Front-End Development in React: An Overview

Songtao Chen[1], Upendar Rao Thaduri[2], Venkata Koteswara Rao Ballamudi[3*]

[1]Jiujiang Vocational and Technical College, Jiangxi, China
[2]ACE Developer, iMINDS Technology Systems, Inc., Pittsburgh, PA 15243, USA
[3]Sr. Software Engineer, HTC Global Services, USA

[*]Corresponding Contact:
Email: venkata.bvk@gmail.com

ABSTRACT

In front-end development, the function that react.js plays is becoming increasingly important, providing developers with new options to create new applications. This article discusses how react.js assists in constructing user interfaces and the benefits it offers in building front-ends. According to the survey carried out by Web Technology Surveys, react.js is utilized by a significant percentage of all websites today. We will not exaggerate the situation if we declare that React.JS is used everywhere. When it comes to websites, new audiences have various interests. This article discusses critical aspects of the framework, including its advantages over competing frameworks, how it works, and its architecture.

Key words:
JavaScript, React.JS, Node.JS, Angular, Virtual DOM, Web Development, Front-End Development

12/31/2019 **Source of Support:** None, **NoConflict of Interest:** Declared

INTRODUCTION

These days, web developers more commonly refer to the products they construct as web applications instead of web pages. Although there isn't a clear-cut distinction between the two, web applications are typically more interactive and dynamic. They give the user the ability to carry out actions and get a reaction in response to those actions. The front-end development industry has never been more intriguing or complicated than it is right now. Brand-new applications, libraries, frameworks, and plugins are released every other day. There is plenty of information to absorb. Grab's online team has been staying current with the most recent recommendations for best practices, and as a result, our web apps now use the modern JavaScript environment (Mandapuram, 2016). In the traditional model, the browser is responsible for rendering HTML after receiving it from the server. It is necessary to perform a full page refresh whenever the user navigates to a new URL, and the server will transmit the newly generated HTML for the page to the user.

On the other hand, client-side rendering is utilized rather than server-side rendering in current SPAs. The first page of the app is loaded from the server into the browser, along with any scripts (frameworks, libraries, or app code) and stylesheets necessary for the

application as a whole. No action is taken to refresh the current page whenever the user visits a different page. The HTML5 History API makes the necessary changes to the page's URL. The browser sends AJAX queries to the server to retrieve any new data needed for the new page. Typically, this data is stored in JSON format. After that, the SPA automatically refreshes the page with the data using JavaScript, which it had previously retrieved while loading the initial page. This architecture operates in a manner that is comparable to that of native mobile applications. The advantages are that the application responds more quickly, and users are spared the flash between page navigations due to full-page refreshes. The server receives fewer HTTP requests because the same assets do not have to be downloaded repeatedly whenever a page is loaded (Desamsetti & Mandapuram, 2017). Clear separation of responsibilities between the client and the server; it is simple to construct new clients for various platforms (such as mobile, chatbots, and smartwatches) without having to rewrite the code that runs on the server. Altering the technological stack individually on the client and server is also possible, provided the API contract is not violated (Mandapuram, 2017a).

Initial page load time is longer since the framework, app code, and assets needed for many pages must be loaded simultaneously. Configuring our server to direct all requests to a single entry point and then letting client-side routing take over from there is an additional step that has to be completed on our server. This can be done by following the instructions in the following section. The rendering of content in SPAs depends on JavaScript, but not all search engines can run JavaScript when they scan websites; therefore, search engines may interpret the material on our page as empty (Gutlapalli, 2017a). This unintentionally impacts the app's SEO (search engine optimization). A clear client-server separation scales well for bigger engineering teams, as the client and server codes may be developed and deployed independently. This is because classic server-side rendered programs are still a viable choice. This is especially true for Grab, as we often have numerous client apps simultaneously accessing the same API server (Mandapuram, 2017b). The structuring of client-side JavaScript has grown increasingly significant due to the trend among web developers toward creating applications rather than pages. It is common practice to insert bits of jQuery code onto each website page that is rendered server-side to provide user interactivity. However, when building huge apps, jQuery alone is not adequate. In the end, jQuery is primarily a library for manipulating the Document Object Model (DOM), and it is not a framework; hence, it does not establish a distinct structure and organization for our application.

