*Team Project*



**Deal Manthan**

***An E-Commerce based Web Crawling using Semantics3***

Under Guidance of Prof. C. Vuppalapati

CMPE 297

Fall 2014

Wednesday (6pm – 8:45pm)

Submitted By:

Abhinaya Sampath(009336574)

Ankur Gupta (009297470)

Ayush Mittal (009311068)

Deven Pawar(009275994)

**ABSTRACT**

**Deal Manthan**

**An E-Commerce based Web Crawling using Semantics3**

**By Abhinaya Sampath, Ankur Gupta, Ayush Mittal, Deven Pawar**

Increasing attendance of retailers online and big data soaring new heights calls for a quick look at what’s trending these days in the context of the e-commerce landscape.

So the main requirements that come in context are to crawl through various websites and at the same time extract the required information out of it, where our focus is to collect product information from specific categories of interest, collecting all the product prices for a comparative analysis by the user and to make a better and informed decision on which link to explore further. Added, we give option to share your choice or a deal if you get any, across multiple domains of social media, Facebook, twitter, LinkedIn, Google plus, email sharing, to name a few.

What’s to be done with all of this collated information? Again, these are the popular trends-The simplest of all is price comparison engine that all of us are so addicted to these days. The consumer is ready to jump from one retailer on to another even for a $1 saving (besides, reduced affinities is another trend with comparison shopping). With such trends, businesses want to be involved with facilitating such information that influences consumer behavior by helping them discover lower prices for a product and have a visualized effect of his search query.

Contents

[1. Project Description 4](#_Toc405392435)

[What is a web crawler? 5](#_Toc405392436)

[What is Semantics3? 5](#_Toc405392437)

[Problem Statement 5](#_Toc405392438)

[Objectives 6](#_Toc405392439)

[Challenges 7](#_Toc405392440)

[2. System Requirements Analysis 7](#_Toc405392441)

[Functional and Non-Functional Requirements 7](#_Toc405392442)

[Functional Requirements: 7](#_Toc405392443)

[Non-Functional Requirements 9](#_Toc405392444)

[3. Web UI Requirement Principles 9](#_Toc405392445)

[User Personas: 9](#_Toc405392446)

[4. Web UI Design Principles 10](#_Toc405392447)

[Storyboard 10](#_Toc405392448)

[Wireframes 11](#_Toc405392449)

[5. High Level Architecture 14](#_Toc405392450)

[6. Component Level Design 15](#_Toc405392451)

[7. Sequence Diagram 16](#_Toc405392452)

[8. HTML 5 Features: 18](#_Toc405392453)

[MODERNIZR 18](#_Toc405392454)

[CANVAS 19](#_Toc405392455)

[CANVAS TEXT 21](#_Toc405392456)

[LOCAL STORAGE 23](#_Toc405392457)

[WEB WORKERS 24](#_Toc405392458)

[OFFLINE WEB APPLICATIONS 25](#_Toc405392459)

[INPUT TYPES 27](#_Toc405392460)

[PLACEHOLDER TEXT 29](#_Toc405392461)

[FORM AUTOFOCUS 30](#_Toc405392462)

[MICRODATA 32](#_Toc405392463)

[9. Interfaces: RESTful and Server Side Design 33](#_Toc405392464)

[10. Client-Side Design 35](#_Toc405392465)

[11. Testing 39](#_Toc405392466)

[12. Automation Testing 42](#_Toc405392467)

[13. Cross Browser Compatibility 44](#_Toc405392468)

[14. JavaScript Libraries 44](#_Toc405392469)

[15. Design Pattern 45](#_Toc405392470)

[16. Localization 48](#_Toc405392471)

[17. Profiling 51](#_Toc405392472)

[Spy-js 51](#_Toc405392473)

[18. SEO 52](#_Toc405392474)

[19. Profiling 53](#_Toc405392475)

[20. Individual Contribution 54](#_Toc405392476)

# Project Description

## What is a web crawler?

Web crawler (also known as a Web spider or Web robot) is a program or automated script which browses the World Wide Web in a methodical and automated manner.

This process is called Web crawling or spidering. Many legitimate sites, in particular search engines, use spidering as a means of providing up-to-date data. Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine, which will index the downloaded pages to provide fast searches. Crawlers can also be used for automating maintenance tasks on a Web site, such as checking links or validating HTML codes. Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting e-mail addresses (usually for spam).

A Web crawler is one type of bot, or software agent. In general, it starts with a list of URLs to visit, called the seeds. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl frontier.

## What is Semantics3?

The Semantics3 Products API gives you comprehensive access to data across millions of products and prices. With this API, you can build sophisticated models to track prices across a wide variety of products, easily build shopping comparison engines, enrich your e-commerce stores with rich metadata.

## Problem Statement

Requirements can be categorized into 3:

Collecting product information from specific categories of interest.

Collecting product prices from all categories on a retailer site.

Option to share your individual search on social media

The idea behind collecting the entire product catalog is to run analyses on rich product attributes. For instance, how was a black IPad case priced over the blue one; or how fast are the i7 processors performing over i5 in the demand vs. supply chain. This technique is turning social too by providing assistance to buyers in order to make informed decisions.

## Objectives

* To provide comparative analysis between different products.
* To have used best features of front-end.
* To implement user-friendly front-end experience.
* To make the application fast and easy to use.
* To make a responsive and dynamic web application.
* To deploy the application on cloud for better scalability.
* To provide visualization of user’s search
* To add social media sharing on each search query.

## Challenges

Initially we were interested to all price comparisons by accessing every e-commerce website individually. Even manual gathering of data from various web sources but the results were limited and efforts were high. Even with the manual effort, structuring of the data in order to import it into their database was a challenge. We wanted to run the comparison engine and perform other monitoring activities, but getting data from highly protected websites with differently structured data was a big challenge. Then we came across Semantic3 API, which helps to resolve the problem to some extent, but even getting deals from the leaders in market was coming at a cost.

# 2. System Requirements Analysis

The system is designed to use Semantic3 api and gives comparative results to the user.

## Functional and Non-Functional Requirements

### Functional Requirements:

* User uses the system to get recommendation on the cuisine and food best suited to his health profile.
* User can update his health statistics to get recommendations based on the updated data.
* User can view similar items and users.
* Can view history of recommendations made.
* Makes the user aware of his calorie intake.

**Client**

Providing interface to the user for interaction with the web crawling using Semantic3 API.

1. Login and Signup: The application will prompt the user to enter his email id and password to register and get a personalized recommendation of deals on sign up.

The user selects username and password to login to the system.

2. Home: The home will provide option to users to query the engine and see results with more personalized deals and suggestions if login, else generalized view.

3. Comparative Analysis: To have a comparative analysis of products searched.

**Server:**

The server does the job of providing results based on the search query provided by the user. Its requirements are:

1. Handle Requests: System must be able to receive requests from users which include their choice of query for e-commerce appliance.

2. Provide Appropriate results to the user using Semantics API via rest calls.

3. Dynamic System: The application should be dynamic and should provide results on updated information.

### Non-Functional Requirements

1. Speed: The system should be efficient in terms of speed. Fast and effective techniques to be used to compute results.

2. Dynamic: System should be able to dynamically update itself according to new information.

3. Scalable: Deployed on cloud for better scalability of the application.

4. Efficient: System should be efficient which means it should be able to give accurate results without compromising on the speed

# 3. Web UI Requirement Principles

## User Personas:

**Angel: Market Analyst**

I am an Analyst. My job is to do market study and I am responsible for products price analysis. I need a website where I can see the price flow for different products across different website.

**Hall: Software Engineer**

I am a software engineer. I like working on new software’s. I always need to buy new software’s every month. I don’t want to put my extra money in buying them. Probably, I need a website that could tell me where are my software cheapest?

**Thomas: Graduate Student**

I am a graduate student. I do research on projects. My focus is to learn and develop innovative products. I need a website where I can buy my stuff so that I do not put in my extra money and save it for my studies.

**James: Customer service center**

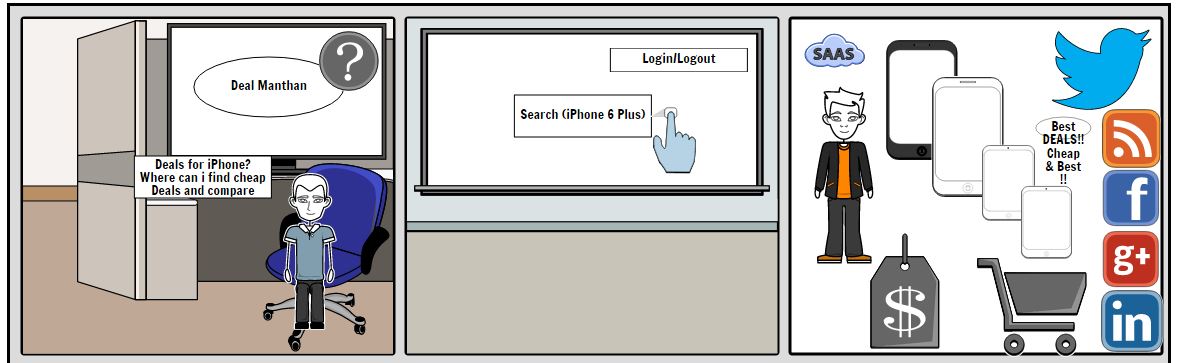
I provide assistance to my customers 24X7. I solve my customer’s issues. Sometimes, I need to order new product as a replacement to the products not working. I would need to go to different sites to check where I can get the economic priced products.

