## Front-end application:

A front-end web application is used to let users view the results of their typed query.
Initially, play framework is applied to our project because of its easy to use and flexibility. After it was tested out using the first homework in which CGI scripts were used to render searched results to user, play framework was put a step back in our front-end application. There are mainly three reasons why we gave up play.

1. Play does not integrate well with IDE, which is intellij here. There are some compiling and dependency issues that are out of control
2. Play seems to have major changes between different versions. It is being wasting time to try to find the solution for the correct version
3. Intellij built-in JettyBean server is slow to start. Once started, laptop is in a very slow state, which is because the team member who takes care of front-end does not have a decent laptop.

AngularJS and nodeJS then caught our attention because of MVC feature, responsive design and fast performance. We first deploy the whole process at local laptop. The work-flow is described as the following.

- NodeJS starts a server that host web page on port 8080.
- Put localhost:8080 on browser. A web page shows up and let user type in keywords she wants to search. This query is further converted to a http GET request ($http.get('/search?query=' + $scope.queryWord + '&max=100' + '&pageResults=10&page='+pageNum) in controller and this request is parsed by api.js on node, which encodes query. At this step, node is listening on a “/search” context, it will parse the search request once it comes.
- The encoded query is then sent as a GET request to another server that back-end java code starts on port 23456.
- After java code processes the request, the response, which is represented as an array of Page JSON object, is sent to controller. Then $scope fields taking care of how returned results are rendered are changed. For example, if there are pages returned for a query, web page containing pages information such as url, title and preview will be rendered. If not pages found, a plain text web page showing like “No documents found for this keyword. Please try something else.” will be shown to user. Since the data-binding feature, the corresponding components in .html will change. Therefore, user is able to see the return results.

Since we crawled about 960 thousand pages, which is about 30GB disk space. It will be great if we can take advantage of the fast performance of cloud services. Then we moved all our back-end part including crawled pages and java source code and associated libraries to Google App Engine, leaving only nodeJS code, front-end html web page, static css and javascript files on local. In order for the communication of node side server with back-end java server on cloud, a http request is sent from node server to remote server using our cloud instances’ external IP

address. Note that domain name can not be used here, only IP address. Since by default google cloud does not allow http traffic from outsource, an allow-http rule is set up using "$ gcloud compute firewall-rules create allow-http –description "Incoming http allowed." – allow tcp:8080 –format json".

## Details of how queries are handled:

1. Create HttpServer in Retriever.
2. Create QueryHandler class. Use HttpHandler's handle() method to handle incoming http request, including extract uriPath (/search) and uriQuery. uriQuery is passed to QueryArguments static class and update this class's fields including query, numberOfResults.
3. Process QueryArguments.query, get all pages
4. Construct html output to frontend using the returned pages.

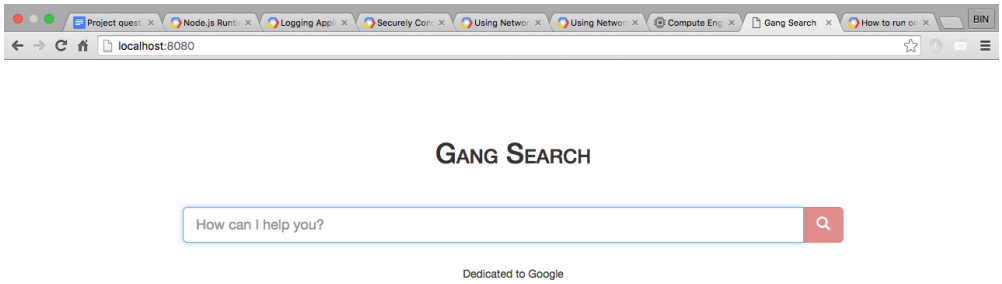
## Obstacles:

1. There are two servers fired in our implementation. One is from nodeJS which hosts web page to let user input query. Another is from back-end java code. How these two servers communicate and in what manner was a major issue
2. If user type in a query, how this query can be correctly parsed as an http request. How this request is handled by both nodeJS side and back-end java side. How response from server can be processed.

## Future work:

1. Use Google App Engine to host our entire front-end and back-end. User can visit our website and search keywords.

## Demo:



Searched results for “donald trump”