Nikita Filatov

Human Computer Interaction

12/4/2014

Final Project Report

**Problem**

The NewsAggregator web application was designed to solve a simple problem with a not-so-simple solution: the ability for users of broad age ranges and computer literacy to view news articles from global news sources, all in one place. I frequently browse news websites and peruse message boards, blogs and threads in an attempt to find unbiased and accurate information about global current events. I consume too much of my free time actually looking for these articles, or going back and forth between sources and sites comparing and contrasting the information I have read. NewsAggregator is meant to allow its users to select the news sources they wish to see articles from, and see them visualized on a world map. The articles are placed in their respective location of occurrence, and the user can view the article within the same page, and quickly switch between different sources and articles. Ultimately, the ability to view articles from different sources quickly, and without navigating away from the NewsAggregator page, will allow users to save time while simultaneously reading more articles from various sources. Although potential users are anyone who care to stay updated on current events without wasting time researching, for the purpose of this project, our sample demographic was 18 – 40 years of age with average computer literacy. The tasks developed to test this application were the following: Filter articles by different topics; Click on an article on the map and view it within the NewsAggregator web page; Click on an article on the map and view it in a different tab.

**Design**

The design of NewsAggregator was based on the initial paper prototype, but with several significant modifications that were a result of 3 rounds of user testing, which will be discussed in the evaluation component of this report. At first, the paper prototype presented a version of the application with 3 main components: a panel on the left that contains filters for sorting the articles on the map, a panel on the right which displays the currently selected news article, and a toolbar on the bottom that contains a drop down menu and a time-slider component. The purpose of the filters panel was to allow the user to select exactly what types or news articles that cared to view, as well the news sources they wished to see. The bottom toolbar was there to allow the user to select a time range within which to view the articles, and the right panel was there to display a currently selected news article. The user would see news articles populating the map with several icons, which corresponded to the filters panel. The filters included the following topics: Science, Technology, Politics, Ecology, Military and Culture. It also included a geography tab which is meant to allow the user to zoom in and constrain the map to specific regions, but was outside of the scope of this project.

**Evaluation**

After the initial paper prototype design, testing ensued. The paper prototype testing was very useful is clearing up glaring UI issues that slipped my mind during the design process. For example, proper naming of UI elements such as drop down menus and links. While designing the UI I was aware what everything did and where it should be placed, but this was not obvious to the end user. Having a paper-prototype testing session was valuable because it removed these “elephant in the room” type issues. After this, the first digital implementation was created, and it used all the corrections made during the paper prototype stage. After the first digital prototype was created, a round of heuristic evaluations was in order. Several people from class tested my application and gave good feedback. I collected their results and aggregated the issues that everyone pointed out, and set about correcting them for the next version of the digital prototype. The issues I gathered from classmates testing results included:

-bottom right buttons turn into select pointer in hover, and don’t have functionality   
-weird clipping fiters panel   
-does not resize well   
-change military icon   
-add collapse functionality to filters and side panel -add zoom in/out gui   
-timeline numbers are too ambiguous, make actual dates   
-allow user to drag right panel width   
-go to source should open a new tab   
-geography should do stuff   
-add help/documentation/tutorial   
-popup is kind of confusing, add way to know if you’ve read a source

After going through and correcting these issues, the biggest changes I made were as follows: I removed the right-most panel and brought it over to the left side of the screen, to ensure that it would always be visible on any screen and would not cover the map too much. In addition, I also added functionality to allow the user to collapse this panel, to see more of the map. I also added collapsibility to the filter panel, allowing the user to collapse each filter section into a single row like an accordion, showing more of the map if they wished. I removed the bottom toolbar, and instead of having a horizontal slider component for the time selector, I added a new row to the filters panel, which contained checkboxes for the current day and the last 4 days. This simplified the time selection process, made it less error prone and more intuitive. This stage of implementation also was extremely significant because I began to implement a real backend. Before I was generating a lorem ipsum text for article data, and placing articles randomly within a constrained latitude/longitude to simulate the app working correctly. The implementation details are discussed in the following section. The last round of user testing was done with 3 users. 2 twenty-three year old caucasion males, college undergraduates, and 1 twenty-two year old Asian female, college graduate. Because the paper prototyping and heuristic evaluation stages were so effective, the majority of the feedback given included desired future prospects of the application. Things such as a search functionality, better ways to control the map (such as zooming in on regions) and overall responsiveness to allow the user to view the application on any device, were among the suggestions given. Overall, users were very satisfied with the application (particularly the functional backend), had no issues completing the tasks, and looked forward to using a completed version.