**Chris: Small MidCap Business**

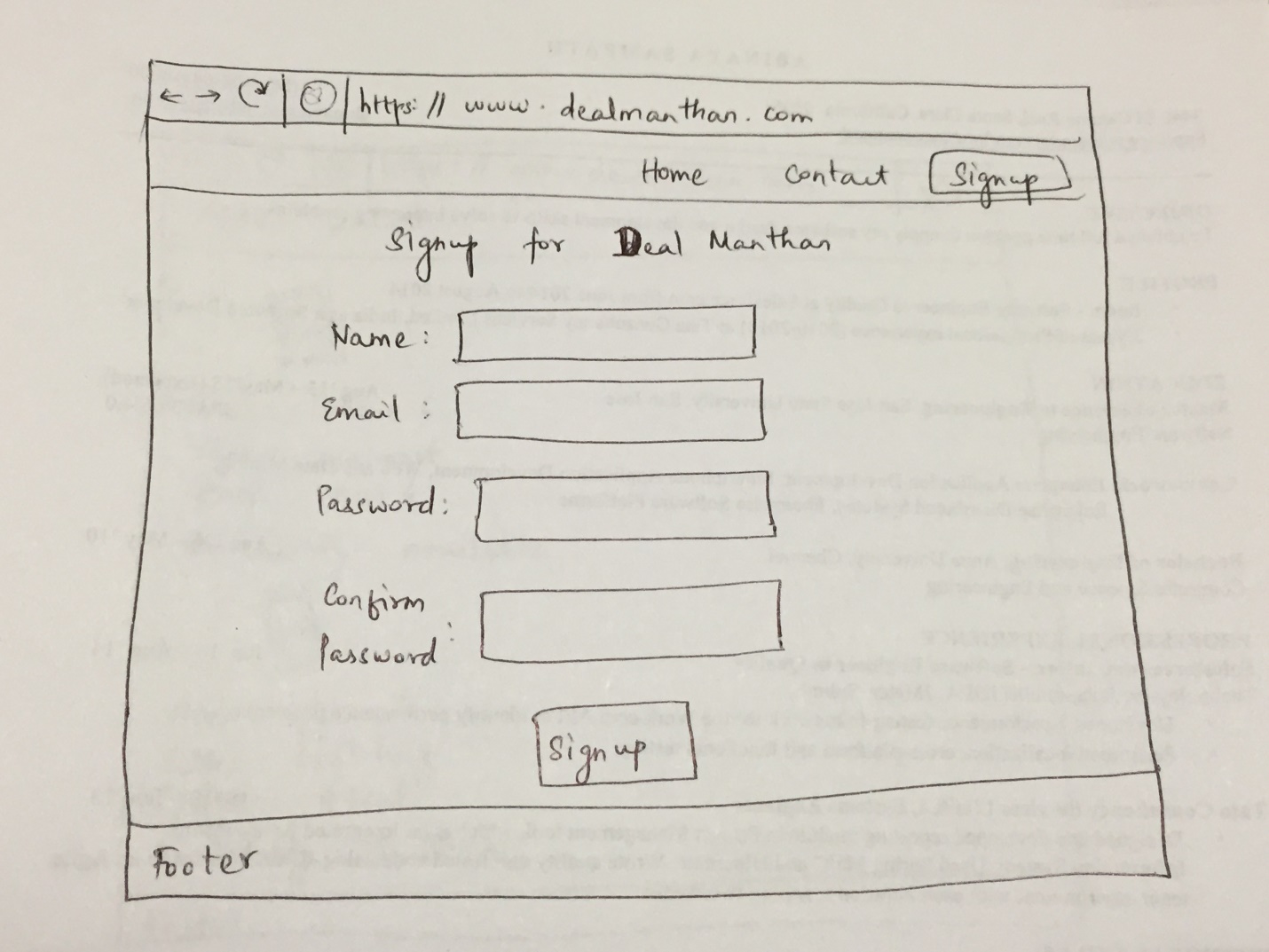
I look out for best deals for my customers so that they get the right product with low price. I always go to different sites and check the best deals out there.