Frameworks written in JavaScript have been developed to offer higher-level abstractions over the DOM (Gutlapalli, 2017b). These abstractions make it possible to store state in memory rather than in the DOM itself. The ability to reuse recommended concepts and best practices when developing applications is another advantage of employing frameworks. Because the code is structured according to a format that is already familiar to the new engineer on the team, even though they are not familiar with the code base itself, it will be easier for them to grasp the code because it has been arranged according to a format that is already established in the framework. New engineers will benefit from tapping into their colleagues' and community's knowledge and experience to get up to speed as quickly as possible. Popular frameworks provide a large number of tutorials and guides.

NEW-AGE JAVASCRIPT

It is essential to become conversant in the language of the web, which is either JavaScript or ECMAScript, before delving into the myriad of facets that comprise the construction of a JavaScript online application. JavaScript is a highly flexible programming language that can

be used to create web servers, native mobile apps, and desktop applications, among other things.

Before 2015, the most recent significant change was ECMAScript 5.1, released in 2011. On the other hand, JavaScript has suddenly witnessed a substantial surge of advances within a relatively short period in recent years. ECMAScript 2015, formerly known as ECMAScript 6, was made available to the public in 2015, and with it came a plethora of syntactic features designed to make the process of developing code more manageable. If we are interested in learning more about the past of JavaScript, Auth0 has penned a good piece on the topic that we may read. Even now, not all browsers have completed the implementation of all of the requirements of the ES2015 specification. Developers can write ES2015 code in their applications thanks to tools like Babel, which then transpires that code down to ES5 to be compatible with browsers (Gutlapalli, 2017c).

It is essential to have a working knowledge of both ES5 and ES2015. Although ES2015 has been around for a while, most open-source software and Node.js applications are still written in ES5. We may not be able to use syntax from ES2015 if we perform debugging in our browser's console. On the other hand, the documentation and example code for many contemporary libraries, some of which we will discuss further below, are still written in ES2015. At Grab, we use babel-preset-env to take advantage of the productivity gain that comes from the syntactic changes that the future of JavaScript delivers, and we are pleased with the results thus far. As native browser support expands for further ES language capabilities, babel-preset-env intelligently identifies which Babel plugins are required (which new language features are not supported and must be transpired). babel-preset-stage-3 is a complete specification that will most likely be implemented in browsers; if we prefer employing language features that have already been stabilized, we may discover that babel-preset-stage-3 is a better option for us because it is more suited. ES5 should receive at least a day or two of review, and ES2015 should be investigated. The ES2015 features "Arrows and Lexical This," "Classes," "Template Strings," "Destructuring," "Default/Rest/Spread operators," and "Importing and Exporting modules" are the ones that are utilized the most frequently by developers.

USER INTERFACE — REACT

React is the JavaScript project that has dominated the front-end ecosystem recently. Facebook smarties designed and open-sourced React. Developers write and compose React web interface components. React introduces radical concepts and challenges developers to rethink best practices (Capała & Skublewska-Paszkowska, 2018). Web developers were taught for years to create HTML, JavaScript, and CSS separately. React recommends writing HTML and CSS in JavaScript. After testing it, this doesn't seem that ridiculous. Because front-end development is moving toward component-based development. Features of React:

Declarative: We wish to see something, not how to get it. Using jQuery, developers had to alter the DOM to switch app states. In React, we transform the state within the component, and the view will update itself according to the state. Render() markup makes it easy to predict the component's appearance. A pure prop-state function, the idea is functional. In most circumstances, a React component is defined by props (external parameters) and state (internal data). The same support and state yield the same view. Functional features and pure functions are easy to test. React's well-defined interfaces make testing easy. We can try a member by passing it into different props and states and comparing the displayed output.

