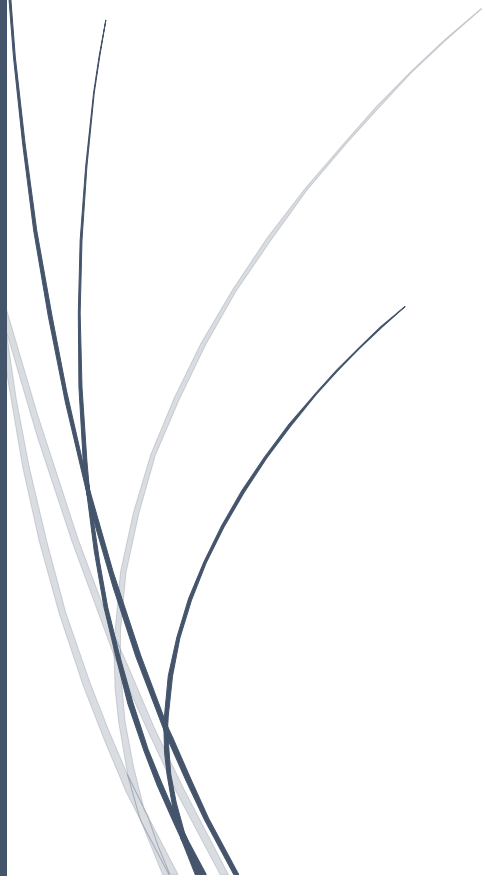
2/15/2020

Making a Dynamic Website: A Simple JavaScript Guide

Upendar Rao Thaduri, Karu Lal



技术与管理回顾

HTTPS://UPRIGHT.PUB/INDEX.PHP/TMR/

Technology and Management Review 技术与管理回顾

Vol 5, No 1 (2020)

Making a Dynamic Website: A Simple JavaScript Guide

[1]**Upendar Rao Thaduri,** *ACE Developer, iMINDS Technology Systems, Inc., PA 15243, USA*
[2]**Karu Lal,** *Integration Engineer, Ohio National Financial Services, USA*

Abstract:
JavaScript is a high-level programming language that complies, mostly just-in-time and adheres to the ECMAScript standard. Dynamic typing, prototype-based object orientation, and first-class functions are all features of this programming language. Thanks to its multi-paradigm nature, it has event-driven, functional, and imperative programming styles that can all be supported. This study is dedicated to the JavaScript language and not the parts specific to Web pages or other host environments. The utility of scripting languages has been demonstrated in various application domains. This study shows that the most common programming language, JavaScript, is in the top 10 languages. And then there is JavaScript, which has become an essential tool for bringing client-side functionality to Web applications as it has grown in popularity.

Keywords: Object-Oriented Programming, JavaScript, Multitier System, Web Design

INTRODUCTION

JavaScript (JS) is a computer language that developers use to make web pages dynamic. JavaScript is a lightweight, cross-platform, single-threaded, interpreted, and compiled object-oriented programming language. It allows developers to dynamically generate material that can be updated and use animations, pop-up menus, clickable buttons, and control multimedia, among other things. JavaScript can be used to do various tasks on the client or the server side. JavaScript is used to add interactive components to web pages that engage users. While HTML and CSS give web pages structure and style, JavaScript adds interactive elements. Without JavaScript, the majority of websites on the Internet would be unchanging.

CORE CONCEPTS

Variables: JavaScript relies heavily on variables. These variables store numbers and objects.
"This is a variable!";

Functions: JScript functions arrange and reuse code to perform actions. Processes reduce duplicate logic, simplify coding, and make code easier to read (Dongyang et al., 2016). JavaScript functions, arguments, parameters, return statements, and scope are defined and used here.

