each element's function will be.



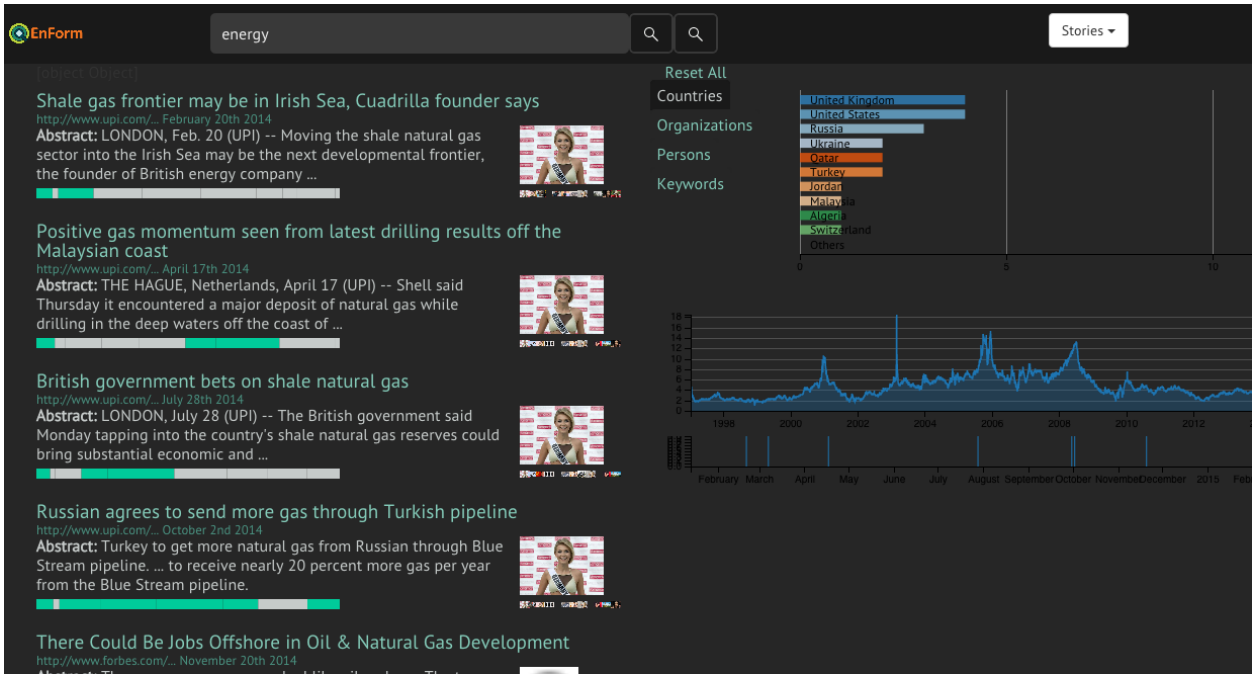
Color scheme specification for search cards



We initially created two fundamentally different layouts for the app: a long-scrolling version and a widescreen version (see image below). They are different enough from one another to spark some interesting questions, for example: how important will it be for users to have both the list of search results and the interactive dataset graphs in view at the same time?



Front-end: Functional prototype v0.1



Explanation of work done

Notice that we jumped straight from sketches and wireframes to a functional prototype. This is a somewhat controversial choice that we had planned from the start of the project. In the interest of developing an end-to-end functional system in the limited time allowed by one semester, we made a team decision early on to skip over the sometimes popular low-fidelity prototypes. Despite this, we were able to maintain a fast iteration schedule for the interface layer because of our initial investment in completing the backend systems in parallel with the initial user research phase.

Prototype v0.1 (pictured above) was about proving initial functionality between the front and backend. The elements on the page have been placed roughly and mostly reflect the default styles, many of which clash with our darker background. The important point to note is that the three visualizations and search results were generated dynamically and are driven by real data.

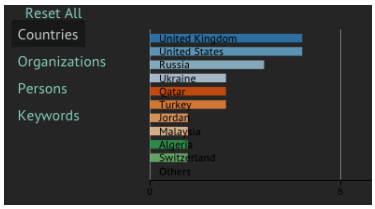
1) **Tile bars for paragraph skims.**

These bars represent a literal mapping of paragraph makeup of the article. The deeper colors indicate paragraphs where more search terms occur. In this version the bars are static, though we have plans for making them interactive, allowing you to read snippets of the article.

2) **Search results generated by working search engine**

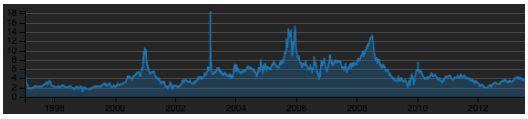
Each result links to its own separate “dive view” which contains the origin webpage within an iframe, and flanked with details about the article (such as extracted textual entities, top keywords, and scraped images).

3) **Distribution of extracted entities (dc.js bar chart)**



This element represents a tabbed browser of four separate charts (one per entity). The bars themselves act as filters that narrow the search results.

4) **Graph of a real EIA dataset (dc.js line chart)**



This element is a proof-of-concept graph of one of the many energy related datasets that we collected from the EIA.