Maintainable: Component-based views encourage reuse. React code is self-documenting when prop Types are defined so the reader knows what to use. Finally, our perspective and logic are contained in the component and should not affect other features. If the element has the same props, large-scale refactoring is trivial.

High Performance: React employs a virtual DOM (not a shadow DOM) and re-renders anything when a state changes. Why is a virtual DOM needed? Modern JavaScript engines are fast, but DOM reading and writing are slow. React stores a lightweight DOM virtualization in memory. Re-rendering everything is misleading. React re-renders the in-memory DOM, not the actual DOM. When component data changes, a new virtual representation is constructed and compared to the previous model. Patching the browser DOM with minimal modifications is then done.

The React API surface is smaller; there are few APIs to understand, and they change rarely. The React community is one of the largest, and it has a robust ecosystem of tools, open-sourced UI components, and excellent online resources to learn React. Many React development tools improve the developer experience. React Developer Tools lets us inspect, view, and change component props and states in our browser (Ciliberti, 2017). Hot reloading with webpack lets us see code changes in our browser without refreshing. Front-end development requires editing code, saving, and restoring the browser. Hot reloading helps by eliminating the last step. Facebook provides coded scripts to migrate code to new APIs after library changes. This makes upgrading easy. The Facebook team deserves praise for their React development efforts.

Over time, more performant view frameworks than React have developed. React may be a challenging library, but its ecosystem, usability, and benefits are top-notch. Facebook is also rewriting the reconciliation process to speed up React (Madaj, 2018). React has taught us to create better code, maintain web projects, and be better engineers. We like it. Watch the React homepage lesson on making a tic-tac-toe game to learn about React. The highly-rated free course React Fundamentals by React Router's developers React specialists is for additional in-depth learning. Complex ideas not provided by React documentation are also covered (Naiki et al., 2018). Facebook's Create React App scaffolds React projects with minimal configuration and is ideal for new React projects.

STATE MANAGEMENT — FLUX/REDUX

As our app expands, its structure may become messy. React has no elegant mechanism to exchange and show common data across app components (Desamsetti, 2016). React is merely the view layer; it doesn't influence how we arrange our app's model or controller in typical MVC approaches. To solve this, Facebook created Flux, a unidirectional data flow app design that complements React's composable view components. Find out how Flux works. Unidirectional data flow makes app changes easier to track and more predictable. Concerns are separated - Flux architecture components have clear roles and are strongly decoupled. It works nicely with declarative programming since the store can update the view without defining how to switch states. Flux is not a framework. Therefore, developers have explored several Flux pattern implementations. Finally, Redux won. Redux is the recommended state management library for React developers, combining Flux, Command pattern, and Elm design. The main concepts are:

- A single JavaScript object (POJO) describes the app state.
- Dispatch an action (POJO) to modify the state.

- A reducer is a pure function that creates a new state from the current state and action.

Though uncomplicated, the concepts allow apps to:

- Render state on server and boot client.
- Record and reverse changes across the app.
- Quickly implement undo/redo functionality.

Dan Abramov, Redux's designer, has meticulously written documentation and created video tutorials for understanding basic and advanced Redux. Their Redux learning resources are invaluable.

COMBINING VIEW AND STATE

Although using Redux with React is not required, it is strongly suggested because the two libraries complement each other and work very well together. Both React and Redux share some concepts and characteristics, including the following:

- The Functional Composition Paradigm states that React is responsible for composing views, which are themselves pure functions, whereas Redux is accountable for collecting pure reducers, which are themselves functions. When the same set of inputs is used, it is possible to predict the output.
- It Is Not Hard To Reason About — This phrase may be familiar to us, but do we clearly understand what it denotes? We take it to mean we have control and comprehension over our code. Our code performs in the ways that we anticipate it will, and when there are issues, we can locate them with relative ease. According to our observations, React and Redux make the debugging process more manageable. Because the data flow is unidirectional, it is much simpler to track the flow of data (server answers, user input events), and it is also much simpler to establish in which layer the issue originates.
- Structure Comprised of Several Layers Each app or Flux architecture layer performs a single, unambiguous function and is assigned specific duties. Writing tests for pure functions is a pretty straightforward process.

