# User-Specific Content Aggregator

*by*
*Amit Bose (University of Pennsylvania)*
*Gary Menezes (University of Pennsylvania)*

# Contents

# 1. Development Process

Through our project we built an application that presents in front of the user various resources such as news, pictures, restaurants, etc that would precisely be of his/her interests. In this process, we used several APIs that would fetch large open data and filter them based on one's interest. Thus we focused mainly on building each of these smaller modules initially and then integrated them to complete the application.

This software development life cycle of ours followed the waterfall model. Our development method was linear and sequentially concentrated on the individual modules at a time. We had set distinct goals for each phase of development. The advantage of following the Waterfall model was that it helped us to departmentalize our work into modules and centralize ourselves on each one. Some phases that we went through where feasibility testing and specification requirement in which we initially thought of going through java servlets but after looking at the various APIs that we would use decided to go ahead with javascript and ajax. Each phase moved in order without any iterative steps. Development moved from concept, through feasibility testing and specification development, design, implementation and testing.

*Technologies used:*

UI - HTML/CSS
Backend: PHP, JavaScript, JQuery, Google Geocoding
API Response Object: JSON
Testing: QUnit

# 2. Problem Statement

The main idea of the project is to develop an interface where a user can view different information related to his/her interests.This is done by getting his/her interests from a social media site like Facebook and using various API's like New York Times, ESPN etc to get relevant information. The motivation behind the project is that a user may find it difficult to search for relevant information based on his/her interests. Creating a framework would aggregate the huge volume of data and present it in an accessible manner.

## User Stories

*(1) As a user, I want to have convenient guidance when I am acting as a tourist.*

*(2) As a user, I want to see data that is relevant to my interests and current location.*

*(3) As a user, I want to have a restaurant finder relevant to my interests and location.*

*(4) As a user, I want to have an interesting news reporter finder relevant to my interests and location.*

*(5) As a user, I want to know about sports updates relevant to my interests and location.*

*(6) As a user, I want to see interesting pictures based on my interests and location.*

# 3. Requirement Specification

Our project provides the user with various interfaces beginning from the home page through the complete flow till the user decides to log out. At the beginning, it provides the user with the home page containing a button asking the user to Login into his/her Facebook account.

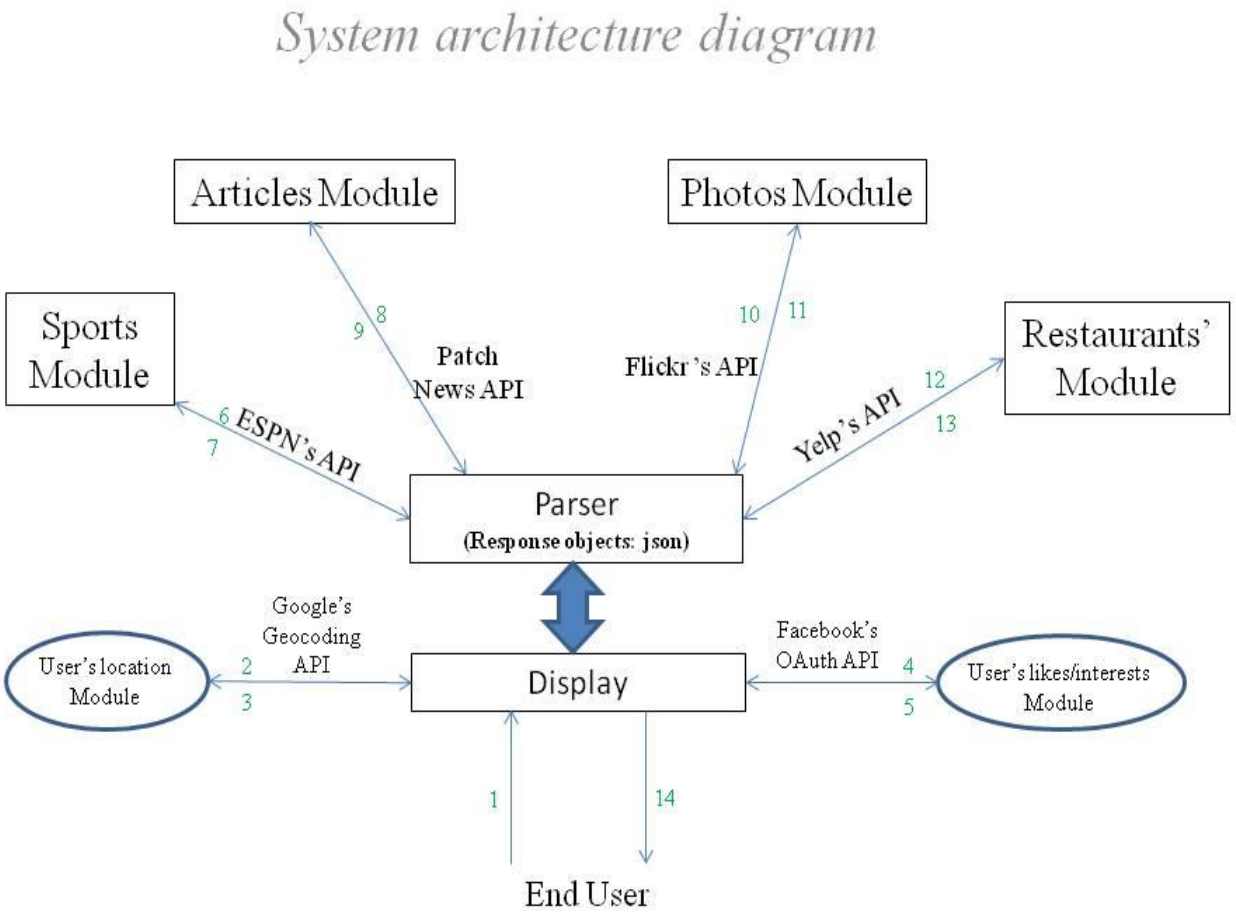
On clicking the button it provides the user with another page asking to enter the username and password credentials, which after being verified takes the user to the next page which has the four buttons -- photos, sports, news and restaurants. Before this interface is provided to the user, we have already filtered out such fields of information specific to the users' interests which is fetched from their Facebook account. In addition, the filtered information we provide is also based on the users' present location. Now, the user can follow the link to any of the four categories to fetch information in that category that matches his/her interests. The user can always return back to the main menu using the menu button provided on each of the respective pages for photos, news, sports or restaurants. Finally the user can log out from his home page using the logout button. A more elaborate description with screenshots is provided in section 8 - 'Outcomes and Future Scope'.