5) **Time series of results (dc.js timeseries chart)**

This visualization depicts articles on a time series by their publish date. This chart doubles as a time filter for search results via interactive brushing and linking.

Key user observations and feedback

The pictures are a problem - Multiple users were drawn to the images initially. They were squinting at them; some thought they were avatars. Many commented on their lack of relevance to the page and that they are distracting. Design decision: remove images

The tile bars are cool!.... once they figured out what they were (sometimes with our help) most users commented how they had never seen this before. General comments praised the usefulness of this high-level view of an article’s make-up. The color coding seems backwards (lighter should mean MORE) Design decision: Add descriptive label, dedicate more real estate to this feature. Reverse the color coding.

The homepage is not descriptive - Both regular users and those familiar with the energy industry are failing to accurately describe the app after given some time to view the homepage. It lacks context. Design decision: Add short descriptions about what can be done with the app.

The dive feature is too noisy - Users report that search result links opening to dive view is unexpected; many were confused on why it did not just send them straight to the article. The iFrame in the dive is difficult to navigate. The page as a whole is described as “too busy”. Design decision: Make the dive a popup modal. Demote scraped images to reduce noise

The mentions section is confusing - A common question was “what are mentions?”. People did not seem to understand that the mentions section acted as filters for the search results. “Mentions feels advanced, why don’t you just call them filters?”.

Page hierarchy is broken - Two design critiquers commented that the two column hierarchy is fighting for her attention. “Its not clear which column is more important” Design decision: Move to a 3 column layout with facets moved to the left column (more traditional)

The search result quality isn’t great - “People are trained by the norms set by google” one user said. “Google will have CPUC and CEC results when you search California wholesale energy”. Reaction: This is a known issue. The quality of our results is directly correlated to the extensiveness of our collection methods; we surrender that the focus of this project is not about building a better collection than google.

The graphs are hard to comprehend - “There are no labels.” “I can’t read the axis labels” “What is this a chart of?” Design decision: Improve color choices of graphs and add more descriptive labeling.

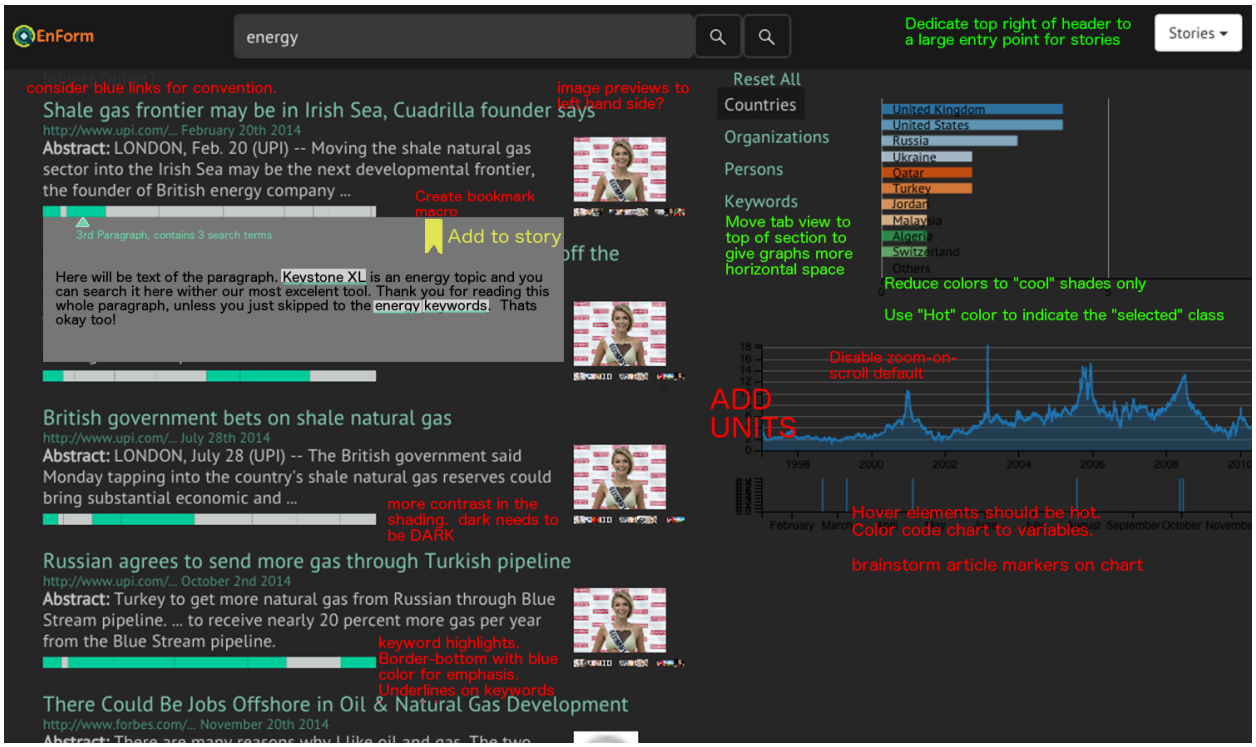
Articles per day may be too granular - “why is every bar the same height?” “I typically look for trends over months and years”. Design decision: Offer a higher level granularity than day for time series