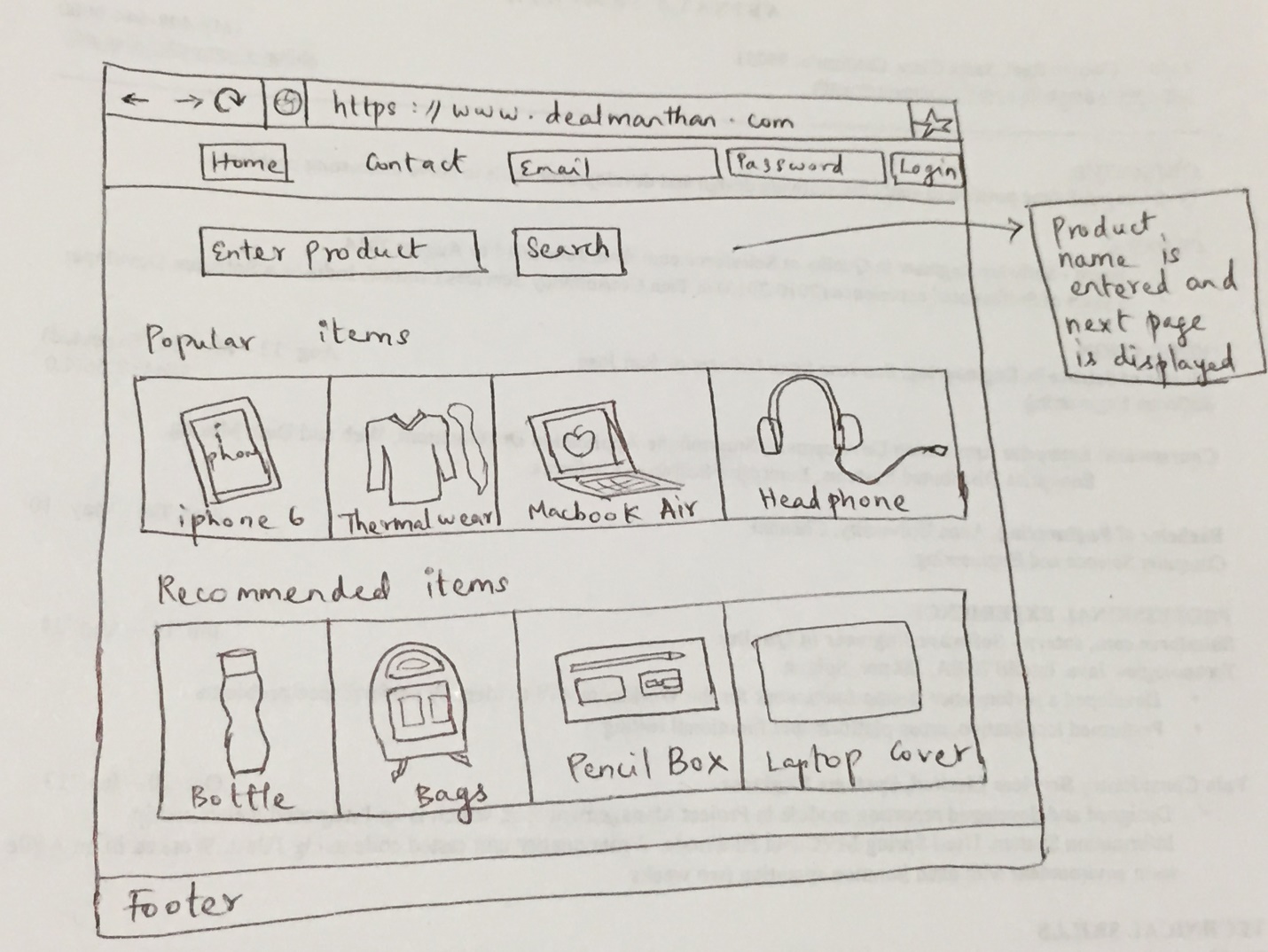
# 4. Web UI Design Principles

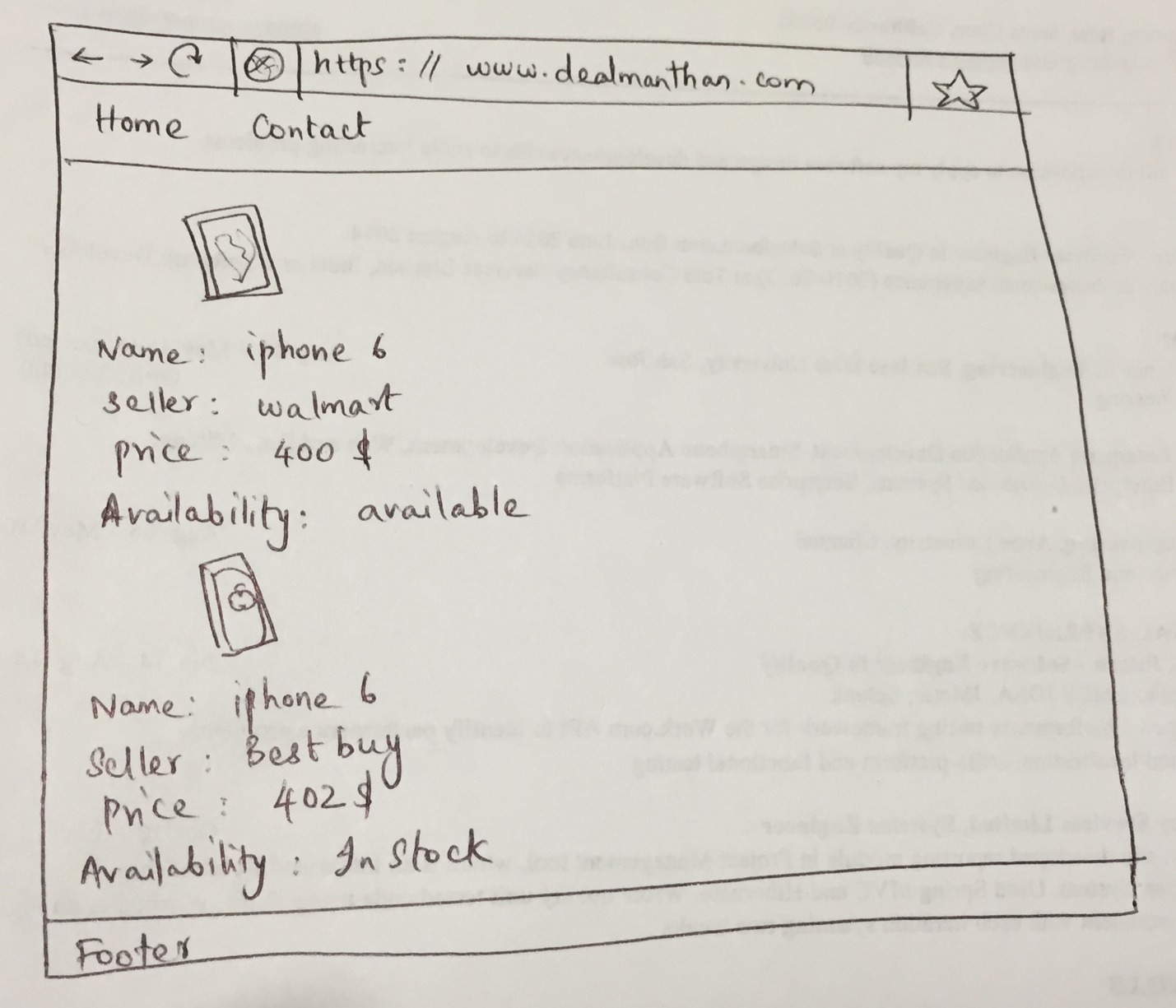
## Storyboard



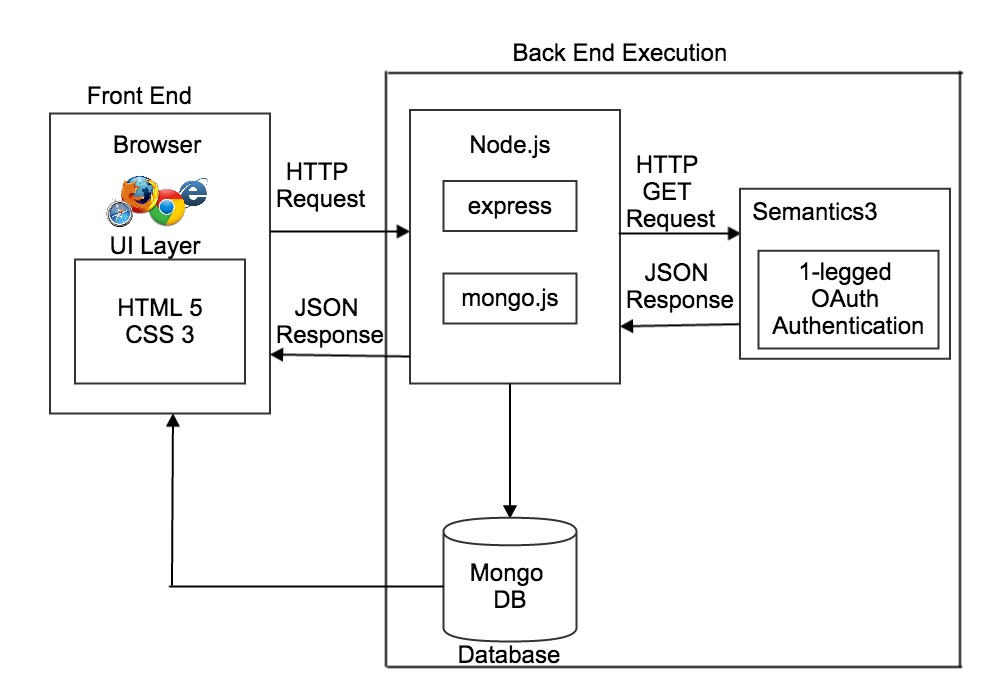
## Wireframes







# 5. High Level Architecture



# 6. Component Level Design

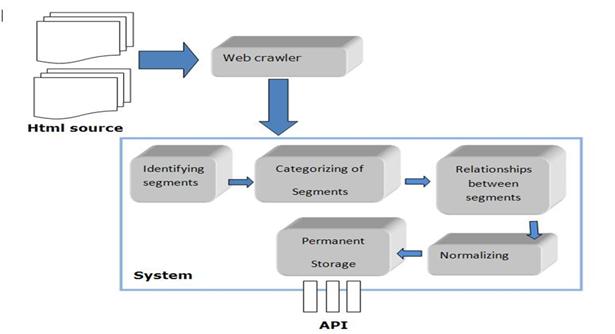
* A component is a basic building block for the computer software.
* It is a higher level abstractions defined by their interfaces.
* It helps in achieving the objectives & requirements of system to be built.

Characteristics of Components:

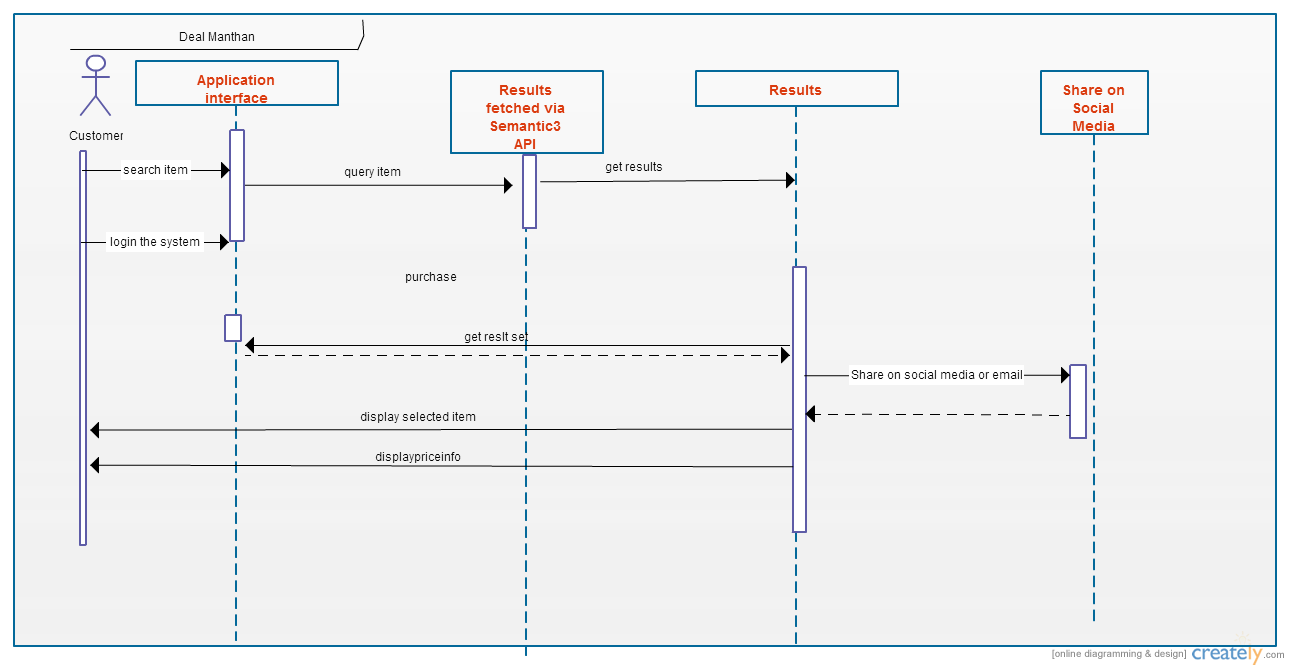
* Standardized
* Independent
* Compostable
* Deployable
* Documented

A software component is a modular building block for computer software. It is a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces. A component communicates and collaborates with other components and entities outside the boundaries of the system.

* Three different views of a component
  + An object-oriented view
  + A conventional view
  + A process-related view



# 7. Sequence Diagram



# 8. HTML 5 Features:

## MODERNIZR

[Modernizr](http://www.modernizr.com/) is an open source, MIT-licensed JavaScript library that detects support for many HTML5 & CSS3 features. You should always use the latest version. To use it, include the following <script> element at the top of your page.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Dive Into HTML5</title>

<script src="modernizr.min.js"></script>

</head>

<body>

...

</body>

</html>

Modernizr runs automatically. There is no modernizr\_init() function to call. When it runs, it creates a global object calledModernizr, that contains a set of Boolean properties for each feature it can detect. For example, if your browser supports the[canvas API](http://diveintohtml5.info/canvas.html), the Modernizr.canvas property will be true. If your browser does not support the canvas API, theModernizr.canvas property will be false.

if (Modernizr.canvas) {

// let's draw some shapes!

} else {

// no native canvas support available :(

}

## CANVAS

HTML5 defines [the <canvas> element](http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html) as “a resolution-dependent bitmap canvas that can be used for rendering graphs, Product graphics, or other visual images on the fly.” A canvas is a rectangle in your page where you can use JavaScript to draw anything you want. HTML5 defines a set of functions (“the canvas API”) for drawing shapes, defining paths, creating gradients, and applying transformations.

Checking for the canvas API uses [detection technique #2](http://diveintohtml5.info/detect.html#techniques). If your browser supports the canvas API, the DOM object it creates to represent a <canvas> element will have a [getContext() method](http://diveintohtml5.info/canvas.html#shapes). If your browser doesn’t support the canvas API, theDOM object it creates for a <canvas> element will only have the set of common properties, but not anything canvas-specific.

function supports\_canvas() {

return !!document.createElement('canvas').getContext;

}

This function starts by creating a dummy <canvas> element. But the element is never attached to your page, so no one will ever see it. It’s just floating in memory, going nowhere and doing nothing, like a canoe on a lazy river.

return !!document.createElement('canvas').getContext;

As soon as you create the dummy <canvas> element, you test for the presence of a getContext() method. This method will only exist if your browser supports the canvas API.

return !!document.createElement('canvas').getContext;

Finally, you use the double-negative trick to force the result to a Boolean value (true or false).

return !!document.createElement('canvas').getContext;

This function will detect support for most of the canvas API, including [shapes](http://diveintohtml5.info/canvas.html#shapes), [paths](http://diveintohtml5.info/canvas.html#paths), [gradients & patterns](http://diveintohtml5.info/canvas.html#gradients). It will not detect the third-party [explorercanvas library](http://diveintohtml5.info/canvas.html#ie) that implements the canvas API in Microsoft Internet Explorer.

Instead of writing this function yourself, you can use [Modernizr](http://diveintohtml5.info/detect.html#modernizr) to detect support for the canvas API.

if (Modernizr.canvas) {

// let's draw some shapes!

} else {

// no native canvas support available :(

}

There is a separate test for the canvas text API, which I will demonstrate next.

## CANVAS TEXT

Even if your browser supports the [canvas API](http://diveintohtml5.info/detect.html#canvas), it might not support the canvas text API. The canvas API grew over time, and the text functions were added late in the Product. Some browsers shipped with canvas support before the text API was complete.

Checking for the canvas text API uses [detection technique #2](http://diveintohtml5.info/detect.html#techniques). If your browser supports the canvas API, the DOM object it creates to represent a <canvas> element will have the [getContext() method](http://diveintohtml5.info/canvas.html#shapes). If your browser doesn’t support the canvas API, the DOM object it creates for a <canvas> element will only have the set of common properties, but not anything canvas-specific.

function supports\_canvas\_text() {

if (!supports\_canvas()) { return false; }

var dummy\_canvas = document.createElement('canvas');

var context = dummy\_canvas.getContext('2d');

return typeof context.fillText == 'function';

}

The function starts by [checking for canvas support](http://diveintohtml5.info/detect.html#canvas), using the supports\_canvas() function you just saw in the previous section. If your browser doesn’t support the canvas API, it certainly won’t support the canvas text API!

if (!supports\_canvas()) { return false; }

Next, you create a dummy <canvas> element and get its drawing context. This is guaranteed to work, because thesupports\_canvas() function already checked that the getContext() method exists on all canvas objects.

var dummy\_canvas = document.createElement('canvas');

var context = dummy\_canvas.getContext('2d');

Finally, you check whether the drawing context has a fillText() function. If it does, the canvas text API is available. Hooray!

return typeof context.fillText == 'function';

Instead of writing this function yourself, you can use [Modernizr](http://diveintohtml5.info/detect.html#modernizr) to detect support for the canvas text API.

if (Modernizr.canvastext) {

// let's draw some text!

} else {

// no native canvas text support available :(

}

## LOCAL STORAGE

[HTML5 storage](http://dev.w3.org/html5/webstorage/) provides a way for web sites to store information on your computer and retrieve it later. The concept is similar to cookies, but it’s designed for larger quantities of information. Cookies are limited in size, and your browser sends them back to the web server every time it requests a new page (which takes extra time and precious bandwidth). HTML5 storage stays on your computer, and web sites can access it with JavaScript after the page is loaded.