Functions principles we must understand include-

- Expressions of functions
- Function() constructor.
- The self-invoking functions
- Consider functions as values.
- Functions as objects
- Arrow functions
- Generator function
- Predefined functions

Arrays: JavaScript arrays store values. JavaScript array objects are variables so that we can treat them like other data types. Array Operations, Methods, Map, Reduce, and Filter are other vital concepts.

real array = ['a', 'b', 'c'];

Loops
Loops execute code repeatedly. Structures like loops are basic but powerful. We must understand these loops:
For loop
The for-in loop
For-of loop
The while loop
The do-while loop: for (var i = 0; i < 100; i++) { console.log(i); }

Objects: JavaScript objects have states and behaviors like real-world entities. Knowing how to create things, properties, methods, and accessors helps.

Data Types: We should know String, Bool, Numbers, Objects, Functions, and Undefined. The type of function is often used to determine the variable type.

Conversion Types: This involves data type conversion. Conversions may include

- Converts implicitly
- Converting strings to numbers and vice versa
- Converting arrays to columns and vice versa
- Primitive to primitive conversions
- Integer-to-float and vice, etc.

Control Flow: If, if..else, nested ifs, continue, break, and other programming constructs must be understood.

Dates: This involves constructing dates, formatting JavaScript dates, growing date objects, retrieving the current date and time, etc.

Operators: Logical, Arithmetic, Comparison, Null and Undefined, Quality, and Inequality check operators are operators.

Scope: The scope of a function captures its context variables. Understanding how to access local and global variables within and outside a function is crucial.

USES OF JAVASCRIPT APPLICATIONS

Websites and web apps use JavaScript extensively. Let us explain JavaScript's practical uses in several sectors.

1. Web Development: Developers utilize JavaScript to create websites. Developed in Netscape, JS lets developers create dynamic, interactive web pages to communicate with users and perform complicated actions. It also allows users to add material to a document without reloading the page (Desamsetti, 2016). Most websites use JavaScript to validate PDFs, widgets, and Flash apps. Some of the biggest tech companies employ JavaScript to improve user experience. Some famous JavaScript-built websites are Google, YouTube, Facebook, Wikipedia, Yahoo, Amazon, eBay, Twitter, and LinkedIn (Desamsetti & Mandapuram, 2017).

2. Web apps: Strong web apps are built using JavaScript frameworks (Laine et al., 2011). Users can zoom in on a map in Google Maps by clicking and dragging. JavaScript controls the browser without contacting the servers. Popular JavaScript front-end frameworks for web apps include React Native, React, Angular, and Vue. Netflix and PayPal used AngularJS and APIs (Mandapuram, 2016).

3. Presentations: Interactive webpages using JavaScript are popular (Gutlapalli, 2016a). HTML-based slide decks can be created with RevealJs and BespokeJs. The RevealJs makes interactive slide decks with transitions, themes, and slide backgrounds in all CSS color formats. BespokeJs is a feature-rich framework with scaling, animated bullet lists, syntax highlighting, etc. JavaScript lets non-programmers create websites and presentations (Lal et al., 2018).

4. Server Apps: JavaScript is used to develop server-side software in Node.js. Programmers can write, test, and debug rapid, scalable network applications. JavaScript manages HTTP requests and generates content. Big firms like Walmart, PayPal, Uber, GoDaddy, and others use Node.js for server infrastructure.

5. Webservers: Node.js lets JavaScript developers build web servers. Since Node.js is event-driven, it goes to the next call without waiting for the previous one. Without buffering, servers send data quickly. The HTTP module creates servers with createServer().

6. Games: Another notable JavaScript application is web game development. JavaScript and HTML5 are crucial to JS game creation. EaselJS delivers excellent game graphics. HTML5 allows full web access without Flash. JavaScript and HTML5 power complex browser games like Tower Building, CrossCode, and HexGL.

7. Art: Canvas is a new HTML5 JavaScript feature that makes painting 2D and 3D graphics on web pages accessible (Gutlapalli, 2016b). This has enabled browser-based digital art projects. Create JavaScript art as a digital artist.