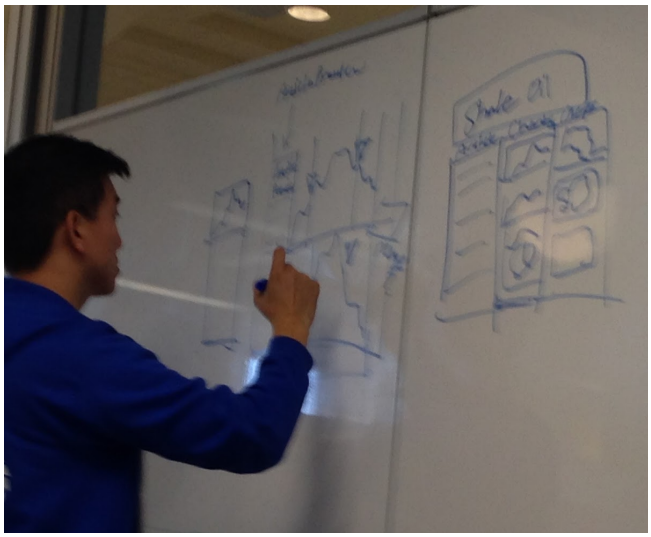
Mentions order was not optimal - Just as a general observation, users who were more familiar with energy topics tended to find the “sources” and “keywords” filters most useful. Design decision: Reorder default position of search filter groups.

Country filters have an ambiguous meaning - Countries are sometimes thought of as the sources of news publications, as in “articles published in England”.

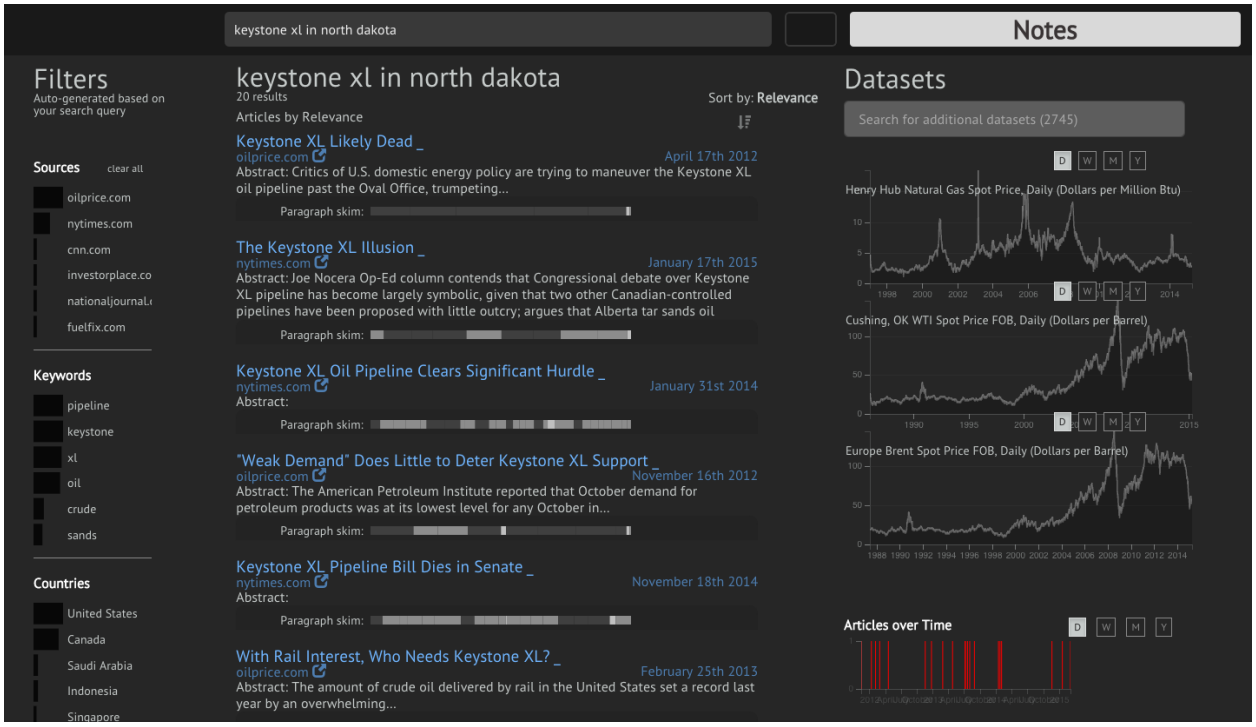
Mockups for the next iteration

With these valuable insights in hand, as well as the trove of feature ideas sourced from our initial research, we continued with mockups for the next iteration of the tool. We find it useful to have design meetings as a team to discuss next steps. In addition to lighting up the whiteboards, we often we annotated action items and mock-up new ideas right on top of screenshots of the existing app (see image below).





