ClosetHigh Level Tools and Architecture Worksheet

NUvention Web 2014, Assigned 1/7/2014 due 1/14/2014 in class

Jordan Timmerman, Noah Conley, Hersey Liu, Carri Cowan, Samarth Zankharia

# Architecture

## Architecture Diagram & Scenario Slice

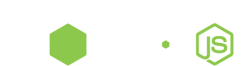
*User and items databases on*

*MongoDB*

*Server Side:*

*Node.js, python,*

*express, mongoose*



*Client Side:*

*Html5, Sass/CSS,*

*jQuery, compiled*

*jade templates*



1. Sneakerhead Bob uploads his sneakers to see what they’re worth on Closet (and how rare they are)
2. The Server submits Bob’s sneakers
3. Server adds Bob’s sneakers and sends data back to the server
4. Server estimates value based on server data
5. Bob gets a nice little estimate for the value of his rare sneakers



## 

## Source Code Location and principal owner

github.com/nuvention-web/closet.git   
responsibility falls to Jordan (username skorlir, email skorlir@gmail.com), and Noah

## Development Environment Assumptions

|  |  |
| --- | --- |
| Description of Client / User Interface Environment | HTML5, Sass, jQuery; on-the-fly compiled Jade templates served by a Node.js server. |
| Local Data Requirements | Client cookie storage or HTML5 LocalStorage possibly necessary for some aspects of session management and tracking data; in some cases could offload portions of database operation to a local IndexedDB for rapid access or even compile user data locally into a database for later batch uploading (where supported by browsers) |
| Communication between client and web server | Ajax-style communication between client and node server which responds as one would expect and serves vanilla html compiled from jade templates to update existing views with fragments, akin to one-page designs like twitter. Other pages will be served statically, using refreshes/regular requests. Thanks to node’s express structure, likely will be easy to implement RESTful API for accessing by a variety of HTTP methods providing various forms of client-side exposure and possibly even presenting the opportunity to extend functionality by user hacks. Ideally most all data will be JSON-encoded and any private information will be sent encrypted if passing ‘in-the-clear’ is necessary |
| Web server environment / language | Node, Express, Passport, and a number of other modules wrapped together to provide a holistic server environment. Likely place Sails.js or Locomotive.js on top of Express for greater functionality. Python scripts and cron jobs will likely be implemented to manage other operations (not exposed to the client in any way) |
| Data Requirements | Database requirements will almost indubitably lead us to use a Document Store, either MongoDB or CouchDB, and in some situations there could arise the feasible need for a redis or riak database to fulfill speed or replicability requirements for user interactions like searches etc. |
| 3rd Party APIs or web services used | API flow will likely include SSO implementations from Google, Facebook, OneID, Twitter, etc – also exposing us to user data including google/facebook contacts and allowing easier user instantiation and acquisition. Likely employ Wallet-esque services – Wallet, Paypal, Venmo, Bitcoin, who knows. Definitely Paypal, Wallet, and basic credit card support. |

## Technical Hypothesis

|  |  |
| --- | --- |
| Which parts of this architecture use technologies or techniques that are the biggest unknowns | *Potential unknowns:*   1. Database experience in the group is relatively minimal, it is unknown what time and resources will be necessary to learn and implement effective db management 2. Experience writing test-driven code is minimal as well, but we want to try it. it is unknown what limitations that could pose for rapid development and whether the trade-offs are worthwhile, but most sources seem to indicate “yes.” 3. Algorithms for determining the value of a user’s closet, connecting users, exposing a robust ranking system, and possibly other complex features will need to be researched, implemented on small data sets, and refined (pretty big question whether we’ll be able to mangle production quality algorithms for so many features) |
| What technical “tests” will we do insure that this can work. The technical tests for architecture are sometimes called “[spikes](http://www.jamesshore.com/Agile-Book/spike_solutions.html)” | Sample data testing of a variety of ranking/suggestion algorithms (small sets in a closed environment)to test against clear ‘correct’ associations (ie, a user with interests in shoes and electronics should be likely receive a suggestion to join a ‘sneakerheads and androids’ group)  Establish the feasibility of utilizing so many newer technologies and justify the learning curve by (1) developing the landing page with this setup to determine its speed, flexibility for coding (2) comparing its performance and ease of use with Rails  Implement a rating/rarity/value calculation for a small set of previously evaluated sneakers or handbags (easily priced items) and test the feasibility (speed) and accuracy of such systems to determine the greatest issues/bottlenecks |