8. Smartwatch Apps: Pebble.js lets developers build Pebble watch apps in JavaScript. Make a smartwatch app with simple JavaScript.

9. Mobile apps: JavaScript is powerful for creating apps for non-web environments, such as items off the Internet (Desamsetti & Mandapuram, 2017). With mobile device use at an all-time high, JavaScript frameworks ease mobile app development for iOS, Android, and Windows. React Native lets developers construct cross-platform mobile apps with a universal front end for Android and iOS.

10. Flying Robots: Finally, JavaScript can program a flying robot. Using Node.js, users may operate miniature robots, creative maker projects, and IoT devices. JavaScript opens the realm of drones, flying robots, and quadcopters.

FUNDAMENTALS OF JAVASCRIPT

We could not have picked a better path, so let me congratulate you. For many years, JavaScript has been the most popular programming language. The percentage of developers who use JavaScript is now around 78% (Gutlapalli, 2017a). Its prevalence is greater than that of Python, Java, HTML, and CSS combined. If you are proficient in advanced JavaScript, you can create more complicated websites without using any frameworks. Additionally, regardless of which framework you choose to study, the process will be much easier (Kienle, 2010).

We have been working on the fundamental ideas of JavaScript for a while now and can easily use them (Gutlapalli, 2017b). I am going to talk about things like arrays, functions, and objects, as well as variables and data types. Are you and I thinking the same thing? Just right! The time has arrived for us to become seasoned software developers, so get your preferred beverage (coffee is recommended) and settle in for the ride!

Callbacks: Enhance function functionality without modifying code. This method is often used in modules (libraries/plugins) whose code cannot be modified.

Promise: The object represents an operation that produces or will produce a value. Promises reduce deeply nested callbacks by wrapping asynchronous task results.

Async and Await: Asynchronous or callback code is easier to maintain. This idea is crucial to JavaScript asynchronous programming.

Prototype: inherits class properties, including methods and variables. After building a solid foundation, we can tackle more advanced topics.

Inheritance: In other object-oriented languages, methods defined on the superclass can be accessed by extending the subclass.

Regular Expressions: A search pattern consisting of a series of characters. These expressions specify text searches.

Setters and Getters: Object attributes that call functions when set/gotten.

WEB PLATFORM TOOLS

- Cookies: allow for the storage of user information on web pages. Know how to create, read, update, and delete cookies.
- Web/Local Storage: Offers temporary key-value string storage and persistence. Knowing how data is kept is crucial.
- Fetch API is crucial. It requests resources.
- Document Object Model: is an object-oriented representation of structured texts like XML and HTML. The DOM maintains data consistency. Knowledge of DOM and manipulation is crucial.
- Web Worker: Allows background script execution without interfering with the user interface. This must also be understood.

OBJECT ORIENTED PROGRAMMING JAVASCRIPT

Object-oriented Programming sees data as a vital part of program development and restricts its movement. It secures data to the function that works on it and prevents extraneous functions from modifying it. OOP divides a problem into objects and constructs data and functions around them.

Object: The primary run-time bodies in an object-oriented system are objects (Fylaktopoulos et al., 2016). They can represent a place, person, account, data table, or whatever else the software needs. Objects can represent user-defined vectors, time, and lists.

Consider two software objects, "customer" and "account ."The customer object can request the bank balance.

Creating Objects in JavaScript

```
var student = {
      name: "Chris,"
```

```
        age: 21,
        studies: "Computer Science",
    };
    document.getElementById("demo").innerHTML = student.name + " of the age " + student.age
+ " studies " + student.studies;
```

Creating objects using the new keyword.

```
var student = new Object();
    student.name = "Chris",
    student.age=21,
    student.studies = "Computer Science";
    document.getElementById("demo").innerHTML = student.name + " of the age " + student.age
+ " studies " + student.studies;
```

Creating an object using the object constructor.

```
function stud(name, age, studies){
        this.name = name;
        this.age = age;
        this.studies = studies;
    }
    var student = stud("Chris," 21, "Computer Science");
    document.getElementById("demo").innerHTML = student.name + " of the age " + student.age
+ " studies " + student.studies;
```

Classes: It is common knowledge that objects store the data and the functions necessary to work with it (Gutlapalli, 2017c). On the other hand, with the assistance of classes, the two can be combined into a single user-defined data type. A class allows for the creation of an unlimited number of objects. Each object has its own data set corresponding to a specific class. Therefore, a collection of objects with the same type is what we mean when discussing a class.