The above information is reiterated here in an easy-to-reference format for use in the accompanying traceability matrix:

1. Home page with a Facebook log in button.
2. Home page with a menu linking to {sports.php, photos.php, news.php, restaurants.php}
3. Following the link for each of the 4 categories:
   a. Generate a list of tags from the logged in user's Facebook profile.
   b. If necessary, get the user's current location.
   c. Fetch a specified number of items.
   d. Format those items as HTML.
   e. Insert those items into the DOM.
   f. Paginate those items into the specified number of items per page.
4. Return to the Home page via a menu link.
5. Log out.

# 4. Architectural Design

## 4.1 Diagram




***The flow is numbered in Green***

## 4.2 Description

The system architecture is modeled as a sequential pipeline of calls to various external web services with some minimal client-side processing. The system can be regarded as the composition of four phases with distinct objectives: (1) determine user identity, (2) generate keywords, (3) retrieve user-relevant content and (4) display aggregated content.

Both Facebook profile information and geolocational position data provided by Google are used as 'filtering attributes', intended to help isolate content relevant to service users. Both Facebook and Google offer APIs that enable easy interfacing from client-side code. Information is brought

into the system as JSON-formatted objects, which are then parsed into attributes and captured locally in dynamically-generated Javascript (JS) containers.

With user attributes appropriately parsed and made accessible, the system proceeds to query a range of supported third-party content providers (also through their APIs). These requests are formatted by embedding parsed information from Facebook and Google to content providers as API call parameters. The results are pre-filtered on the server side and returned in convenient JSON format to the client. The output of the call is then quickly formatted into an HTML page which is dynamically-generated and displayed to the user.

An important caveat arising from the use of certain third-party APIs (and the reason the service is deployed entirely in client-side code) is that persistent storage of retrieved data is strictly prohibited. Therefore, the system's behavior is not only pipelined but also executes on-the-fly in response to dynamic queries from embedded JS. While this would usually compromise efficiency, the impact on service performance is minimized by the overall limitation on application scale given the small number of APIs involved and the retrieval of largely textual data. The result is a system that is sufficiently responsive.