Q: Is local storage really part of HTML5? Why is it in a separate specification?  
A: The short answer is yes, local storage is part of HTML5. The slightly longer answer is that local storage used to be part of the main HTML5 specification, but it was split out into a separate specification because some people in theHTML5 Working Group complained that HTML5 was too big. If that sounds like slicing a pie into more pieces to reduce the total number of calories… well, welcome to the wacky world of standards.

Checking for HTML5 storage support uses [detection technique #1](http://diveintohtml5.info/detect.html#techniques). If your browser supports HTML5 storage, there will be alocalStorage property on the global window object. If your browser doesn’t support HTML5 storage, the localStorageproperty will be undefined. Due to an unfortunate bug in older versions of Firefox, this test will raise an exception if cookies are disabled, so the entire test is wrapped in a try..catch statement.

function supports\_local\_storage() {

try {

return 'localStorage' in window && window['localStorage'] !== null;

} catch(e){

return false;

}

}

Instead of writing this function yourself, you can use [Modernizr](http://diveintohtml5.info/detect.html#modernizr) (1.1 or later) to detect support for HTML5 local storage.

if (Modernizr.localstorage) {

// window.localStorage is available!

} else {

// no native support for local storage :(

// try a fallback or another third-party solution

}

## WEB WORKERS

[Web Workers](http://www.whatwg.org/specs/web-workers/current-work/) provide a standard way for browsers to run JavaScript in the background. With web workers, you can spawn multiple “threads” that all run at the same time, more or less. (Think of how your computer can run multiple applications at the same time, and you’re most of the way there.) These “background threads” can do complex mathematical calculations, make network requests, or access [local storage](http://diveintohtml5.info/detect.html#local-storage) while the main web page responds to the user scrolling, clicking, or typing.

Checking for web workers uses [detection technique #1](http://diveintohtml5.info/detect.html#techniques). If your browser supports the Web Worker API, there will be a Workerproperty on the global window object. If your browser doesn’t support the Web Worker API, the Worker property will be undefined.

function supports\_web\_workers() {

return !!window.Worker;

}

Instead of writing this function yourself, you can use [Modernizr](http://diveintohtml5.info/detect.html#modernizr) (1.1 or later) to detect support for web workers.

if (Modernizr.webworkers) {

// window.Worker is available!

} else {

// no native support for web workers :(

// try a fallback or another third-party solution

}

Note that JavaScript is case-sensitive. The Modernizr attribute is called webworkers (all lowercase), but the DOM object is called window.Worker (with a capital “W” in “Worker”).

## OFFLINE WEB APPLICATIONS

Reading static web pages offline is easy: connect to the Internet, load a web page, disconnect from the Internet, drive to a secluded cabin, and read the web page at your leisure. (To save time, you may wish to skip the step about the cabin.) But what about web applications like [Gmail](http://mail.google.com/) or [Google Docs](http://docs.google.com/)? Thanks to HTML5, anyone (not just Google!) can build a web application that works offline.

[Offline web applications](http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html#offline) start out as online web applications. The first time you visit an offline-enabled web site, the web server tells your browser which files it needs in order to work offline. These files can be anything — HTML, JavaScript, images, even [videos](http://diveintohtml5.info/detect.html#video). Once your browser downloads all the necessary files, you can revisit the web site even if you’re not connected to the Internet. Your browser will notice that you’re offline and use the files it has already downloaded. When you get back online, any changes you’ve made can be uploaded to the remote web server.

Checking for offline support uses [detection technique #1](http://diveintohtml5.info/detect.html#techniques). If your browser supports offline web applications, there will be anapplicationCache property on the global window object. If your browser doesn’t support offline web applications, theapplicationCache property will be undefined. You can check for offline support with the following function:

function supports\_offline() {

return !!window.applicationCache;

}

Instead of writing this function yourself, you can use [Modernizr](http://diveintohtml5.info/detect.html#modernizr) (1.1 or later) to detect support for offline web applications.

if (Modernizr.applicationcache) {

// window.applicationCache is available!

} else {

// no native support for offline :(

// try a fallback or another third-party solution

}

Note that JavaScript is case-sensitive. The Modernizr attribute is called applicationcache (all lowercase), but the DOM object is called window.applicationCache (mixed case).

## INPUT TYPES

You know all about web forms, right? Make a <form>, add a few <input type="text">elements and maybe an <input type="password">, and finish it off with an <input type="submit"> button.

You don’t know the half of it. HTML5 defines over a dozen new input types that you can use in your forms.

1. [<input type="search">](http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#text-state-and-search-state) for search boxes
2. [<input type="number">](http://www.whatwg.org/specs/web-apps/current-work/multipage/number-state.html#number-state) for spinboxes
3. [<input type="range">](http://www.whatwg.org/specs/web-apps/current-work/multipage/number-state.html#range-state) for sliders
4. [<input type="color">](http://www.whatwg.org/specs/web-apps/current-work/multipage/number-state.html#color-state) for color pickers
5. [<input type="tel">](http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#telephone-state) for telephone numbers
6. [<input type="url">](http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#url-state) for web addresses
7. [<input type="email">](http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#e-mail-state) for email addresses
8. [<input type="date">](http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#date-state) for calendar date pickers
9. [<input type="month">](http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#month-state) for months
10. [<input type="week">](http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#week-state) for weeks
11. [<input type="time">](http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#time-state) for timestamps
12. [<input type="datetime">](http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#date-and-time-state) for precise, absolute date+time stamps
13. [<input type="datetime-local">](http://www.whatwg.org/specs/web-apps/current-work/multipage/states-of-the-type-attribute.html#local-date-and-time-state) for local dates and times

Checking for HTML5 input types uses [detection technique #4](http://diveintohtml5.info/detect.html#techniques). First, you create a dummy <input> element in memory. The default input type for all <input> elements is "text". This will prove to be vitally important.

var i = document.createElement("input");

Next, set the type attribute on the dummy <input> element to the input type you want to detect.

i.setAttribute("type", "color");

If your browser supports that particular input type, the type property will retain the value you set. If your browser doesn’t support that particular input type, it will ignore the value you set and the type property will still be "text".

return i.type !== "text";

Instead of writing 13 separate functions yourself, you can use [Modernizr](http://diveintohtml5.info/detect.html#modernizr) to detect support for all the new input types defined in HTML5. Modernizr reuses a single <input> element to efficiently detect support for all 13 input types. Then it builds a hash called Modernizr.inputtypes, that contains 13 keys (the HTML5 type attributes) and 13 Boolean values (true if supported, false if not).

if (!Modernizr.inputtypes.date) {

// no native support for <input type="date"> :(

// maybe build one yourself with [Dojo](http://docs.dojocampus.org/dojox/widget/Calendar) or [jQueryUI](http://jqueryui.com/demos/datepicker/)

}

## PLACEHOLDER TEXT

Top of Form

Bottom of Form

Besides [new input types](http://diveintohtml5.info/detect.html#input-types), HTML5 includes several small tweaks to existing forms. One improvement is the ability to set [placeholder text in an input field](http://www.whatwg.org/specs/web-apps/current-work/multipage/common-input-element-attributes.html#the-placeholder-attribute). Placeholder text is displayed inside the input field as long as the field is empty and not focused. As soon you click on (or tab to) the input field, the placeholder text disappears. The [chapter on web forms](http://diveintohtml5.info/forms.html#placeholder) has screenshots if you’re having trouble visualizing it.

Checking for placeholder support uses [detection technique #2](http://diveintohtml5.info/detect.html#techniques). If your browser supports placeholder text in input fields, the DOM object it creates to represent an <input>element will have a placeholder property (even if you don’t include a placeholderattribute in your HTML). If your browser doesn’t support placeholder text, the DOMobject it creates for an <input> element will not have a placeholder property.

function supports\_input\_placeholder() {

var i = document.createElement('input');

return 'placeholder' in i;

}

Instead of writing this function yourself, you can use [Modernizr](http://diveintohtml5.info/detect.html#modernizr) (1.1 or later) to detect support for placeholder text.

if (Modernizr.input.placeholder) {

// your placeholder text should already be visible!

} else {

// no placeholder support :(

// fall back to a scripted solution

}

## FORM AUTOFOCUS

Web sites can use JavaScript to focus the first input field of a web form automatically. For example, the home page of [Google.com](http://www.google.com/) will autofocus the input box so you can type your search keywords without having to position the cursor in the search box. While this is convenient for most people, it can be annoying for power users or people with special needs. If you press the space bar expecting to scroll the page, the page will not scroll because the focus is already in a form input field. (It types a space in the field instead of scrolling.) If you focus a different input field while the page is still loading, the site’s autofocus script may “helpfully” move the focus back to the original input field upon completion, disrupting your flow and causing you to type in the wrong place.

Because the autofocusing is done with JavaScript, it can be tricky to handle all of these edge cases, and there is little recourse for people who don’t want a web page to “steal” the focus.

To solve this problem, HTML5 introduces [an autofocus attribute on all web form controls](http://www.whatwg.org/specs/web-apps/current-work/multipage/association-of-controls-and-forms.html#autofocusing-a-form-control). The autofocus attribute does exactly what it says on the tin: it moves the focus to a particular input field. But because it’s just markup instead of a script, the behavior will be consistent across all web sites. Also, browser vendors (or extension authors) can offer users a way to disable the autofocusing behavior.

Checking for autofocus support uses [detection technique #2](http://diveintohtml5.info/detect.html#techniques). If your browser supports autofocusing web form controls, theDOM object it creates to represent an <input> element will have an autofocus property (even if you don’t include theautofocus attribute in your HTML). If your browser doesn’t support autofocusing web form controls, the DOM object it creates for an <input> element will not have an autofocus property. You can detect autofocus support with this function:

function supports\_input\_autofocus() {

var i = document.createElement('input');

return 'autofocus' in i;

}

Instead of writing this function yourself, you can use [Modernizr](http://diveintohtml5.info/detect.html#modernizr) (1.1 or later) to detect support for autofocused form fields.

if (Modernizr.input.autofocus) {

// autofocus works!

} else {

// no autofocus support :(

// fall back to a scripted solution

}

## MICRODATA

[Microdata](http://www.whatwg.org/specs/web-apps/current-work/multipage/links.html#microdata) is a standardized way to provide additional semantics in your web pages. For example, you can use microdata to declare that a photograph is available under a specific Creative Commons license. As you’ll see in [the distributed extensibility chapter](http://diveintohtml5.info/extensibility.html), you can use microdata to mark up an “About Me” page. Browsers, browser extensions, and search engines can convert yourHTML5 microdata markup into a [vCard](http://en.wikipedia.org/wiki/VCard), a standard format for sharing contact information. You can also define your own microdata vocabularies.

The HTML5 microdata standard includes both HTML markup (primarily for search engines) and a set of DOM functions (primarily for browsers). There’s no harm in including microdata markup in your web pages. It’s nothing more than a few well-placed attributes, and search engines that don’t understand the microdata attributes will just ignore them. But if you need to access or manipulate microdata through the DOM, you’ll need to check whether the browser supports the microdata DOM API.