Front-end: Functional Prototype v0.2



In Prototype v0.2 (pictured above) we introduced a number of new features from the roadmap in addition to modifying existing elements based on user feedback.

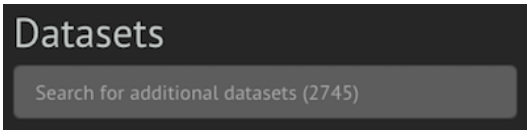
1) **From 2 to 3 columns**

   User feedback and lessons in visual hierarchy led us to a three column layout to address some of our initial issues with users being confused about what the main page element was. Text entity filters were now on the left and stripped of their color to send them into more of a background role.

2) **Skim tile interactions**

   Tile bars are now interactive. Clicking on them brings up a custom popover that displays the full text of the paragraph they represent. Add spacing between the tiles to improve readability of the visualization.

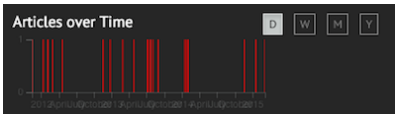
3) **Separate dataset search**



   In order to provide access to the many EIA datasets indexed in our collection, we provide a dedicated search bar with

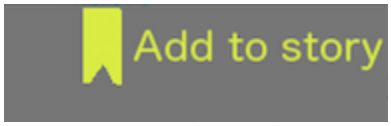
autocomplete enabled. Adding a dataset will populate the right column with an additional line chart.

4) **Time series aggregations**



We added the ability to choose how granular to display time series data (both in raw datasets and the articles timeseries). Available options: Day, Week, Month, Year

5) **Bookmarking and story view [non-functional mockups in app]**



Though the top priorities were search and discovery features, we started sneaking in elements that eluded to what a bookmarking and story-building interaction might feel like. These were helpful as prompts in usability tests to spark both conversation and user's imaginations.

Key user observations and feedback

Articles time series filter lacks affordances to indicate it is interactive - Users doing poorly on the task of filtering results by time. The feature is generally praised as useful, but doesn't "look like something I can click". "I thought it was a static graph". Once they try to interact, first reaction is generally a click (not a click-and-drag). Reaction: We want to make the brushing handles enabled by default, but this is proving difficult as it requires a customization to the dc.js library.

Home page issues - The search bar is much bigger than it needs to be. "You could make this search bar a lot smaller." Also, users seem to be taking a longer time than expected to understand what the page is about. Design decisions: Reduce search bar size; move it to header to align with rest of the app. Dig up some richer images to "get them in the mood". Lower the majority of text to below the fold (less overwhelming).

Sorting is unclear - "Why are you showing things from 3 years ago?" "I expected something more recent" "How is this sorted?" Its unclear. Design decisions: Add sort options: By date (newest first), by date (oldest first), relevance

Skim bars are cool, but implementation needs work - Continued praise for skim over other features. Many felt it was hard to decipher the various shades of the tiles. Some of tiles are too small to click. Reading horizontally is a bit odd at first for some people. People are passing the "light is more" test (insight from previous iteration). Low discoverability that tiles can be clicked. Having to click each one is too tiring. Design decisions: Introduce a minimum bar size to get around tiny tiles issue. Study shades that have a clearer contrast to the dark gray background. Change click interaction to a hover.

Its not clear that bookmarking adds items to your story - Seems to be general disagreement between user tests about what bookmarking would achieve. Design decisions: Align bookmark icons and color to story icon and color. Improve tooltips to indicate which elements are adding content to stories.

Story feature is a bit of a mystery to people - Many people did not guess that they could build stories using this tool. "Does it create a watch list of topics I'm interested in being notified about?" Reaction: It's still unclear whether users will find value in building stories directly from their search results. More fleshed out prototypes of this feature are needed to explore further.

Keyword highlighting style is confusing - "Underlining and bolding the search terms is annoying, pick one.". The underlines make them seem like links, yet clicking on them does nothing. Design decision: Highlight keywords with a contrasting background color.

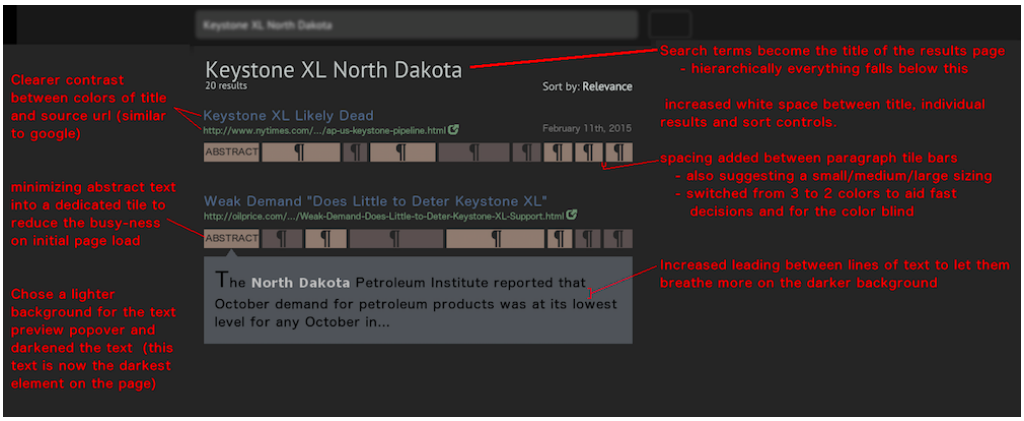
The graphs are interesting but they are too small - "I'd like to click to enlarge this small chart." Design decision: Dedicate prime real estate above the fold to a single chart. Allow the user to choose to expand which chart they are most interested in viewing