For example, consider the class “Fruits”. We can create multiple objects for this class -
Fruit Mango;
This will create an object mango belonging to the class fruit.

Class Implementation in JavaScript

JavaScript uses the ES6 standard to define classes. Consider the following example.

```
class Cars {
    constructor(name, maker, price) {
      this.name = name;
      this.maker =  maker;
      this.price = price;
```

```
  }
  get details(){
     return (`The name of the car is ${this. name}.`)
  }
 }
 let car1 = new Cars('Rolls Royce Ghost,' 'Rolls Royce,' '$315K');
 let car2 = new Cars('Mercedes AMG One,' 'Mercedes,' '$2700K');
 console.log(car1.name);
 console.log(car2.maker);
 console.log(car1.getDetails());
```

Encapsulation: Encapsulation combines data and its associated functions into a single entity known as a class (Alexandru, 2015). The most notable characteristic of a class is known as its "data encapsulation," which ensures that the data it contains is inaccessible to the outside world and can be accessed only by the functions wrapped inside the class. These functions act as an interface between the data stored in the object and its software (Desamsetti, 2016). Wrapping the property and function in a single unit is essential to encapsulation.

```
class Emp_details{
   constructor(name,id){
      this.name = name;
      this.id = id;
   }
   add_Address(add){
      this.add = add;
   }
   getDetails(){
      console.log(`Employee Name: ${this.name}, Address: ${this.add}`);
   }
}
let person1 = new Emp_details('Anand',27);
person1.add_Address('Bangalore');
person1.getDetails();
```

Here, the class holds the data variables name and id along with the functions add_Address and getDetails. All are encapsulated within the class Emp_details.

Inheritance: Inheritance is the process by which members of one class take on the characteristics of members of a different class (Lal & Ballamudi, 2017). It gives credence to the idea of a hierarchical classification system. Take, for example, the thing called a "car," which belongs to the classes "Light Weight Vehicles" and "Vehicles."In object-oriented programming, reusability is ensured through the inheritance idea. This indicates that new features can be added to an existing class without the class being modified. This objective can be accomplished by deriving a new class from an existing one (Gutlapalli et al., 2019).

Polymorphism: Polymorphism is a concept employed in the object-oriented paradigm, allowing us to use the same function in various ways (Andrew, 2012). This eliminates unnecessary repetition and makes the code snippet applicable to various scenarios. Generic, overloading, and structural sub-typing are the three methods used in JavaScript to implement polymorphism (Mandapuram et al., 2018). Let us examine each one of them in further depth.

Parametric Polymorphism: The purpose of incorporating parametric polymorphism into our code is to make it more general. More specifically, it prevents our code from being dependent on the definition of data types. A more generic code ignores the types while maintaining the types-safety guarantee.

Overloading is another name for ad-hoc polymorphism: Ad-hoc polymorphism is how overloading in JavaScript can be accomplished. When creating functions, we can take advantage of a technique known as ad-hoc polymorphism (Thaduri et al., 2016). This technique produces functions that respond differently to various kinds of input values. Take, for instance:

+ when applied to integers and floats, it demonstrates the addition property. + when applied to strings and arrays, it performs the concatenation operation.