# 5. Project Plan

| Phase | Objectives | Period | Duration |
|---|---|---|---|
| Development of concept | Building an idea to work with large open data. Set user stories. | 11/15/12-11/19/12 | 5 days |
| Feasibility test and specification requirement | Checking out the feasibility of our approach and alternatives | 11/20/12-11/23/12 | 4 days |
| Architectural design | Agree on the overall design | 11/24/12-11/28/12 | 5 days |
| Abstract | Submission of the abstract | 11/28/12-11/29/12 | 2 days |
| Implementation/Construction | Implement the various APIs and filter them to get better results. | 11/30/12-12/13/12<br>Flickr - 2 days,<br>Yelp -3 days,<br>Patch News- 3 days,<br>ESPN - 2 days,<br>Facebook and Geo Location - 5 days | 2 weeks |
| Integration of different modules | Integrate all modules and work on UI | 12/14/12-12/27/12 | 2 weeks |
| Validation and verification | Unit test and integration testing | 12/28/12-01/15/13 | 2.5 weeks |
| Demonstration to professor | Demo Preparation | 01/16/13-01/20/13 | 4 days |
| Project Report | Drafting,revising and finalizing | 01/21/13-01/29/13 | 10 days |

# 6. Implementation

## 6.1 Summary

The application is deployed entirely as embedded, client-side Javascript.

Our application uses four APIs (ESPN, Patch News Api, Flickr, and Yelp) to fetch relevant information and content for a user. To make sure that content displayed is relevant we fetch the likes and interests of a user from his/her facebook account. We initially ask the user to log in to their account, and using Facebook's Javscript SDK, we extract most of the user's likes and interests. This is all handled in a separate Javascript file (**fb_login.js**). In this file we have a function named **getUserTagsAndDisplay**() that specifically fetches a user's likes, movies, books, groups and events. Our tag generation is very simple and was significantly limited by our testing set of one facebook account. We collect the users' data into a string of words, remove common delimiters, and use the remaining words as tags. We pass the 'tags' array as a GET parameter to each of the four API modules that fetch relevant content.

Each of the four APIs is implemented in a separate Javascript file. The general structure of methods in each API module is the same three step process, though. First, there is a method, **displayCategory()**, wrapping the ajax call to module's primary API and, if needed, Google's geolocation API. Second, there is a method, **renderCategory**() and helper method **formatCategory()**, which take the fetched items and insert them into the page's html. Finally, a call is made to a method, **generate_pagination**(), in a shared file, that looks at the page and finds html elements of a specified structure and replaces them with a Javascript-paginated element.

## 6.2 Module Specifics

*Patch News API*:
We have implemented this API in a file named **news.js.** In a function named **codeLatLng()** we use the geocoding API to fetch the present location of the user in terms of latitude and longitude. The location information is used to provide the 'state' and 'city' in the Patch News API call. We set the response object format to be 'json' and then parse it to extract the different news stories. We render the output and insert them into the page's html using the function **renderNews()** and **formatNews()**. Finally, **generate_pagination**() is called.

*ESPN API*:
We have implemented this API in a file named **sports.js.** The method structure here is identical to the News API and the implementation has only a few minor differences. The API ajax call is, of course, to the ESPN API rather than the Patch news API, and the **renderSports()** and **formatSports()** function formats the retrieved json objects differently.

*Flickr API*:
We implemented this API in a file named **photos.js.** The function from where we make our API call to Flickr's flickr.photos.search API using ajax is named **displayPhotos()**. We use the elements from 'tags' (the array consisting of all the likes,movies,events fetched from a user's Facebook account) as the set of filtered keywords used in the Flickr API call. Here too we request our response object format to be json. Finally, the **renderPhotos()** and **formatPhotos()** function formats the retrieved json objects as html image elements.

*Yelp API:*
This API has been implemented in the file named **restaurants.js.** The function structure here too is similar to the News API with the same use of the **codeLatLng()** function. Similar to the Flickr API we use the 'tags' generated as our query when making our Yelp api call. Our json response object elements are then rendered using **renderRestaurants()** and **formatRestaurants().**

# 7. Verification and Validation

Our testing strategy focused on automated tests but utilised manual testing under special conditions.

We used the QUnit javascript testing framework to test requirements 3a through 3f:

3. Following the link for each of the 4 categories:
   a. Generate a list of tags from the logged in user's Facebook profile.
   b. If necessary, get the user's current location.
   c. Fetch a specified number of items.
   d. Format those items as HTML.
   e. Insert those items into the DOM.
   f. Paginate those items into the specified number of items per page.

Testing these features in an automated fashion was made difficult by the API calls we used. These API calls were all asynchronous, and added an inconvenient non-determinism to our process. We worked around this to some degree by refactoring in order to have display formatting and other synchronous steps in separate methods. These methods are tested with canned data.

For testing asynchronous calls, we used QUnit's asyncTest feature which simply pauses interpretation of the test until the 'start()' method is called. This allows us to set a timeout (javascript's setTimeout) for which to wait before running a given test's assertions. This timeout allows the remote API call to complete. The remaining problem is that our selected timeouts are sometimes not sufficient due to varying network / API latency. In this case, some tests take too long and fail and the developer is instructed to rerun the test.