Checking for HTML5 microdata API support uses [detection technique #1](http://diveintohtml5.info/detect.html#techniques). If your browser supports the HTML5 microdataAPI, there will be a getItems() function on the global document object. If your browser doesn’t support microdata, thegetItems() function will be undefined.

function supports\_microdata\_api() {

return !!document.getItems;

}

Modernizr does not yet support checking for the microdata API, so you’ll need to use the function like the one listed above.

# 9. Interfaces: RESTful and Server Side Design

Rest based web application are simple, light weight and fast. We have used restful web service as it is built to work best on the web. Rest is an architectural style in designing a web based application that specifies constraints applied to web service such as scalability, extensibility, modifiability, etc.

We have used Semantics s3 Products Data API as a rest service to provide data relating to products details along with the current price in the market. It gives a comprehensive access to data across millions of products. It uses Oauth as an authentication. The Access Token URL used is <https://www.semantics3.com>.

The request headers are Consumer Key, Consumer Secret, Access Token and Access Secret. For example, Get method used for requesting the product details is: <https://name-semantics3-products-data.p.mashape.com/products>.

**Service side:**

We have used Express as a MVC web framework. It is minimal and flexible and provides a robust web application. It is good for providing end to end high performance.

We used NOSQL as our database. We have used Mongo DB which executes over the express.

*<<write about mongodb>>*

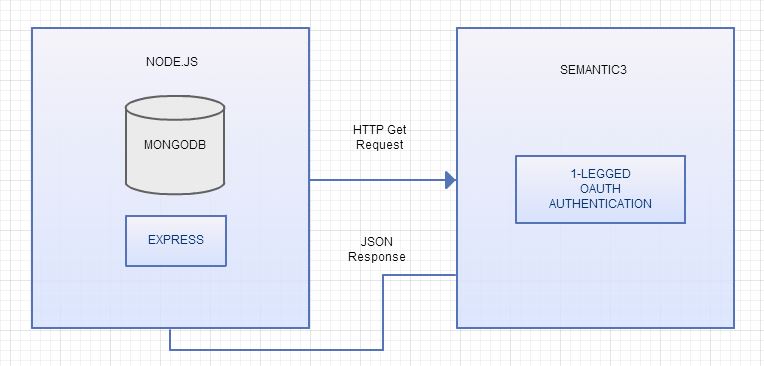
*<<write about mvc>>*

Semantics3 API allows us to access the data about many different products and prices. This API is useful to compare prices from various websites.

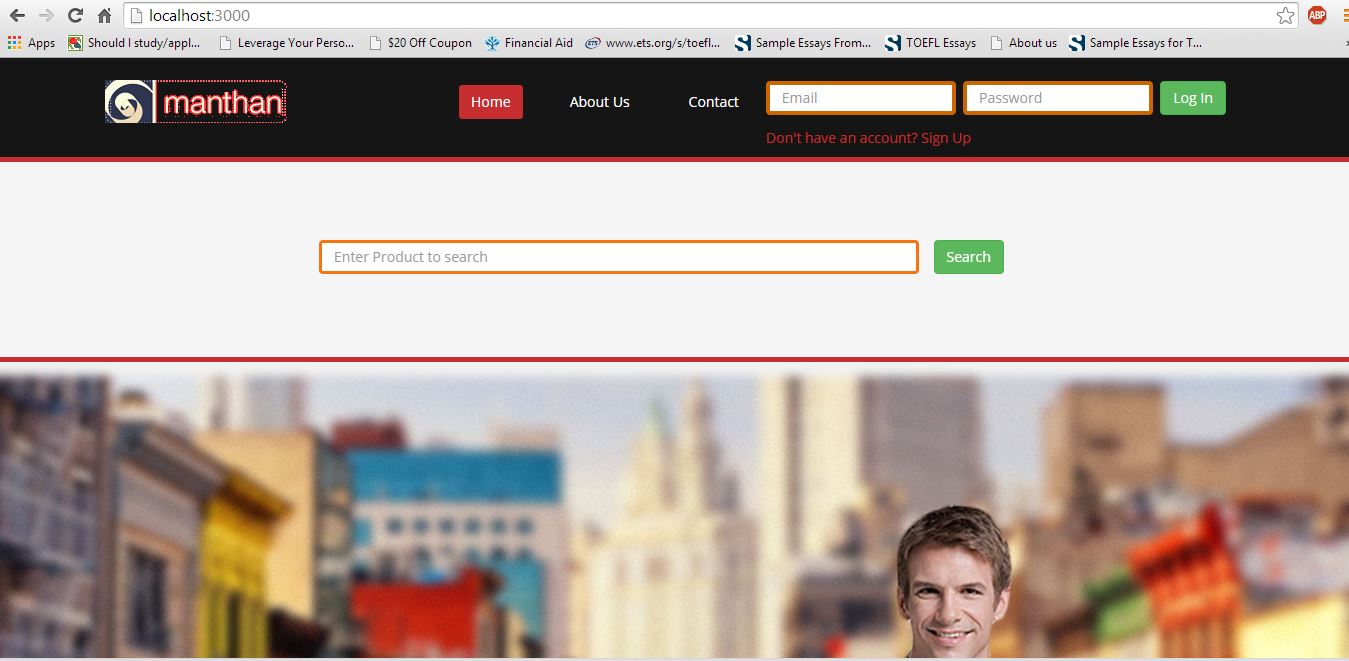
The API is RESTful. The call to the semantics3 API is made as HTTP GET requests in the production environment. The responses and the error messages are in the form of JSON.

The details about the product such as the price, seller, images, etc are retrieved based on the search parameter. This is a like a free text search that we do in a search engine. The product name is given as the search parameter and the JSON response is retrieved. The received response is a nested JSON, which needs parsing at different levels. The JSON response is parsed and the items are sorted based on the price.

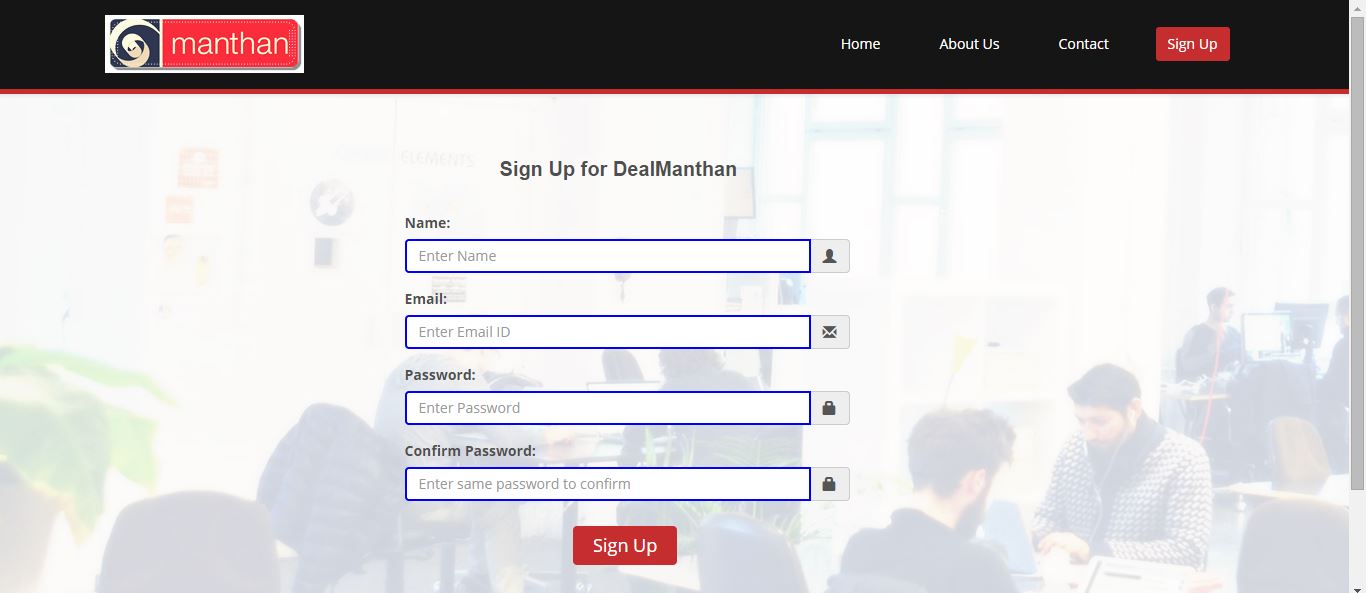
There is a lot of information in the JSON response but the most important aspects such as price, seller information, availability, images and the url are retrieved from the response. The information retrieved is then sorted to find out the best price a seller offers. A single product has more than one seller and the prices are sorted for each product separately. So, the sorting is done at product level and for the overall results.