CLIENT-SIDE WEB APIS

When we build client-side JavaScript for websites or applications, we immediately run into Application Programming Interfaces (APIs) (Yunsik et al., 2017). APIs, or application programming interfaces, are coding elements that allow developers to manipulate data from other websites or services and many parts of the web browser and operating system on which the site runs. In this lesson, we will investigate what application programming interfaces (APIs) are and how to use some of the most common APIs you are likely to encounter frequently during your development work.

Web APIs: First things first, let us begin by taking a high-level look at application programming interfaces (APIs): what are they, how do they function, how do you utilize them in your code, and how are they organized? In addition, we will investigate the many primary categories of APIs and the kinds of applications that may be accomplished with each one.

Manipulating: When it comes to developing web pages and applications, one of the most typical things we will want to do is change web documents in some kind (Mandapuram, 2017a). This is typically accomplished by utilizing the Document Object Model (DOM), a collection of application programming interfaces for manipulating HTML and style information and extensively using the Document object (Mandapuram, 2017b). In this study, we will examine how to use the Document Object Model (DOM) in detail and look at several other intriguing APIs that can affect your environment in interesting ways.

Fetching data: In contemporary websites and applications, one of the tasks frequently performed is getting individual data items from the server to update web page areas without loading an entirely new page (Larkin, 2015). This is a fairly typical practice. The functionality and behavior

of websites have been profoundly altered as a direct result of this seemingly insignificant change. In this piece, the idea will be broken down, and we will also investigate the technologies—like XMLHttpRequest and the Fetch API—that make its implementation practicable.

Third-party APIs: The application programming interfaces (APIs) we have discussed thus far are built into the browser, but not all are. Many large websites and services, such as Google Maps, Twitter, Facebook, PayPal, and others, provide APIs that enable developers to make use of their data (for example, displaying your Twitter stream on your blog) or services (for example, displaying custom Google Maps on your site, or using Facebook login to log in your users). These APIs are called application programming interfaces (APIs) (Mandapuram & Hosen, 2018). This article compares and contrasts the APIs provided by browsers with those provided by third parties and demonstrates some typical applications of the latter.

Graphics: The browser comes equipped with several highly potent graphics programming tools, such as the Scalable Vector Graphics (SVG) language and APIs for drawing on HTML canvas> components (for more information, see The Canvas API and WebGL). These tools allow for the creation of vector graphics. This article will give us an overview of the Canvas API and point us toward additional resources that will help us learn more.

Video and audio APIs: HTML has components for embedding rich media in documents, such as 'video> and 'audio>. These elements, in turn, come with their application programming interfaces (APIs) for managing playing, seeking, and other similar functions. This tutorial will teach us how to perform typical activities, such as generating individualized playback controls.

Client-side storage: Today's Web browsers have various technologies that enable users to store data connected to websites and retrieve it when necessary. These technologies allow users to keep data for an extended period, save websites offline, and do other things (Thodupunori & Gutlapalli, 2018). This essay will explain the absolute fundamentals of how these things function.

PERFORMANCE

HTML, CSS, and JavaScript are the fundamental building blocks for websites. We need to establish a positive user experience if we want to construct websites and applications that people will want to use and that will attract and keep users. Ensuring the material loads quickly and is sensitive to user input is essential to providing a positive user experience (Lennon, 2018). This aspect of the web is referred to as web performance, and throughout this module, we will concentrate on the fundamentals of constructing websites that operate well. Although the remainder of our learning material for beginners made an effort to adhere as closely as possible to online best practices such as performance and accessibility, it is still a good idea to concentrate mainly on such issues and ensure we are familiar with them.

Web Performance: We know that web performance is crucial, but what exactly is "web performance?" This article provides an introduction to the various aspects of performance, such as the loading and rendering of web pages, the process by which our material is loaded into the

browser of our users so that it can be viewed, and the various demographics of users that need to be taken into account when thinking about performance.