In the special case of Facebook federated log in (requirements 1 & 4), the code we used is heavily derived from the Facebook documentation. Also, requirements 2 and 5 are very simple php implementations. To avoid more non-deterministic testing, we tested these features manually.

The complete testing for our application was done using QUnit testing. Given below is a screenshot of our test cases and results.

# SCORE Tests

☐ Hide passed tests ☐ Check for Globals ☐ No try-catch

**Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.97 Safari/537.11**

Tests completed in 44439 milliseconds.
442 assertions of 442 passed, 0 failed.

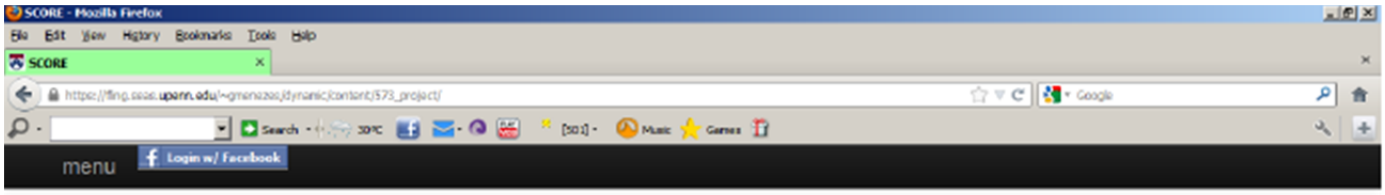
1. Sports: **formatSports zero items (0, 1, 1)** Rerun
2. Sports: **formatSports one item (0, 1, 1)** Rerun
3. Sports: **formatSports multiple items (0, 1, 1)** Rerun
4. Sports: **displaySports negative items (0, 2, 2)** Rerun
5. Sports: **displaySports zero items (0, 2, 2)** Rerun
6. Sports: **displaySports one item (0, 13, 13)** Rerun
7. Sports: **displaySports multiple items (0, 37, 37)** Rerun
8. Sports: **displaySports multiple pages (0, 33, 33)** Rerun
9. Photos: **formatPhotos zero items (0, 1, 1)** Rerun
10. Photos: **formatPhotos one item (0, 1, 1)** Rerun
11. Photos: **formatPhotos multiple items (0, 1, 1)** Rerun
12. Photos: **displayPhotos negative items (0, 1, 1)** Rerun
13. Photos: **displayPhotos zero items (0, 1, 1)** Rerun
14. Photos: **displayPhotos one item (0, 10, 10)** Rerun
15. Photos: **displayPhotos multiple items (0, 22, 22)** Rerun
16. Photos: **displayPhotos multiple pages (0, 18, 18)** Rerun
17. News: **formatNews zero items (0, 1, 1)** Rerun
18. News: **formatNews one item (0, 1, 1)** Rerun
19. News: **formatNews multiple items (0, 1, 1)** Rerun
20. News: **displayNews negative items (0, 1, 1)** Rerun
21. News: **displayNews zero items (0, 1, 1)** Rerun
22. News: **displayNews one item (0, 13, 13)** Rerun
23. News: **displayNews multiple items (0, 37, 37)** Rerun
24. News: **displayNews multiple pages (0, 33, 33)** Rerun
25. Restaurants: **formatRestaurants zero items (0, 1, 1)** Rerun
26. Restaurants: **formatRestaurants one item (0, 1, 1)** Rerun
27. Restaurants: **formatRestaurants multiple items (0, 1, 1)** Rerun
28. Restaurants: **displayRestaurants negative items (0, 1, 1)** Rerun
29. Restaurants: **displayRestaurants zero items (0, 1, 1)** Rerun
30. Restaurants: **displayRestaurants one item (0, 24, 24)** Rerun
31. Restaurants: **displayRestaurants multiple items (0, 92, 92)** Rerun
32. Restaurants: **displayRestaurants multiple pages (0, 88, 88)** Rerun

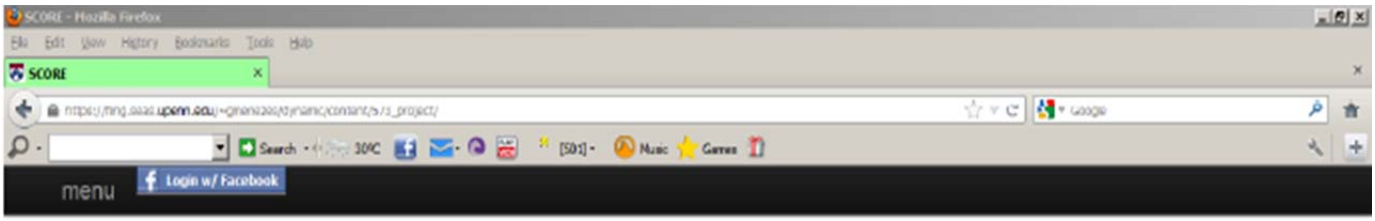
# 8. Outcomes and Future Scope

## Screenshots of our application

*The following guide briefly describes usage for the application and is accompanied by screenshots to inform navigation.*