Lacking sources I would expect to see - Our more energy-savvy interviewees called out some missing sources that could be useful as datasets: CPUC, CEC. Reaction: This confirms feedback we gathered in our initial research. Its good to know what other data sources would be useful, but for the scope of this project we are intentionally limiting our collection.

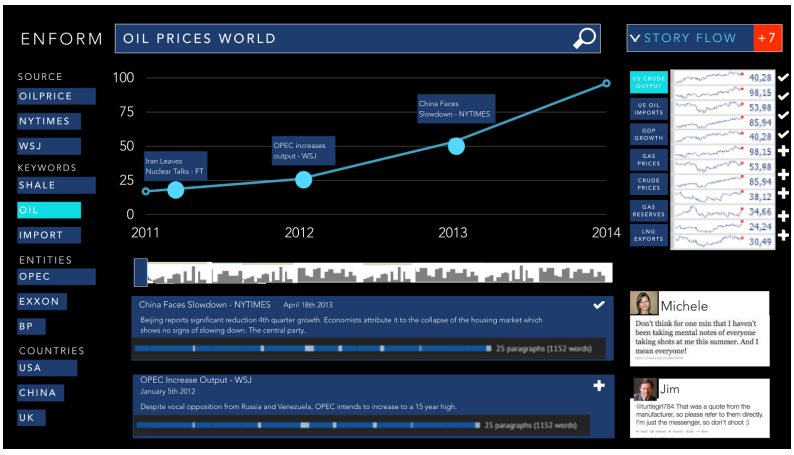
Search card colors are not optimal - Colors are hard on the eyes. The strong contrasts of deep blues aren't working. Design decisions: Create a color palette that works well with dark background.

Search card colors are not optimal - Colors are hard on the eyes. The strong contrasts of deep blues aren't working. Design decisions: Create a color palette that works well with dark background.