**Implementation**

NewsAggregator is a web application developed in HTML, CSS and Javascript. The UI elements are created using standard HTML markup, including the use of DIV tags for containers, and standard HTML5 elements such as checkboxes, vertical and horizontal scrollbars, links, and buttons. CSS was used to style the overall look of the application. I chose to go with a minimalist dark theme, which I plan on improving later, as well as adding a light theme option. CSS was also used to give small animations the buttons and clickable elements, improving feedback to the user. The left panel was able to slide in and out of the side of the screen with CSS transitions, and the filters panel could be collapsed in an accordion style with transitions as well. Javascript was used to add events to the HTML GUI elements through the use of JQuery. JQuery allowed me to quickly access elements on the page, and assign callback functions to mouse events corresponding to each element. The actual implementation used several different API’s and I am proud of the way everything is assembled. The page first loads a world map using the OpenStreetMap API. This API is like google maps, but has less limitations and allows for more queries-per-second. This map uses another extension called Leaflet.js, which allows for the creation of HTML5 elements on the map. I used this to create the actual icons which represented the articles. After the map loads, I use the Google Feeds API to load dozens of RSS feed URL’s into my application. Because of time constraints I was only able to implements a single news source: Reuters. However, I used multiple RSS feeds from Reuters to diversify the articles and allow the user to sort by topics. After the RSS feeds are loaded, I create javascript objects out of each article in the feed, and parse the XML format to create data within these objects. RSS feeds have a specific format which helped me immensely. Among the formatted data I gathered from within the feed included: Headline, Publication Date, Author, Source, Content and Link. Each of these components are stored in a JS object called Article, and displayed on the screen in various ways. The most challenging part of developing this application was how to assign location data to each article. Because “location” was not part of the RSS format, I had to search the content of the article for the location. Luckily, it is standard practice to include the city where the event occurred for news articles (apart from a select few articles which were global events or were too general to have a location). After searching string data within the content of the news article, I would find a location, and pass it on to the next part of my application. I used the Google Geolocation API to obtain latitude and longitude coordinates of each article by passing a location string obtained by searching through the article’s content snippet. After this, the articles and their corresponding icons are populated on the map using these coordinates. I ran into an issue with the Geocoding API, because Google only allows 3 queries per second. This dramatically slowed down populating the map with articles, so I had to come up with a solution. My solution was to use Firebase, which is an API that allows developers to store data at a URL location. Whenever an article is generated from an RSS feed, I first check if the location exists within my firebase URL, and if it exists, it is retrieved and the article is placed on the map without querying the geocoding API. If the location does not exist in my firebase URL, the location is geocoded, and then placed into firebase for later reference. This drastically sped up the rate at which article were parsed and displayed, and significantly reduced the need to query google’s database.

**Reflection**

I think that every part of the development process was interesting and necessary. Each testing phase was helpful and resulted in useful iterations. I was honestly surprised by how useful the feedback was from heuristic evaluations. I now know that if I develop another application, I can easily test the interface’s efficiency with online heuristic evaluations. There are a lot of resources online for testing your web apps, and now that I know how useful there evaluations really are, I plan on using resources like this in the future. These evaluations get rid of blatant and glaring issues in a relatively short amount of time, and most online resources are decently cheap. The issues that these testing results resolve are often issues that are not clear to the developer, because of their deep level of involvement. I found it extremely useful to have a list of problems that I could go through and correct. It let me focus on other aspects of development, such as backend, instead of sitting around and trying to figure out what could be wrong with my interface. Delegating the finding of errors and inefficiencies to testers sped up my development process significantly. In addition, I know that I can always do more testing iterations if I make changes. Overall, I enjoyed this project. It was an opportunity to develop an application that I was dreaming of doing, and gave me the chance to evaluate it with testers that gave legitimate feedback. I definitely plan on using this iterative test-based process in the future, and I also plan on continuing this project, by fully implementing the rest of the backend, as well as adding new features requested by testers, and testing these features as I did previously in the semester.