Perceive performance: is more important than the actual speed of our website measured in milliseconds since it reflects how quickly visitors feel the site is loading for them. The actual page load time, idling, responsiveness to user engagement, and the smoothness of scrolling and other animations all impact users' perceptions of how long a page takes to load. Along with a discussion of the various loading metrics, animation metrics, and responsiveness measurements, this article also includes recommendations for best practices to enhance user perception, even if the actual timings still need to be addressed.

Measuring performance: Now that we understand a few performance measures, we will delve deeper into performance tools, metrics, and APIs, as well as discuss how we may include performance into the workflow of web development.

Multimedia (images): Regarding improving website performance, media optimization is frequently the "lowest hanging fruit." It is not impossible to serve different media files according to each user agent's capabilities, dimensions, and pixel densities. In this post, we will cover photographs' effect on performance and the strategies that may be used to decrease the amount of data transmitted along with each image.

Multimedia (video): Regarding improving website performance, media optimization is frequently the "lowest hanging fruit." In this post, we analyze the effect that video content has on performance, and we address strategies that help enhance performance, such as eliminating audio tracks from videos that are playing in the background.

JavaScript performance optimization: JavaScript, when utilized correctly, can make it possible to have engaging and immersive web experiences (Paola & Albert, 2017). On the other hand, if it is not optimized correctly, it can drastically slow down download times, render times, in-app performance, battery life, and user experiences (Deming et al., 2018). This article explains several JavaScript best practices that should be considered so that even complex material can perform well.

HTML performance optimization: The order in which our markup is sourced and specific properties might affect the performance of our website. We may significantly enhance the quality of the user experience by reducing the amount of DOM nodes utilized, ensuring that the appropriate attributes and order are applied when incorporating material like styles, scripts, media, and scripts from third-party sources, and decreasing the total number of DOM nodes (Dekkati et al., 2019). This article takes a comprehensive look at the various ways in which HTML can be applied to ensure the highest possible level of performance.

CSS performance optimization: CSS may be a less critical optimization target for enhanced performance. However, certain CSS features have a more significant impact on performance than others do (Dekkati & Thaduri, 2017). In this article, we look at specific CSS features that affect performance and offer strategies for managing styles to ensure that performance is not adversely affected.

Fonts and performance: A look at whether or not we need to incorporate external fonts, and if we do, how to include the typefaces our design demands while having as little of an impact as possible on the performance of our website (Chen et al., 2019).

Mobile performance: Because it is so common for people to use their mobile devices to access the Internet and because all mobile platforms come equipped with full-featured web browsers, it is essential to take into account the performance of our website's content on mobile devices, which may have restricted bandwidth, processing power, and battery life. This article examines performance factors unique to mobile devices (McGranaghan, 2011).

The economic justification for high web performance: A developer can do several things to boost performance, but the question is: how fast is fast enough? How exactly will we persuade those in authority of the significance of these efforts? After optimization, what steps should be taken to prevent bloat from occurring again? In this post, we will discuss techniques to persuade management, create a performance culture and performance budget, and provide methods to ensure that regressions do not make their way into our code base (Bodepudi et al., 2019).

CONCLUSION

JavaScript is a language that is based on prototypes. It is compatible with the object-oriented programming paradigm as well as the functional programming paradigm. It is neither an OOP nor an FP in its entirety. The JavaScript language does not support the creation of classes. Although we use the class keyword, this is merely syntactic sugar for our prototype-based inheritance system. Object-oriented programming in JavaScript is supported by four primary pillars: classes, encapsulation, inheritance, and objects. Classes are like blueprints for real-world objects, while objects are instances of classes or prototypes that have already been created. Encapsulation is the process of tying data and actions together, and inheritance is the concept that we can take some attributes from the objects' parents while defining new properties and actions within the child objects. Polymorphism is a notion that is utilized in the oops in JavaScript. It gives us the ability to use the same function in a variety of various guises. In order to provide interactivity to websites, HTML and CSS are typically used in conjunction with JavaScript. In certain circumstances, JavaScript may use third-party libraries to provide advanced functionality to an application without requiring the developer to write the necessary code from scratch on every occasion. We hope this article helps explain what JavaScript is and how it can be used in various applications and websites. Check out our beginner's guide to coding if you are interested in learning JavaScript or any other programming language.