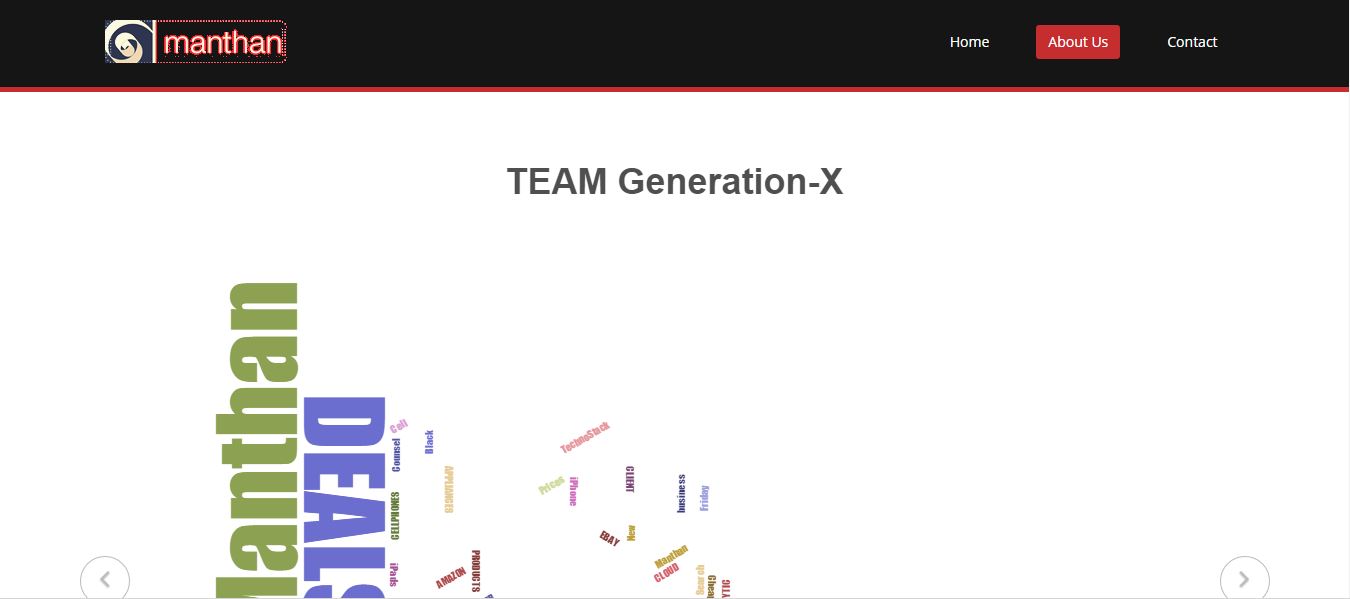


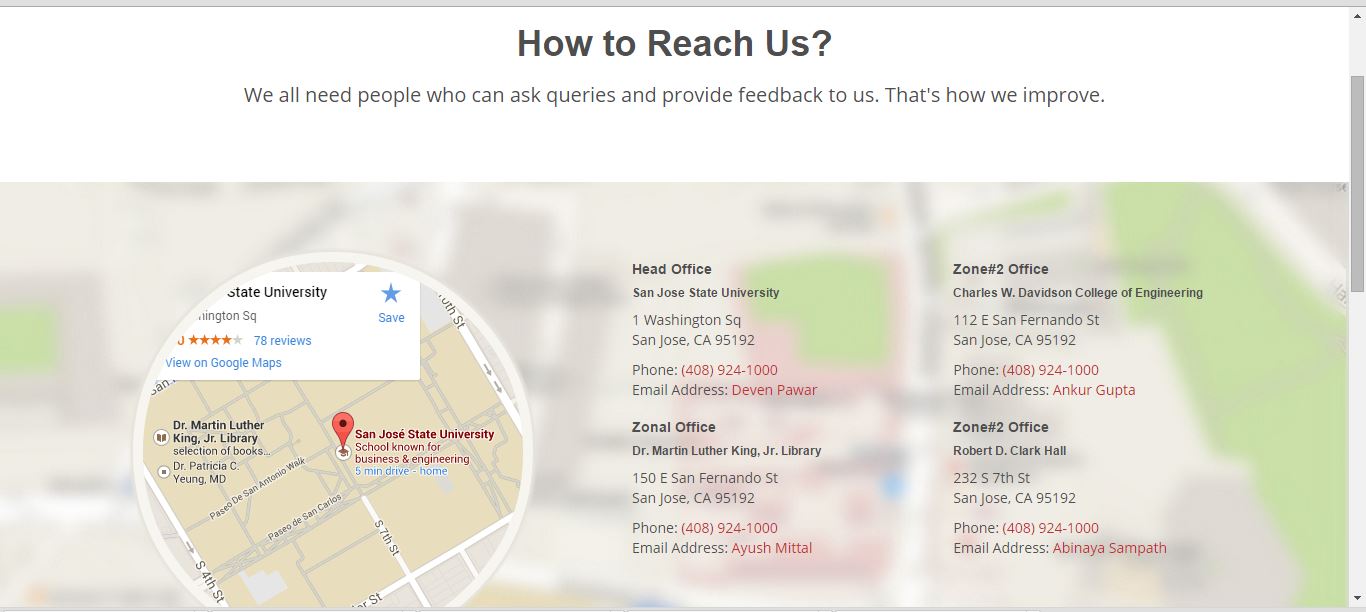
# 10. Client-Side Design

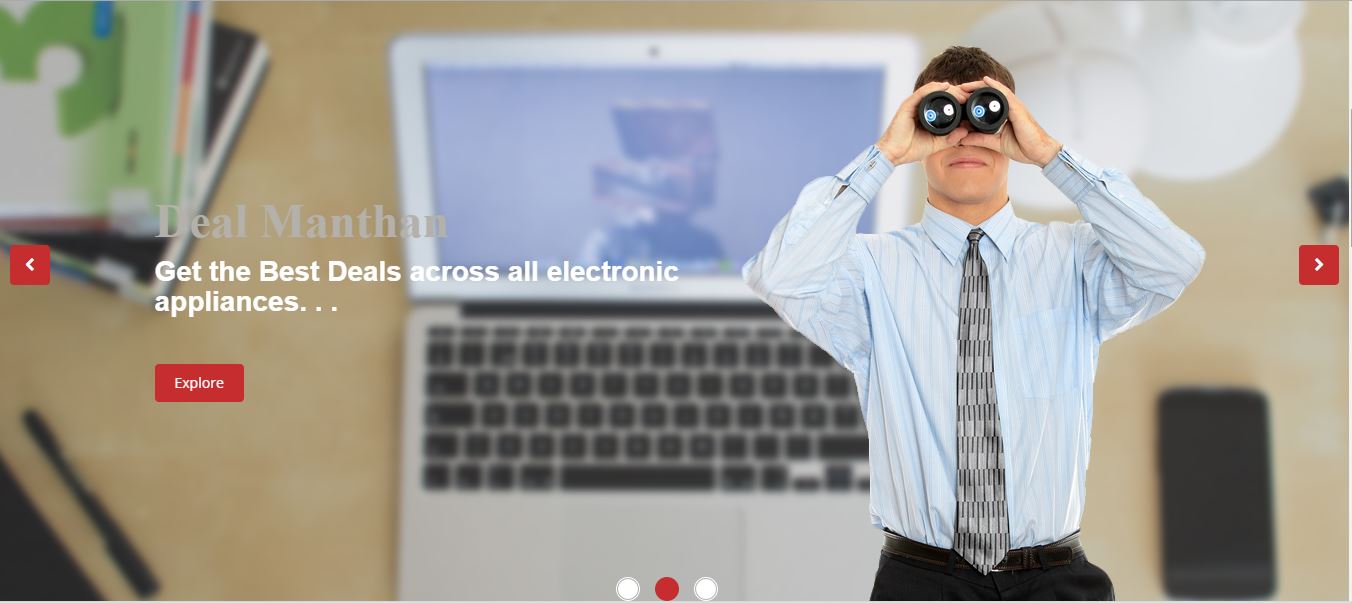


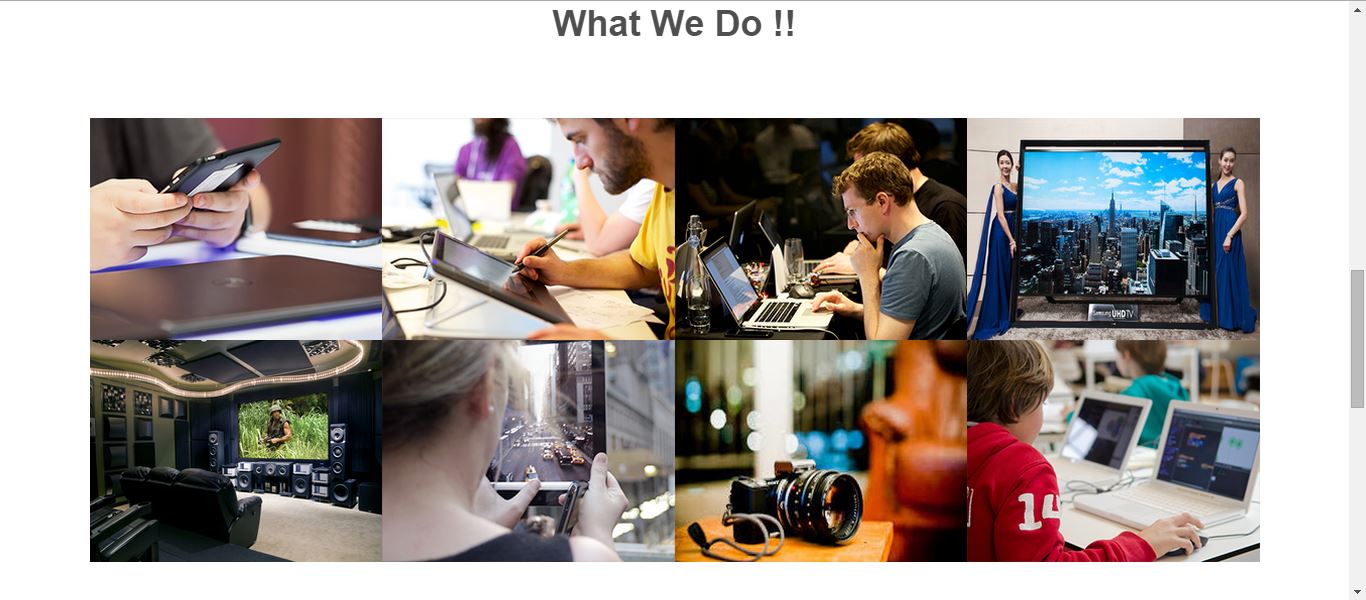


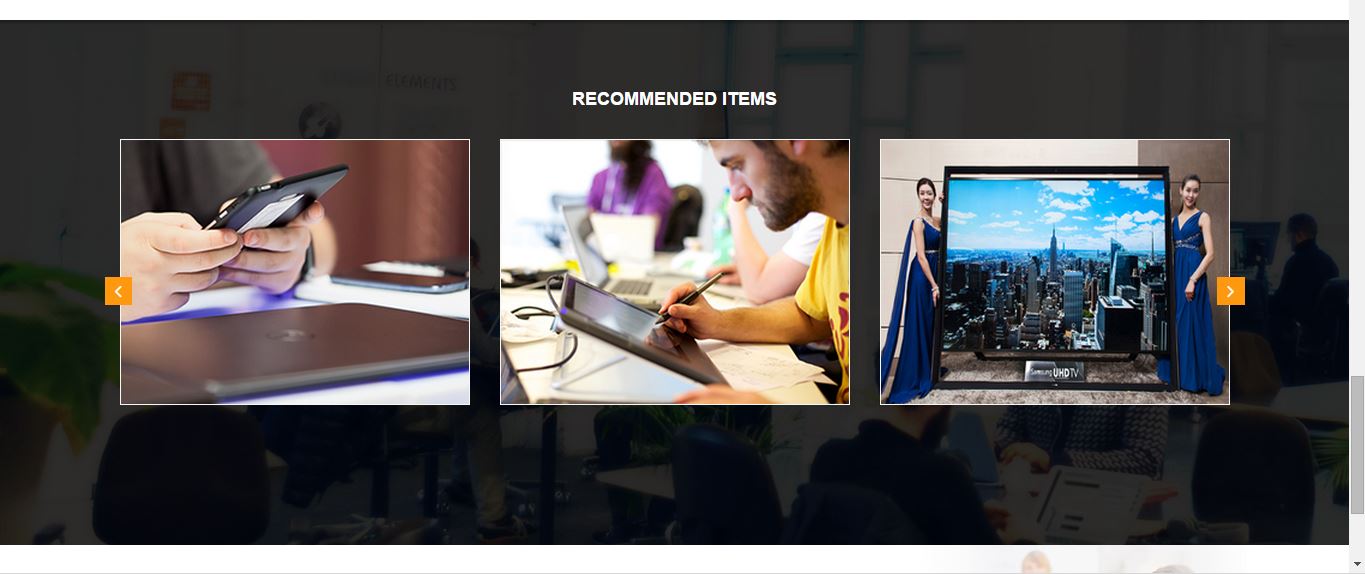












# 11. Testing

**What is User Interface Testing?**

User interface testing, a testing technique used to identify the presence of defects is a product/software under test by using Graphical user interface [GUI].