DEVELOPMENT EXPERIENCE

A significant amount of work has been put into developing tools, such as Redux DevTools, that will assist in inspecting and debugging the application while it is being designed (Gutlapalli, 2016a). Our program may have to handle asynchronous calls, such as queries sent to a remote API. Redux-thunk and redux-saga are two programs that were developed specifically to address these issues. Understanding them requires familiarity with functional programming and generators, which could add extra time to the process (Thaduri et al., 2016). Our recommendation is to deal with it only when it's essential. React-redux is the official React binding for Redux, and it is a pretty straightforward concept to grasp.

CODING WITH STYLE — CSS MODULES

CSS rules describe HTML element appearance. Writing decent CSS is hard. Learning to design sustainable and scalable CSS takes years of practice and the frustration of shooting oneself in the foot. With its global namespace, CSS is meant for web documents, not components-based web programs (Gutlapalli, 2016b). Thus, experienced front-end

developers have developed methods like SMACSS, BEM, SUIT CSS, etc., to help users write organized CSS for complex applications.

CSS approaches encapsulate styles artificially via conventions and rules. They break when developers don't comply. As we may have noticed, the front-end ecosystem is full of tools, and some of them solve some of the challenges with creating CSS at scale. "At scale" means several developers touch the same stylesheets on a significant project. There is no community-agreed technique for producing CSS in JS, but we hope a winner, like Redux, will emerge among Flux implementations. For now, we trust CSS Modules. CSS modules solve the global namespace problem in CSS by letting us write local, component-encapsulated styles. This functionality is tool-based. Large teams can design modular, reusable CSS without conflicting with other program elements with CSS modules. After all, CSS modules are still compiled into globally namespaced CSS that browsers recognize; it's essential to comprehend raw CSS (Thodupunori & Gutlapalli, 2018). Codecademy's HTML & CSS course is suitable for CSS beginners. Next, learn about the Sass preprocessor, which improves CSS syntax and encourages style reuse. Check out CSS modules and CSS techniques.

TYPES — FLOW

Static typing improves app writing. They can spot typical code mistakes early. Types document and promote code readability. As codebases increase, types become more important because they boost refactoring confidence. When new team members know what values each object stores and what parameters each function expects and returns, it's easier to onboard them. Adding types to our code increases verbosity and syntax learning. However, this learning cost is upfront and amortized. Types have more pros than cons in complex projects where code maintainability counts and developers shift over time. We fixed a bug in a code base we hadn't touched in months. Because of the types, I could refresh my coding knowledge and feel confident in my patch. The two most significant competitors for adding static types to JavaScript are Facebook's Flow and Microsoft's TypeScript. There is no apparent winner in the battle. For now, we're utilizing flow. Flow offers a smaller learning curve than TypeScript and requires less work migrating code. Being designed by Facebook, Flow integrates with React better out of the box. Flow author James Kyle compares TypeScript to Flow. Due to their identical syntax and semantics, moving from Flow to TypeScript is easy, and we will reevaluate the issue later. Using one is better than none.

BUILD SYSTEM — WEBPACK

Because configuring webpack can be a time-consuming process that may be off-putting to developers who are already overwhelmed by the onslaught of new information they need to learn for front-end development so that this section will be kept to a minimum, Webpack is a module bundler, which, in a nutshell, means that it compiles a front-end project and all of its dependencies into a final bundle that can then be served to customers. In most cases, the webpack configuration will already be set up for the project, and developers will seldom need to adjust it. In the long term, having a working knowledge of webpack will be beneficial. Webpack is responsible for the capabilities allowing hot reloading and CSS modules to existing. Learning webpack with the webpack tour provided by SurviveJS has proven to be our most helpful resource. It is a valuable supplement to the official documentation, and we recommend that we follow the tour first and then go to the documentation later whenever the need for extra customization arises.