**(1)** Home screen with Facebook login button. Click on the ‘Login with Facebook’ to pull up the login-screen popup.

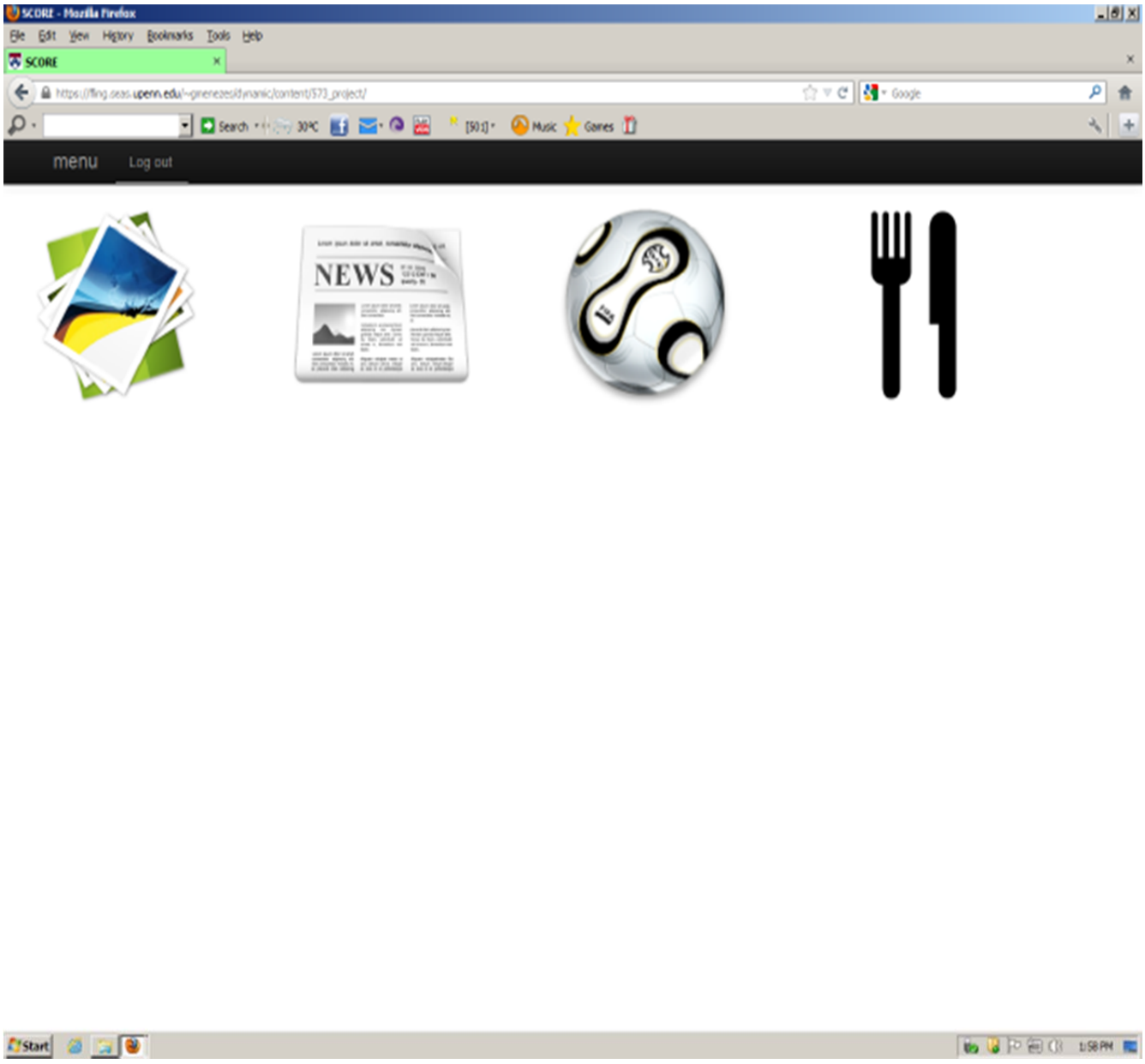




SCORE - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SCORE
menu
Login w/ Facebook