**GUI Testing - Characteristics:**

* GUI is a hierarchical, graphical front end to the application, contains graphical objects with a set of properties.
* During execution, the values of the properties of each objects of a GUI define the GUI state.
* It has capabilities to exercise GUI events like key press/mouse click.
* Able to provide inputs to the GUI Objects.
* To check the GUI representations to see if they are consistent with the expected ones.
* It strongly depends on the used technology.

**GUI Testing - Approaches:**

* **Manual Based -**Based on the domain and application knowledge of the tester.
* **Capture and Replay -**Based on capture and replay of user actions.
* **Model-based testing -**Based on the execution of user sessions based on a GUI model. Various GUI models are briefly discussed below.

**Model Based Testing - In Brief:**

* **Event-based model -**Based on all events of the GUI need to be executed at least once.
* **State-based model -**"all states" of the GUI are to be exercised at least once.
* **Domain model -**Based on the application domain and its functionality.

**GUI Testing Checklist:**

* Check Screen Validations
* Verify All Navigations
* Check usability Conditions
* Verify Data Integrity
* Verify the object states
* Verify the date Field and Numeric Field Formats

**Steps for User Interface Testing:**

1) Check with the spellings for all the labels

2) Check whether the Tab sequence is working properly in all the fields

3) Anyone option have to be selected as default in the Radio box

4) Able to select the options in the radio box by using right and left arrows in the keyboard

5) Select option has to display as default in the combo box

6) It will not allow entering a new option in the combo box

7) We can able to select the options in the list by using up and down arrows in the keyboard

8) Symbols have to display in all the mandatory fields

9) By clicking submit button without entering any texts, it have to display error message in mandatory fields

10) By Clicking Cancel button it has to display the previous or home page

11) Font size should be proper as per the requirement

12) Font style should be proper as per the requirement

13) Spaces between the texts and the fields should be proper

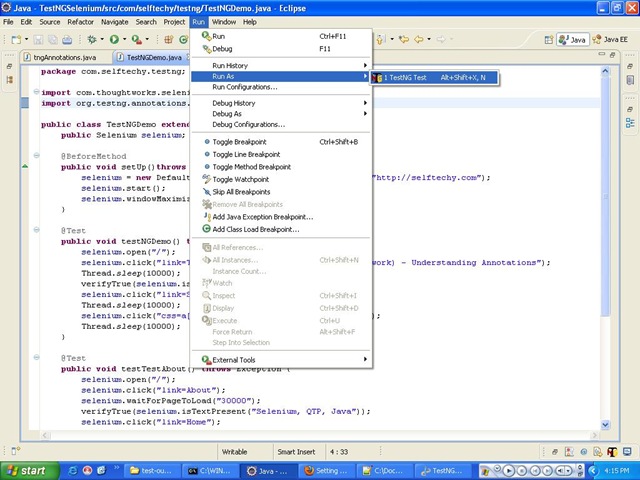
# 12. Automation Testing

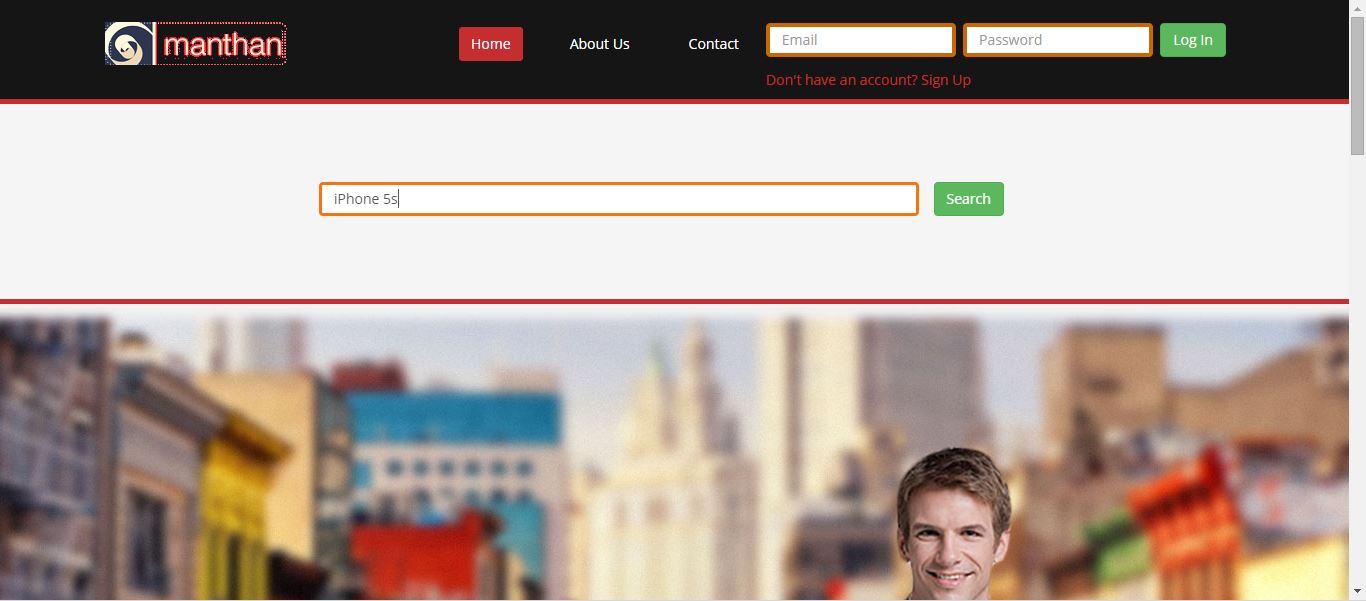
We have used **selenium** for automating our application. It comes with the integration of web driver. We have used Java programming language as our web driver. For setting up the environment, we used Maven based project. It’s easy to build with all the dependencies. We used Eclipse as our IDE.

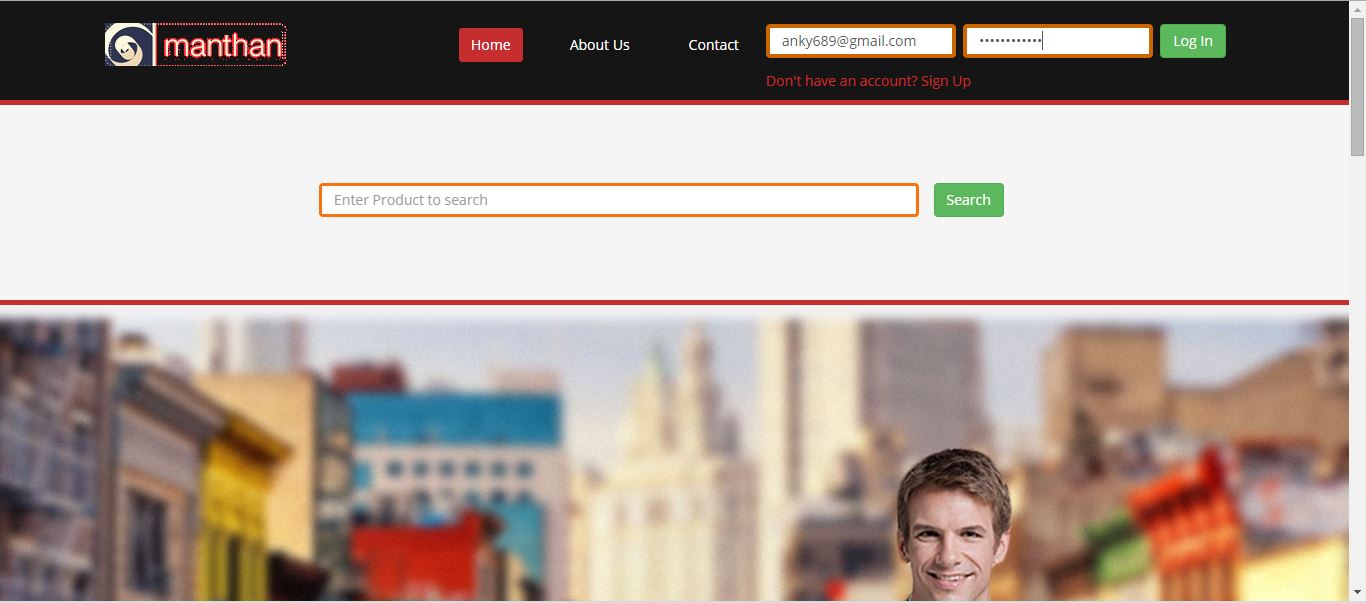
The testing scope that were covered:

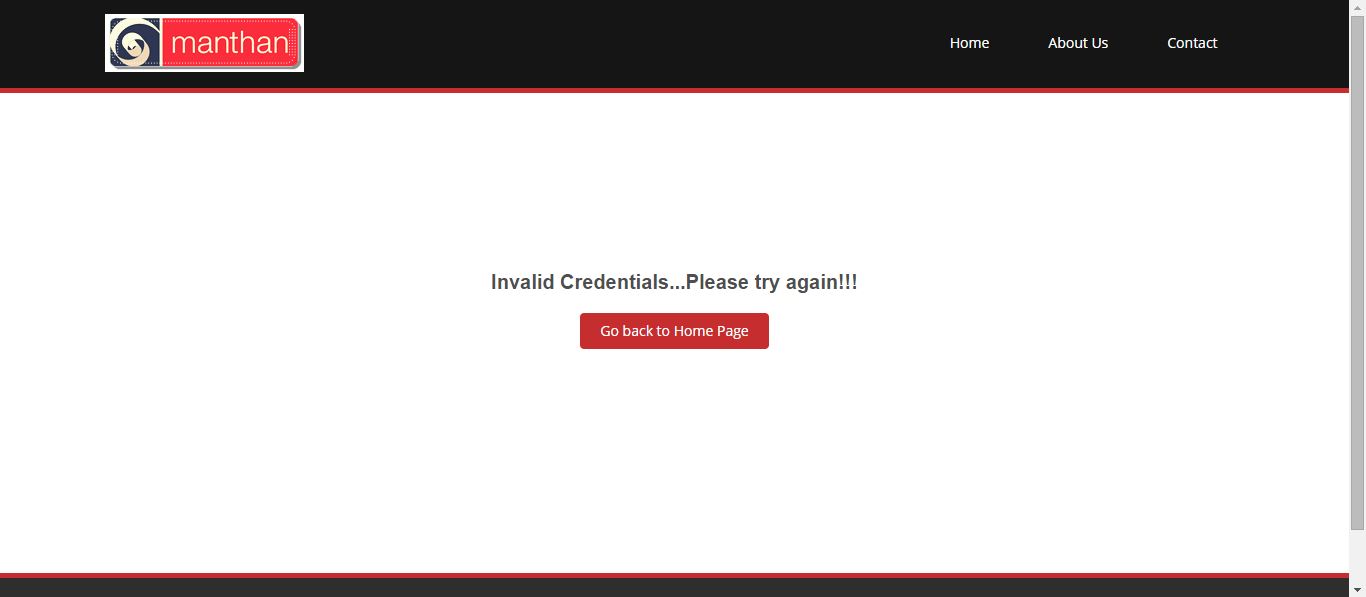
* User Interface Testing
* Cross browser testing

We made sure that each component designed in the user interface comes up. The test case covered testing each component on each page & verifying our project URL on different browsers.