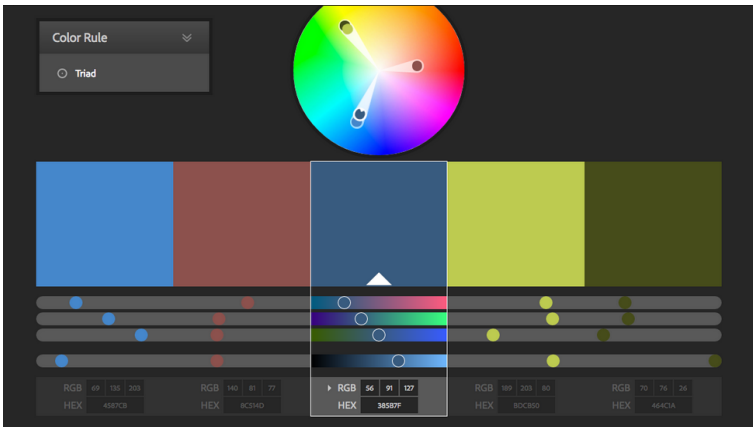
Mockups for the next iteration



Mockup of new search card design with annotations

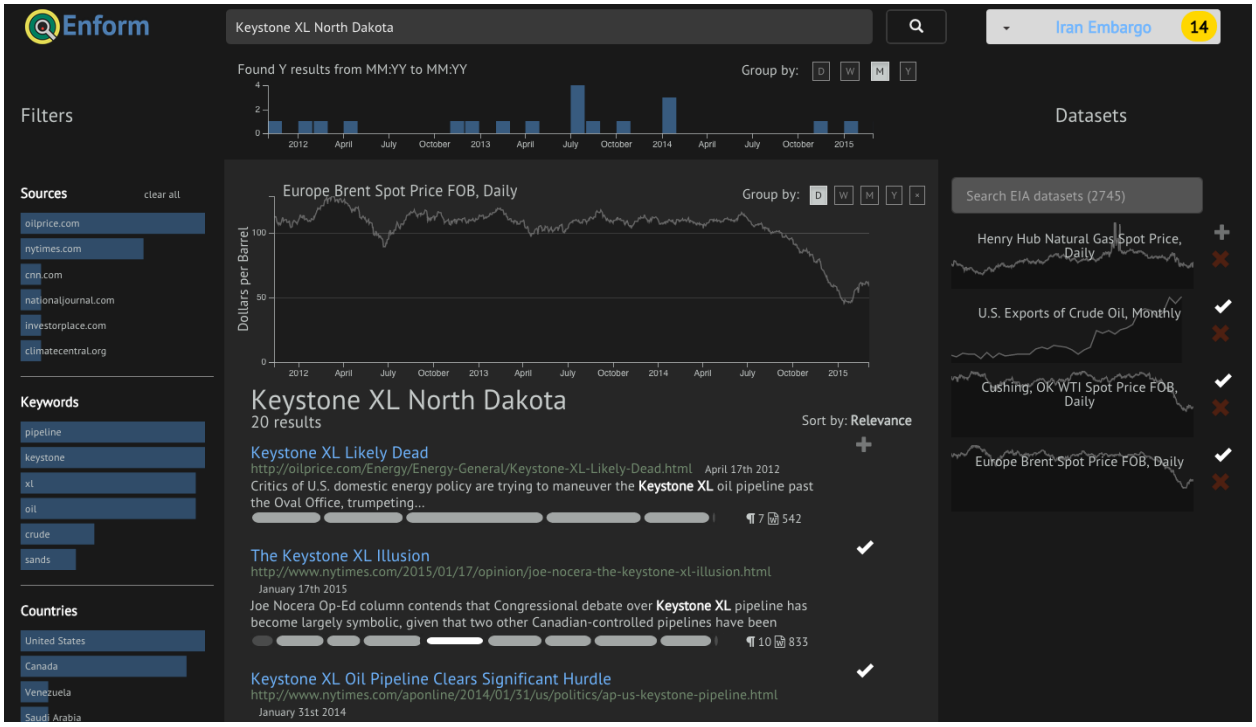


Mockup of a story view with auto-generated infographic



Exploring pale colors for bookmarked items

Front-end: Function Prototype v0.3

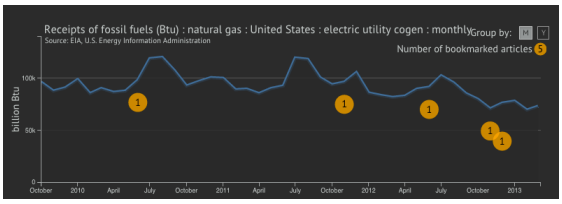


Prototype v0.3 saw a number of feature enhancements as well as significant aesthetic improvements thanks to inspiration from design critiques and lectures by Prof. Ryokai and Lisa Prescott's in their Interface Aesthetics course.

1) **Functional bookmarks**

Users are able to bookmark both articles and datasets to their active story. The colors and symbols used for bookmarking have been aligned to those used in the story menu.

2) **Story mode with automatic infographic**



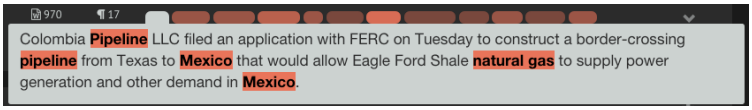
Bookmarked items appear in story view in two forms:

- As a time series infographic overlaying article occurrence with the dataset of interest. Other bookmarked datasets can be easily swapped out.
- As a list of items (similar to search results) and open comment boxes for quick annotating.

3) Story action menu

This feature adds the ability to create, switch and delete stories.

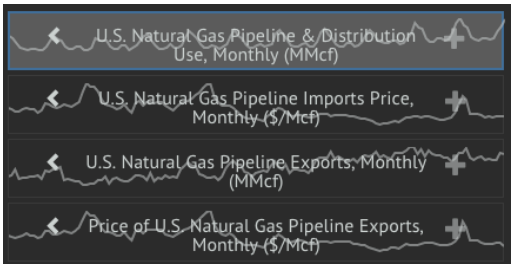
4) Visual design improvements to skim tile bars



We introduced a rounded-corner look to the tiles in an attempt to further distinguishing them as unique, interactive elements. Color contrast was improved. Hover interaction visually connects tile with custom tooltip. Improved readability of paragraph text.

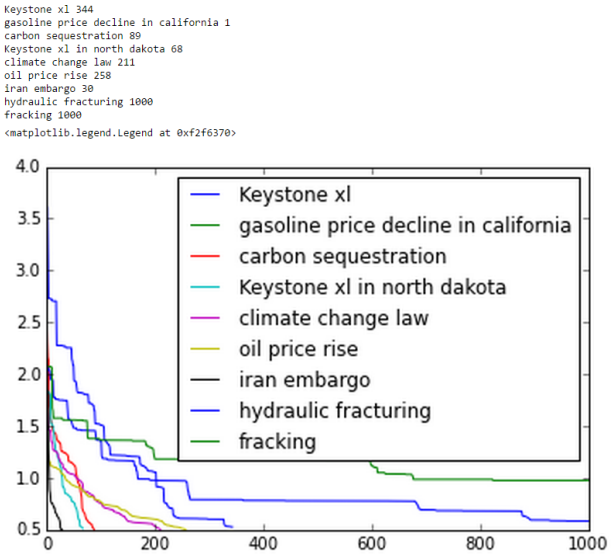
5) Dataset previews as sparklines

In an effort to simplify the dataset views and improve the selection mechanism, we reduced previews of them to a title and simple sparklines (no axis or ticks). A single click promotes the sparkline to the primary dataset on display.



6) Improving search speed.