REFERENCES

Andrew, A. M. (2012). Data output from Javascript. *Kybernetes, 41*(10), 1604-1606. https://doi.org/10.1108/03684921211276774

Bodepudi, A., Reddy, M., Gutlapalli, S. S., & Mandapuram, M. (2019). Voice Recognition Systems in the Cloud Networks: Has It Reached Its Full Potential?. *Asian Journal of Applied Science and Engineering*, *8*(1), 51–60. https://doi.org/10.18034/ajase.v8i1.12

Chen, S., Thaduri, U. R., & Ballamudi, V. K. R. (2019). Front-End Development in React: An Overview. *Engineering International*, *7*(2), 117–126. https://doi.org/10.18034/ei.v7i2.662

Dekkati, S., & Thaduri, U. R. (2017). Innovative Method for the Prediction of Software Defects Based on Class Imbalance Datasets. *Technology & Management Review*, *2*, 1–5. https://upright.pub/index.php/tmr/article/view/78

Dekkati, S., Lal, K., & Desamsetti, H. (2019). React Native for Android: Cross-Platform Mobile Application Development. *Global Disclosure of Economics and Business*, *8*(2), 153-164. https://doi.org/10.18034/gdeb.v8i2.696

Deming, C., Dekkati, S., & Desamsetti, H. (2018). Exploratory Data Analysis and Visualization for Business Analytics. *Asian Journal of Applied Science and Engineering*, *7*(1), 93–100. https://doi.org/10.18034/ajase.v7i1.53

Desamsetti, H. (2016). Issues with the Cloud Computing Technology. *International Research Journal of Engineering and Technology (IRJET)*, *3*(5), 321-323.

Desamsetti, H., & Mandapuram, M. (2017). A Review of Meta-Model Designed for the Model-Based Testing Technique. *Engineering International*, *5*(2), 107–110. https://doi.org/10.18034/ei.v5i2.661

Dongyang, H., Jun, C., Hao, W. (2016). ISPRS International Journal of Geo-Information. *Basel* *5*(7), 105. https://doi.org/10.3390/ijgi5070105

Fylaktopoulos, G., Goumas, G., Skolarikis, M., Sotiropoulos, A. & Maglogiannis, I. (2016). An overview of platforms for cloud-based development. *SpringerPlus*, *5*, 38. https://doi.org/10.1186/s40064-016-1688-5

Gutlapalli, S. S. (2016a). An Examination of Nanotechnology's Role as an Integral Part of Electronics. *ABC Research Alert*, *4*(3), 21–27. https://doi.org/10.18034/ra.v4i3.651

Gutlapalli, S. S. (2016b). Commercial Applications of Blockchain and Distributed Ledger Technology. *Engineering International*, *4*(2), 89–94. https://doi.org/10.18034/ei.v4i2.653

Gutlapalli, S. S. (2017a). Analysis of Multimodal Data Using Deep Learning and Machine Learning. *Asian Journal of Humanity, Art and Literature*, *4*(2), 171–176. https://doi.org/10.18034/ajhal.v4i2.658

Gutlapalli, S. S. (2017b). The Role of Deep Learning in the Fourth Industrial Revolution: A Digital Transformation Approach. *Asian Accounting and Auditing Advancement*, *8*(1), 52–56. Retrieved from https://4ajournal.com/article/view/77

Gutlapalli, S. S. (2017c). An Early Cautionary Scan of the Security Risks of the Internet of Things. *Asian Journal of Applied Science and Engineering*, *6*, 163–168. Retrieved from https://ajase.net/article/view/14