# 13. Cross Browser Compatibility

Our application supports all browsers; it provides all the features to function in every environment. The boot strap library allows it to use it on mobile devices as well. We did an automating testing for IE, Mozilla, and Chrome etc. The user interface design has been done in such a way that the design does not degrades or bow out in different browsers. It is important to consider cross browser compatibility when designing a good web user interface.

# 14. JavaScript Libraries

**D3.js**

D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG and CSS. D3’s emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

D3 allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document. For example, you can use D3 to generate an HTML table from an array of numbers. Or, use the same data to create an interactive SVG bar chart with smooth transitions and interaction.

D3 is not a monolithic framework that seeks to provide every conceivable feature. Instead, D3 solves the crux of the problem: efficient manipulation of documents based on data. This avoids proprietary representation and affords extraordinary flexibility, exposing the full capabilities of web standards such as CSS3, HTML5 and SVG. With minimal overhead, D3 is extremely fast, supporting large datasets and dynamic behaviors for interaction and animation. D3’s functional style allows code reuse through a diverse collection of [components](https://github.com/mbostock/d3/wiki/API-Reference) and [plugins](https://github.com/d3/d3-plugins).

**jQuery**

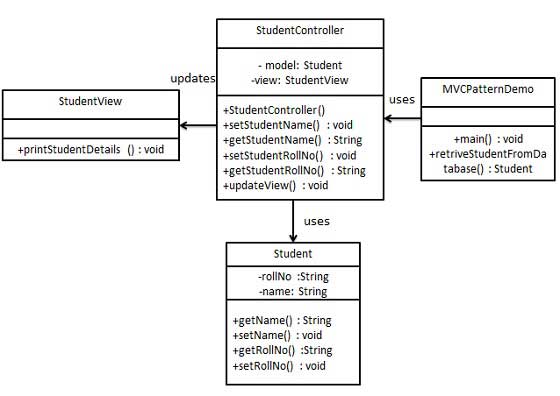
jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

# 15. Design Pattern

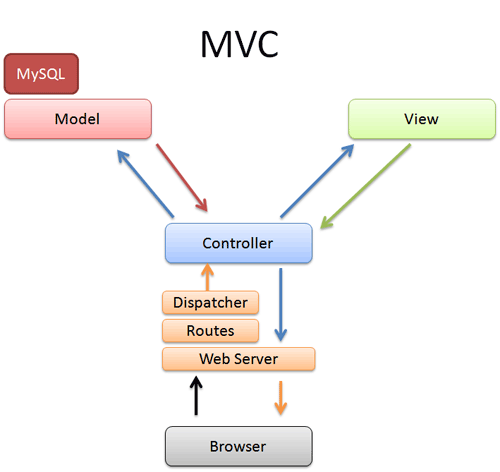
Our project is built based on Model View Controller framework where View comprises of web pages built using Twitter bootstrap, HTML, CSS and JSP. It implements 4 basic functionalities that is: Login, Sign Up, get results from query and share your links based on preferences to display and contrast evaluation results.

Say for example if we take sttudent object acting as a model. StudentView will be a view class which can print student details on console and StudentController is the controller class responsible to store data in Student object and update view StudentView accordingly.

MVCPatternDemo, our demo class will use StudentController to demonstrate use of MVC pattern.



Example: Design Pattern



MVC Architecture

MVC Pattern stands for Model View Controller, where Model works on data layer. It may also have the logic to makes changes to the controller in case of any changes to data. View is the visual representation of data, while Controller works as a layer between model and view and controls data flow into model object and make changes to view when there is any change to the data.

# 16. Localization

We have used and implemented local storage while implementing our application.

Lets understand with a piece of code-snippet:

function saveProductState() {

if (!supportsLocalStorage()) { return false; }

localStorage[".Product.in.progress"] = gProductInProgress;

for (var i = 0; i < kNumPieces; i++) {

localStorage[".piece." + i + ".row"] = gPieces[i].row;

localStorage[".piece." + i + ".column"] = gPieces[i].column;

}

localStorage[".selectedpiece"] = gSelectedPieceIndex;

localStorage[".selectedpiecepurchased"] = gSelectedPiecePurchased;

localStorage[".movecount"] = gMoveCount;

return true;

}

As you can see, it uses the localStorage object to save whether there is a Product in progress (gProductInProgress, a Boolean). If so, it iterates through the pieces (gPieces, a JavaScript Array) and saves the row and column number of each piece. Then it saves some additional Product state, including which piece is selected (gSelectedPieceIndex, an integer), whether the piece is in the middle of a potentially long series of hops (gSelectedPieceHasSearched, a Boolean), and the total number of moves made so far (gProductCount, an integer).

On page load, instead of automatically calling a newProduct() function that would reset these variables to hard-coded values, we call a resumeProduct() function instead. Using HTML5 Storage, the resumeProduct() function checks whether a state about a Product-in-progress is stored locally. If so, it restores those values using the localStorage object.

function resumeProduct() {

if (!supportsLocalStorage()) { return false; }

gProductInProgress = (localStorage["Product.in.progress"] == "true");

if (!gProductInProgress) { return false; }

gPieces = new Array(kNumPieces);

for (var i = 0; i < kNumPieces; i++) {

var row = parseInt(localStorage["piece." + i + ".row"]);

var column = parseInt(localStorage[".piece." + i + ".column"]);

gPieces[i] = new Cell(row, column);

}

gNumPieces = kNumPieces;

gSelectedPieceIndex = parseInt(localStorage[".selectedpiece"]);

gSelectedPieceHasMoved = localStorage[".selectedpiecehasmoved"] == "true";

gMoveCount = parseInt(localStorage[".movecount"]);

drawBoard();

return true;

}

The most important part of this function is the caveat that I mentioned earlier in this chapter, which I’ll repeat here: *Data is stored as strings. If you are storing something other than a string, you’ll need to coerce it yourself when you retrieve it.* For example, the flag for whether there is a Product in progress (gProductInProgress) is a Boolean. In the saveProductState() function, we just stored it and didn’t worry about the datatype:

localStorage[".Product.in.progress"] = gProductInProgress;

But in the resumeProduct() function, we need to treat the value we got from the local storage area as a string and manually construct the proper Boolean value ourselves:

gProductInProgress = (localStorage[".Product.in.progress"] == "true");

Similarly, the number of moves is stored in gMoveCount as an integer. In the saveProductState() function, we just stored it:

localStorage[".movecount"] = gMoveCount;

But in the resumeProduct() function, we need to coerce the value to an integer, using the parseInt() function built into JavaScript:

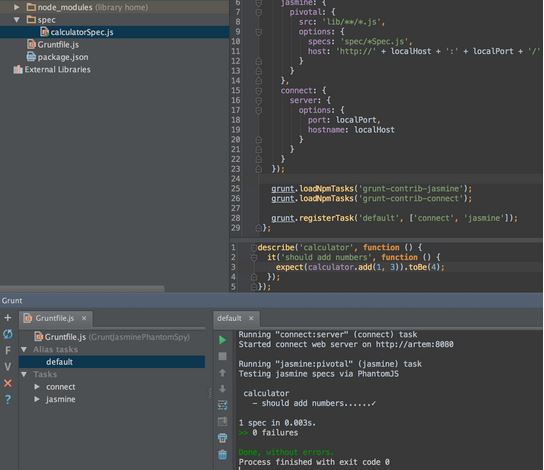
gMoveCount = parseInt(localStorage[".movecount"]);

# 17. Profiling

## Spy-js

Wondering what is slowing down your front-end or what to optimize in your code to get better javascript performance? Want to continuously see how changes being introduced are affecting code execution speed? With spy-js you can quickly get an idea of how long do certain events and functions take to execute and find good candidates for your javascript tuning?

In a nutshell, spy-js is a tool for JavaScript developers that allows to simply debug/trace/profile JavaScript on running on different platforms/browsers/devices. It fills gaps that existing browser development tools have and tackles common development tasks from a different angle and we have used for profiling of our application.



# 18. SEO

We have implemented SEO by using the following parameters:

* + 1. Creating unique and accurate page titles.
    2. Accurately Describing the page content.
    3. Create unique title tags for each page
    4. Use brief but descriptive titles
    5. Make use of descriptive tags
    6. Accurately summarizing page content
    7. Using unique descriptions for each page
    8. Improving the structure of urls
    9. Creating a simple directory structure
    10. Making the site easier to navigate

We considered these parameters to implement SEO, i.e. search engine optimization.

# 19. Profiling

There are three primary tactics that are we used for SEO pagination:

* Classic Method (using noindex)
* View All Method
* Rel Prev/Next Method

We have implemented View All Method.

The most elegant method is to utilize a View All page. In this approach, all component pages rel canonical back to the View All.

There are a few requirements for this approach:

The View All must load quickly; at least 3 seconds end-to-end. Maile Ohye pointed out at SMX West that even if load times are excessive, if the page can load progressively the user experience will not suffer as much (since content will be viewable on the page immediately).

The truth is, this is a “real world” goal and while not ideal, reflects the actual load times that we see on large sites.

Another requirement for the View All method is to ensure all products, or items, that are included on the component pages are featured on the View All itself.

This ensures that there won’t be anything left out of the crawl, as pages annotated with rel canonical tags will not necessarily have links within their HTML crawled. It will also ensure there is a relevant match between what is being folded together in the paginated series.

Users tend to love view all pages. In our experience and testing, pages with a lot of products or items all featured at once convert much higher than landing pages with a smaller selection of products. But the pages need to be fast. All component pages in the series transfer their equity to the View All in a fairly direct fashion

# 20. Individual Contribution

* 1. Ankur Gupta: UI, Testing, Mongodb, Share Functionality
  2. Ayush Mittal: Product Display, Semantic3, Cloud Deploy
  3. Deven Pawar: UI, Nodejs, Mongodb, Share Functionality, jQuery
  4. Abhinaya: UI, Nodejs, Semantic3 api, Rest.