We discovered that the quality of the search results diminished dramatically after 300 results. This meant that we could reduce the overall results to improve speed of the system.



Key user observations and feedback

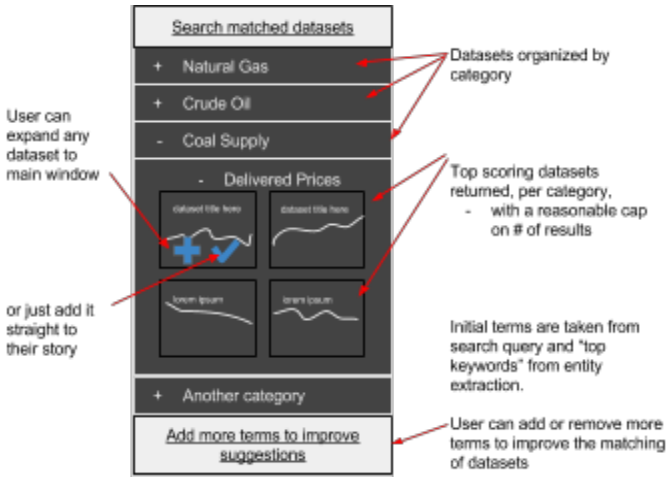
Sources of data are unclear - Multiple users failed to notice the source of the datasets. Design decision: Add a “source” label to primary charts.

Hierarchy issues between dataset picker and primary chart - Recent design critique exposed the idea that the sparklines do not seem to be related to the primary chart (though they are intended to directly control it). Design decision: Create a full-width band with a lighter background to connect primary chart area with sparkline chooser. Restrict sparklines to a scrollable container.

Toggle button for stories is hard to find - When asked to switch to story mode, users would scan the whole page, often taking longer than 10 seconds before realizing that the toggle was in the top right corner. Design decision: Redesign the header with clearer icons and larger button target to make the call-to-action more apparent.

Mockups for future iterations

In response to an improved algorithm for dataset suggestions, we created some mockups for ways that an improved selection interface would look.



An alternative story view idea that plays a timeline animation of the user's collected stories.

Front-end: Functional Prototype v0.4

Prototype v0.4 is our latest version as of this report writing. The focus was on a personalized experience and providing educational (tutorial) hints to help users understand the rich feature set available to them. We have not had a chance to test any of these new features with users.

1) **Support for user names (personalized story lists)**

By setting a username you receive a personalized space for your stories. The default user is always “system” which has a number o

2) **Header redesign with master toggle between results and story**

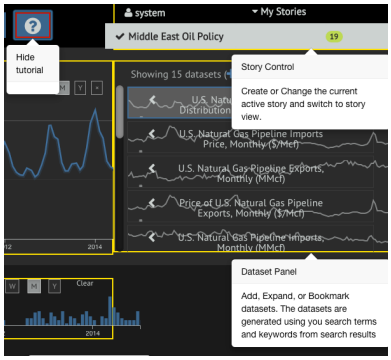
New header with animation that exaggerates the toggle between search results view and story view. It also includes a number of behind-the-scenes layout changes and updates to more relevant icons.

3) **Removal of dedicated dataset search bar**

The dataset chooser was a major dispute item for the team. One side argued for simplicity that restricted all search to the primary search bar, while the other maintained that a separate search bar for data important as it was dedicated to datasets and provided full access to all of them. The ultimate compromise was an improved “suggested” dataset search function and an “load more” button to retrieve more suggestions based on top keyword matches.

4) **Tips while loading (tutorial)**

With limited time to refine our interface to be more intuitive, we opted to provide some up-front training to new users, pointing out how they can interact with the major features. This tutorial appears while the page loads.

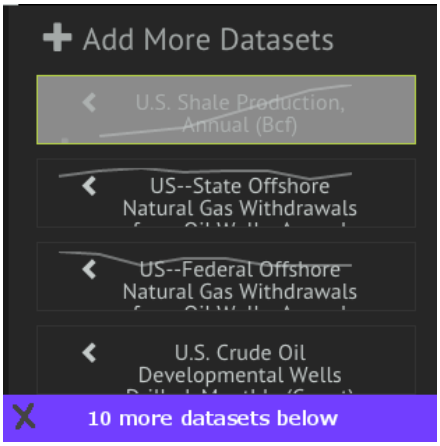


5) **Contextual help popovers**

As an alternative idea to “tips while loading” we are also experimenting with a default set of help popovers that are positioned directly on the app in the locale of the feature they are describing.

Mockups for future iterations

Loading new datasets needs a visual cue when complete



The ability to toggle individual facets is a simple, useful addition

Front-end: Design Aesthetics

In a system as information rich as this one, understanding how to build a clear and pleasing visual display is especially important for training the user how information is related and what interactions the controls afford. Four main topics kept recurring throughout the project: information hierarchy, typography, color and grid. The ideal interface is generally one that stays out of the way, allowing its users to focus on the content within.

EnForm’s interface is far from ideal and we made a lot of poor aesthetic choices along the way. Thankfully our usability tests and design critiques caught a lot of them, but there are still more to deal with.