Gutlapalli, S. S., Mandapuram, M., Reddy, M., & Bodepudi, A. (2019). Evaluation of Hospital Information Systems (HIS) in terms of their Suitability for Tasks. *Malaysian Journal of Medical and Biological Research*, *6*(2), 143–150. https://doi.org/10.18034/mjmbr.v6i2.661

Kienle, H. M. (2010). It's About Time to Take JavaScript (More) Seriously. *IEEE Software, 27*(3), 60-62. https://doi.org/10.1109/MS.2010.76

Laine, M., Shestakov, D., Litvinova, E. and Vuorimaa, P. (2011).Toward Unified Web Application Development. *IT Professional*, *13*(5), 30-36. https://doi.org/10.1109/MITP.2011.55

Lal, K., & Ballamudi, V. K. R. (2017). Unlock Data's Full Potential with Segment: A Cloud Data Integration Approach. *Technology &Amp; Management Review*, *2*, 6–12. https://upright.pub/index.php/tmr/article/view/80

Lal, K., Ballamudi, V. K. R., & Thaduri, U. R. (2018). Exploiting the Potential of Artificial Intelligence in Decision Support Systems. *ABC Journal of Advanced Research*, *7*(2), 131-138. https://doi.org/10.18034/abcjar.v7i2.695

Larkin, H. (2015). A framework for programmatically designing user interfaces in JavaScript. *International Journal of Pervasive Computing and Communications*, *11*(3), 254–269. https://doi.org/10.1108/IJPCC-03-2015-0014

Lennon, B. (2018). JavaScript Affogato: Programming a Culture of Improvised Expertise. *Configurations 26*(1), 47–72. https://doi.org/10.1353/con.2018.0002

Mandapuram, M. (2016). Applications of Blockchain and Distributed Ledger Technology (DLT) in Commercial Settings. *Asian Accounting and Auditing Advancement, 7*(1), 50–57. https://4ajournal.com/article/view/76

Mandapuram, M. (2017a). Application of Artificial Intelligence in Contemporary Business: An Analysis for Content Management System Optimization. *Asian Business Review*, *7*(3), 117–122. https://doi.org/10.18034/abr.v7i3.650

Mandapuram, M. (2017b). Security Risk Analysis of the Internet of Things: An Early Cautionary Scan. *ABC Research Alert*, *5*(3), 49–55. https://doi.org/10.18034/ra.v5i3.650

Mandapuram, M., & Hosen, M. F. (2018). The Object-Oriented Database Management System versus the Relational Database Management System: A Comparison. *Global Disclosure of Economics and Business*, *7*(2), 89–96. https://doi.org/10.18034/gdeb.v7i2.657

Mandapuram, M., Gutlapalli, S. S., Bodepudi, A., & Reddy, M. (2018). Investigating the Prospects of Generative Artificial Intelligence. *Asian Journal of Humanity, Art and Literature*, *5*(2), 167–174. https://doi.org/10.18034/ajhal.v5i2.659

McGranaghan, M. (2011). ClojureScript: Functional Programming for JavaScript Platforms. *IEEE Internet Computing*, *15*(6), 97-102. https://doi.org/10.1109/MIC.2011.148

Paola, G., Albert, S., (2017). Scientific Annals of Computer Science. *Iasi, 27*(1), 19-76. https://doi.org/10.7561/SACS.2017.1.19

Thaduri, U. R., Ballamudi, V. K. R., Dekkati, S., & Mandapuram, M. (2016). Making the Cloud Adoption Decisions: Gaining Advantages from Taking an Integrated Approach. *International Journal of Reciprocal Symmetry and Theoretical Physics*, *3*, 11–16. https://upright.pub/index.php/ijrstp/article/view/77

Thodupunori, S. R., & Gutlapalli, S. S. (2018). Overview of LeOra Software: A Statistical Tool for Decision Makers. *Technology & Management Review*, *3*(1), 7–11.

Yunsik, S., Seman, O., Yangsun, L. (2017). International Information Institute (Tokyo). *Koganei*, *20*(5A), 3259-3266.