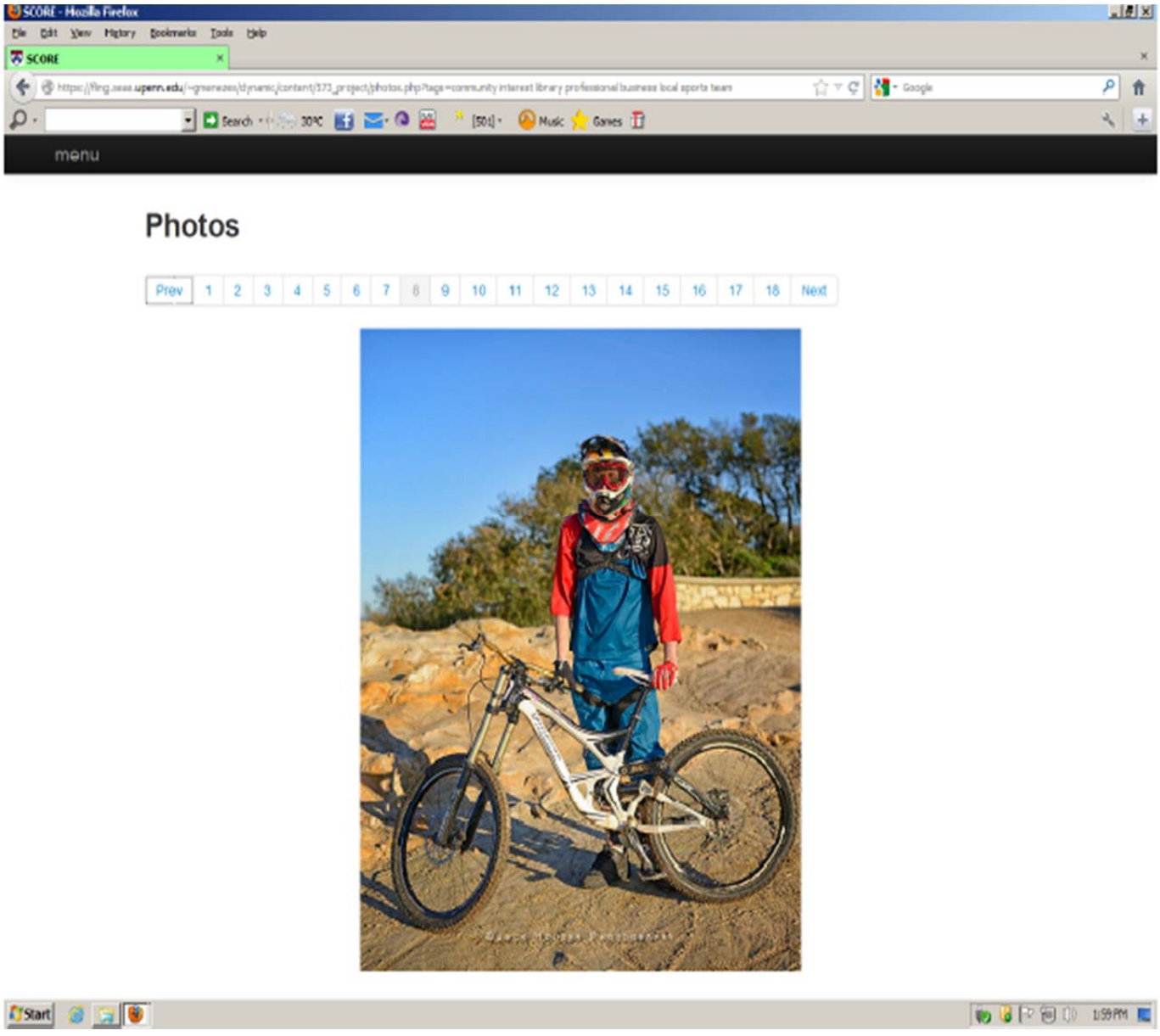

Log In | Facebook - Mozilla Firefox
Facebook
Log in to Facebook to personalize your experience with this social plugin.
Email or Phone:
Password:
Keep me logged in
Forgot your password?
Sign up for Facebook
Log In
Cancel

