**SWOT Technology**

Contents

[1. Javascript: 1](#_Toc463504640)

[a. Strength 1](#_Toc463504641)

[b. Weakness: 1](#_Toc463504642)

[c. Op 2](#_Toc463504643)

[d. Threats 2](#_Toc463504644)

[2. No-SQL (mongoDB, redit, elasticsearch): 2](#_Toc463504645)

[a. Strength: 2](#_Toc463504646)

[b. Weakness 2](#_Toc463504647)

[c. Op 2](#_Toc463504648)

[d. Threats 3](#_Toc463504649)

[3. AngularJS 3](#_Toc463504650)

[a. Strength 3](#_Toc463504651)

[b. Week 3](#_Toc463504652)

[c. Op 4](#_Toc463504653)

[d. Threats 4](#_Toc463504654)

[**4.** **Node.js** 4](#_Toc463504655)

[a. Strengths 4](#_Toc463504656)

[b. Weaknesses 4](#_Toc463504657)

[c. Opportunities 4](#_Toc463504658)

[d. Threats 5](#_Toc463504659)

# Javascript:

## Strength

* **Javascript is executed on the client side**  
  This means that the code is executed on the user's processor instead of the web server thus saving bandwidth and strain on the web server.
* **Javascript is a relatively easy language**  
  The Javascript language is relatively easy to learn and comprises of syntax that is close to English. It uses the DOM model that provides plenty of prewritten functionality to the various objects on pages making it a breeze to develop a script to solve a custom purpose.
* **Javascript is relatively fast to the end user**  
  As the code is executed on the user's computer, results and processing is completed almost instantly depending on the task (tasks in JavaScript on web pages are usually simple so as to prevent being a memory hog) as it does not need to be processed in the site's web server and sent back to the user consuming local as well as server bandwidth.
* **Extended functionality to web pages**  
  Third party add-ons like Greasemonkey enable Javascript developers to write snippets of Javascript which can execute on desired web pages to extend its functionality. If you use a website and require a certain feature to be included, you can write it yourself and use an add-on like Greasemonkey to implement it on the web page.

## Weakness:

* **Security Issues**  
  Javascript snippets, once appended onto web pages execute on client servers immediately and therefore can also be used to exploit the user's system. While a certain restriction is set by modern web standards on browsers, malicious code can still be executed complying with the restrictions set.
* **Javascript rendering varies**  
  Different layout engines may render Javascript differently resulting in inconsistency in terms of functionality and interface. While the latest versions of javascript and rendering have been geared towards a universal standard, certain variations still exist. [Website Usability Consultants all over the world](http://www.nextprise.com/website-usability-consulting.php) make a living on these differences, but it enrages thousands of developers on a daily basis.

## Op

* JavaScript is currently the most popular programming language with the richest OSS module ecosystem, and the only programming language with a really convincing universal deployment story: The “write once, run everywhere” dream that Java aspired to — Java fell short. JavaScript pulled it off.
* Universal JavaScript is an app that runs on servers, on the web platform, and even in native devices (check out React Native), sharing a bulk of the same app logic (not just libraries) across all target platforms. If you’re not using Universal JavaScript to write your apps, you’re wasting a lot of time and money.

## Threats

* + **Information leakage**
* With the rise of interactive web applications the sensitive information stored and managed with web browsers are become and attractive target for malware authors. Using JavaScript an attacker can collect valuable information that are sent to a server under their control to perform more dangerous operations (like cross-site request forgery) or to be sold to third parties.
  + **Control flow hijacking attacks.**
* Traditional attacks that leverage memory errors in the browser can be performed also using JavaScript. Even though their potential harm is always notable, their employing is becoming less prevalent with the rise of drive-by download attacks.

# No-SQL (mongoDB, redit, elasticsearch):

## Strength:

* Uninterrupted access/high availability
* Scalability
* Security and flexibility
* Freedom to choose

## Weakness

* Porting the applications
* No Normalization, unions or joins
* Lost updates

## Op

* Huge investments
* Data Intensive applications

## Threats

* FOSS business model
* FUD (Fear, uncertainity and doubt) amongst Users

# AngularJS

## Strength

* No need to use observable functions; Angular analyses the page DOM and builds the bindings based on the Angular-specific element attributes. That requires less writing, the code is cleaner, easier to understand and less error prone.
* Angular modifies the page DOM directly instead of adding inner HTML code. That is faster.
* Data binding occurs not on each control or value change (no change listeners) but at particular points of the JavaScript code execution. That dramatically improves performance as a single bulk Model/View update replaces hundreds of cascading data change events.
* Quite a number of different ways to do the same things, thus accommodating to particular development styles and tasks.
* Extended features such as dependency injection, routing, animations, view orchestration, and more.
* Supported by IntelliJ IDEA and Visual Studio .NET IDEs.
* Supported by Google and a great development community.

## Week

* Angular is big and complicated. With multiple ways to do the same thing it is hard to tell which way is better for particular task. Mastering Angular over the “Hello world” level requires considerable efforts. Different developers’ coding styles and habits might complicate integration of different components into a whole solution.
* The lifecycle of Angular application is complex, and to master it you really need to read the code. Compile and link are not intuitive, and specific cases can be confusing (recursion in compile, collisions between directives etc.).
* As the project grows with time, you most likely will need to throw away existing implementations and create new versions using different approaches. Angular implementations scale poorly.
* More than 2000 watchers can severely lag the UI. That limits the complexity of your Angular forms, especially big data grids and lists.

## Op

* now a days we know that mobile apps become more popular. Most mobile apps are in ionic framework. If we have knowledge in angularjs concepts we can easily develops apps using angularjs..In the website development also we can use this.  
   Now many IT companies using [angularjs Resources and Information.](http://angularjs.it/). It is a growing technology and also very easy to learn and  use. We can develop apps and sites in a professional way.. And also many career opportunities are there for angularjs.  
  To put it another way, the modern enterprise needs to be able to embrace fast-moving change on the Web, which also implies the need to accept fast-moving change within frameworks like AngularJS that make Web application development simple and powerful.  
  So in my view it is a technology which offers a better career for all professionals.

## Threats

* Retail Data Hacks
* Smartphone Vulnerability Threats
* Phishing Attacks & Social Engineering
* Healthcare Data Hacks
* Attacks on Banks

# Node.js

## Strengths

* vibrant and engaging community
* changing fast, so you can be an influence
* modular from its beginning
* non-blocking / very fast
* ability to share code between client and server JavaScript a very common language, especially on the web
* the community tends to be biased in support of non-relational databases such as MongoDB

## Weaknesses

* not good with computation heavy apps ([reference](http://www.srirangan.net/2012-05-node-js-critics-which-part-of-event-driven-non-blocking-io-model-you-dont-understand))
* not as mature as "alternatives" (though, to be fair, there are no alternatives)
* event-based programming can be difficult to work with: debugging, promises, etc; it takes a lot of training / experience to do it well

## Opportunities

* maturity: simply needs time, since they have the community involvement
* develop comprehensive frameworks so you don't have to build your own framework from scratch

## Threats

* I see no threats provided the status quo continues; in other words, they are on the right track. A very low probability threat would be something like a resurgence / revolution in Ruby or Python development, or if PHP's current revolution wins over the Node.js crowd.