The color scheme was a topic of contention (and still is) within the team. We held one constant throughout the development process: a dark background. Everything else was up for debate and change, but the dark background was to remain as a challenge to think differently about color and space within an application. We learned quickly that everything really does play differently with dark themes. Your typical default interface elements are no longer just “plug and play”. It forced us to really think through every pixel, and it was a lot of fun.

Reflections

Without doubt, the front-end design represented the difficult and often contentious portion of the development process. It was also the most fun! We had a lot of wins along the way that have contributed to the novel features that EnForm brings to the table, like:

- The skim tile bars (a.k.a. paragraph pills, a.ka. data beans)
- Refinements to the filters that have resulted in a novel and useful hybrid of faceted search with a distribution visualization.
- The time series chart of search results

Its clear that the app in its current state has a higher learning curve than say, Google Search. But we think people are willing to learn if it adds new value for them, and thus we think an investment in contextual tutorials can help with that.

Our commitment to good software engineering practices enabled us to remain nimble with our functional prototypes. We had an great architectural baand we built interface templates that provided flexibility as we iterated through many phases.

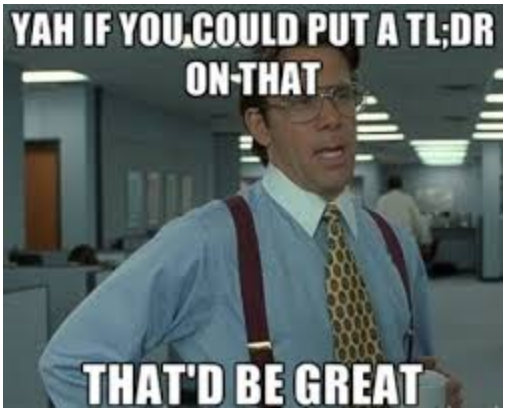
The story mode is still a wildcard component of the app. Feedback on it has been both hot and cold. There was concern among the team early on that this feature was more of a “nice to have” and could be a distraction, but we reached a compromise that in a short, provocative project like this, it is beneficial to err on the side of experimentation than perfection.

Our open debates on the story functions include:

- Is more granular paragraph bookmarking a compelling value add?
- What should the default ordering of the story be (by publish date, or by time added by the user, or something else completely)?
- Is entering the story view like entering a different mindset, and should we then design it for a fundamentally different user experience?
- Should elements bookmarked in the story view keep a consistent look and feel with the search results that they come from?

... the jury is still out

Conclusion



Our goals for the project were ambitious and the experience has been incredibly rewarding. With more than 15 customer interviews, 30+ usability tests, 5 design critiques and well over 10,000 lines of code, we learned a lot about research and hopefully sparked some interesting discussions about how modern day tools can improve the experience for the information foragers out there. We wanted to provide an unified platform to enable quick search and visualization of both data, research, and semantic information, and in many aspects we succeeded. We are especially excited that we developed the platform to be content agnostic, meaning it can be applied to any sector, and we hope incubate the project inside the Berkeley Institute for Data Science.

Reviewing latest design with Professor Marti Hearst (May 2015)

Appendices

Appendix 1: Compare Stanford NER with NLTK NER

Stanford NER	NTLK NE
{ "DATE": ["June 20", "September", "yesterday", "September", "a year", "June 14", "May"], "LOCATION": ["Tanzania", "Mozambique", "East Africa", "Kenya", "Tanzania", "Indian Ocean", "Norway", "Tanzania", "Tanzania", "Mozambique", "Dar es Salaam"], "ORGANIZATION": ["Bloomberg", "Statoil ASA", "Energy", "Indian Ocean coast.BG Group Plc", "Ophir Energy Plc"], "PERSON": ["Sospeter Muhongo", "David Malingha Doya", "Bryson Hull"] }	{ "GPE": ["Dar", "East Africa", "Indian", "Kenya", "Mozambique", "Statoil", "Tanzania"], "GSP": ["Uganda"], "LOCATION": ["Ocean"], "ORGANIZATION": ["BG Group Plc", "Bloomberg"], "PERSON": ["Bryson Hull", "David Malingha Doya", "Energy", "Mozambique", "Muhongo", "Ophir Energy Plc", "Salaam", "Sospeter Muhongo", "Statoil ASA"] }

Appendix 2: Components

Data Sources

- Structured: production, prices, economic and socio-economic metrics, environmental data.
- Geospatial: Geopolitical maps, street, view, satellite Imagery.
- Unstructured: News articles, expert reports, Audio/Video transcription.

Natural Language Processing

- Name Entity Extraction and Relationship Mapping
- Text Summarization & Synthesis
- Sentiment Analysis

Interface

- Visualizations of quantitative data from multiple sources
- Interaction design for navigating content
- Text synthesis presentation
- Geospatial presentation of the both data types.

Appendix 3: FAQs

Does this platform violate copyrights from the origins it pulls from?

No. Our method of breaking apart textual components and serving them via unique sort and filter mechanisms represents a transformative work. We provide a highly contextual view of sourced material and always include references the content origin.