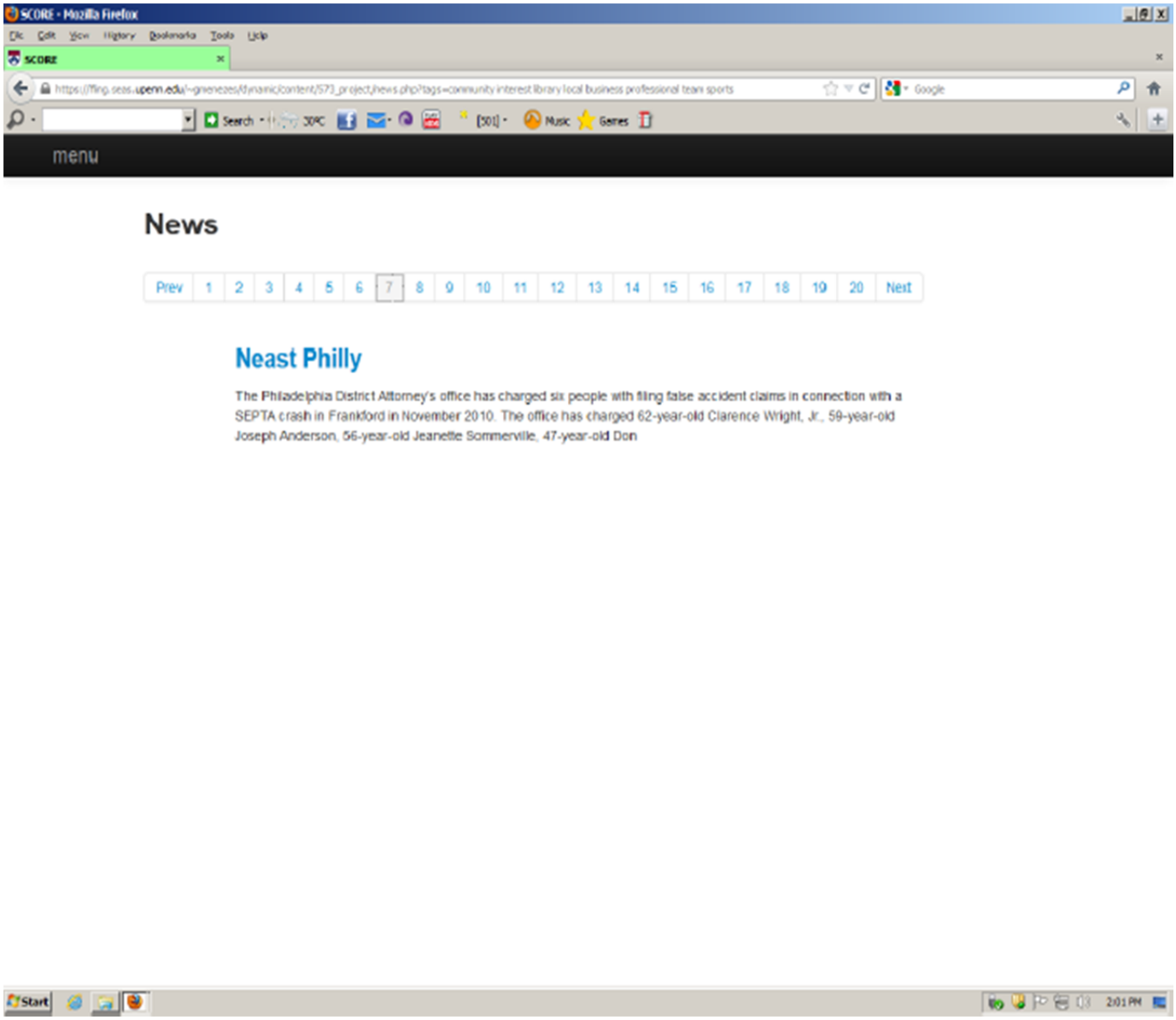

Start
1:57 PM

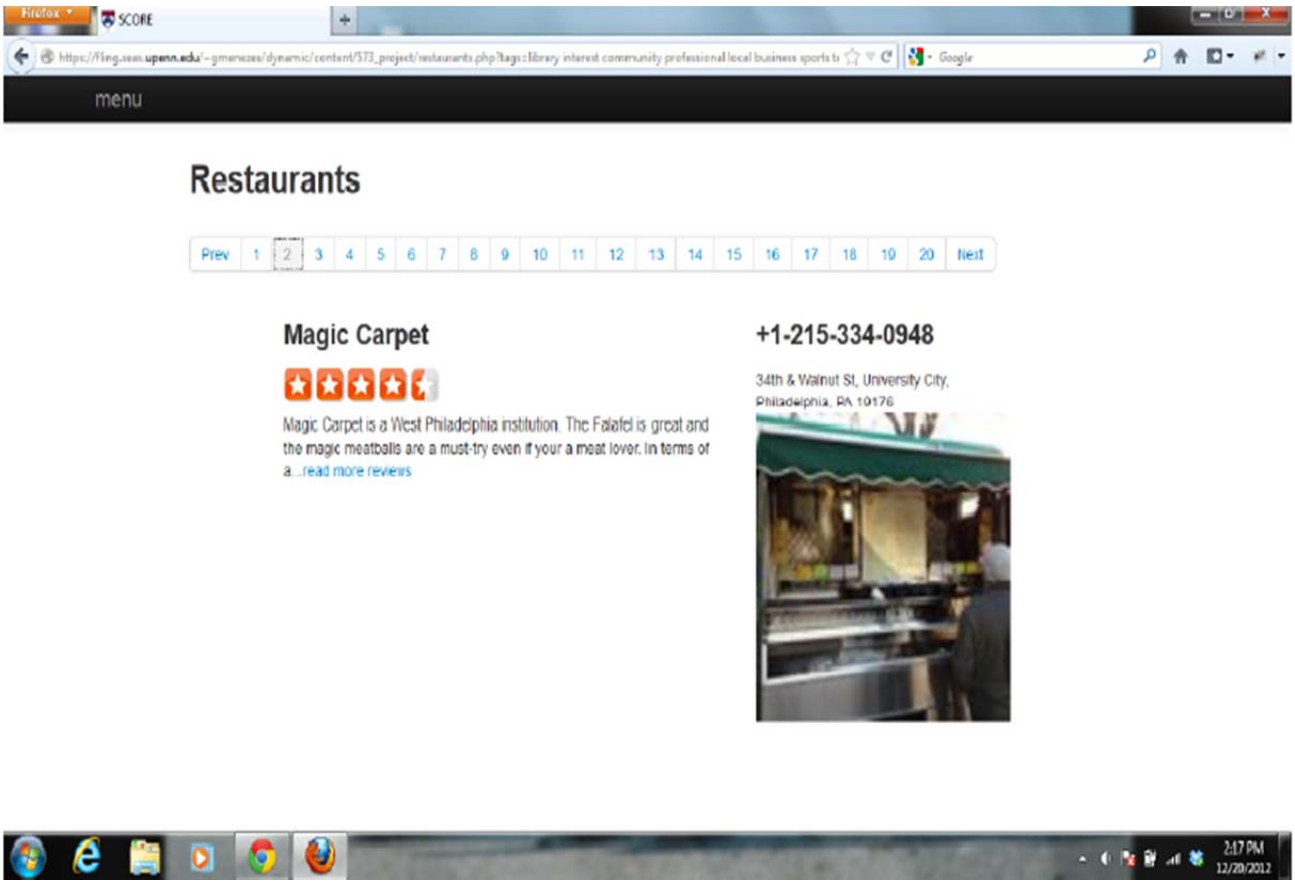
**(2)** Upon validating, the application will open with pictorial tabs for each category supported. In the screenshot, the restaurant category is indicated by a fork and knife, sports by a soccer ball, news by the newspaper, and photos by the photo stack.



**(3)** Each screen pulls up an enumerated navigation bar, which defaults to the first tab. Navigate content using the tabs. Each access to a particular category generates a variable number of results and these change according to updates by the content provider. The following are the Photos, News, and Restaurants categories.




SCORE - Mozilla Firefox
File Edit View History Bookmarks Tools Help
SCORE
https://fling.seas.upenn.edu/~gmenezes/dynamic/content/57/_project/news.php?tags=community interest library local business professional team sports
Google
Search
Music
Games
menu
News
Prev 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 Next
Neast Philly
The Philadelphia District Attorney's office has charged six people with filing false accident claims in connection with a SEPTA crash in Frankford in November 2010. The office has charged 62-year-old Clarence Wright, Jr., 59-year-old Joseph Anderson, 56-year-old Jeanette Sommerville, 47-year-old Don
Start
2:01 PM




## Future improvements

Our final product, while a complete experience for the user, leaves a few user stories unanswered. With these in mind, some further improvements to this application are:

1. Allowing users to specify locations for search rather than just using their current location.
2. Adding more categories / APIs. Our design is specifically oriented towards easy addition of categories, and by following the structure we have laid out, adding any number of categories is a simple task.
3. Preloading category items. In our current implementation, there is a couple seconds' delay when loading a category page. An obvious solution here would be preloading data while the user is doing nothing, but there are a few hang ups that we did not figure out in time. The solutions we came up with were:
    a. Cache categories server side. The problem is that this would require separate caching for users' unique results. And, since most of these APIs disallow storage of data, we would have to cache in memory.

    b Cache categories client side. We considered using html5's 'localStorage', but are not sure of the precise definition of 'saving data' in these APIs' terms, so we passed on this idea.

4 More intelligent filtering. Our current implementation has a very rudimentary tag generation from Facebook data. We remove common delimiters (the, at, a, 's, …) and consider the remaining words to be good tags. This is usually not true.
    a One easy improvement here would be to use a larger dictionary of common words in addition to the current delimiters. Removing all these would result in a more unique and relevant tag set for each user.
    b Another improvement would be to generate different tag sets for different categories. For example, we could have a Movies category with tags that are derived from the users' Facebook movie likes.