PACKAGE MANAGEMENT — YARN

If we look inside our node_modules directory, we will be astounded by the sheer volume of guides. Each Babel plugin and Lodash function is considered a separate package. When we have more than one project going at once, these packages will be repeated for each one and will be identical. When we start a new project and execute the npm install, these packages will be downloaded repeatedly, even though they are already present on our machine in one of our earlier projects (Mandapuram & Hosen, 2018). In addition, there was the issue of non-determinism in the packages that were installed via npm install. When the CI server pulls in minor updates to some packages that contain breaking changes, it causes some of our CI builds to fail. This happens at the moment in time when the CI server is installing the dependencies. This would not have occurred if library authors had honored the server, and engineers would not have assumed that API contracts would be respected at all times if they had not made that assumption.

These issues can be resolved by using yarn. The case of non-deterministic installed packages is handled by a file called yarn.lock, which assures that each installation results in the same file structure in node_modules across all computers. This file is located in the yarn directory. Yarn uses a global cache directory that is stored locally on our machine; this means that already downloaded packages do not need to be downloaded anymore. This also makes it possible to install dependencies without an internet connection! These are the Yarn commands that are used the most frequently. Using the npm versions rather than the corresponding yarn commands is OK because most other yarn commands are very similar to their npm counterparts. The yarn upgrade-interactive command is one of our favorites because it makes updating dependencies a breeze, which is especially useful considering that the average modern JavaScript project has many dependencies.

LEVEL UP REACT CONDITIONALS

Any application that uses React must have solid conditionals as one of its core components. Our apps use conditionals to show or conceal certain features and components. In a nutshell, if we want to be a successful React developers, we need to be familiar with designing effective conditionals. Let's go through all of the primary patterns we need to be familiar with to design clear and concise conditionals and the anti-patterns we should try to avoid.

Use if-statements primarily. No need for else or else-if.

Let's begin with the most fundamental form of conditional that React has to offer. If we have data, we intend to show it. In such a case, we have nothing else to display. Consider for a moment that we are retrieving an array of post data from an API. During the time that it is retrieving the data, the value of posts is undefined. We may use a straightforward if-statement to test whether or not that value is present. The fact that we are leaving earlier is the key to the success of this pattern. If the requirement is met (if the variable! posts has a boolean value of true), our component will not display anything because we will have returned null if statements are helpful in situations where there are many conditions to be checked for. For instance, if we want to check for loading and error conditions before displaying our data, we could do the following:

We may recycle the if-statement and do not need to write if-else or if-else-if. This allows us to reduce the code we need to write while maintaining its readability.

Use the ternary operator to write conditionals in JSX

When we wish to end the program early and display either nothing or a completely different component, if-statements are a terrific tool to use. But what if we don't want to place a conditional in a different file from our returning JSX and instead want to write it immediately within it? Expressions, not statements, should be used within our JSX code while working with React since a word is "something that resolves to a value." Because of this, we are required to use only ternaries when writing conditionals in our JSX rather than if statements (Mandapuram et al., 2018). For instance, if we wanted to display one nested component on a screen that was the size of a mobile device and another on a larger screen, the ideal solution would be to use a ternary:

When it comes to utilizing ternaries, most developers believe that this style is the only one they can use. We can leave some of these ternaries directly in the JSX supplied, saving space in our component tree. Remember that we can assign the result of a ternary to a variable, which we can then utilize in any way we see fit because ternaries resolve to a value. This is because ternaries are values.

No other condition? Use the && (and) operator

We will likely need to utilize a ternary in our JSX code frequently, but we will eventually conclude that we do not want to display anything if the condition is not satisfied. The following is an example of what this ternary would look like condition. Component />: not set to any value.

Use the and or operator if we don't have any other conditions to check:

Switch statements for multiple conditions

What if we find ourselves in a circumstance in which we are confronted with more than simply one or two distinct conditions? We can write more than one if statement, but all of these if accounts, as we have seen in a previous section, go above the JSX that is returned. An excessive number of if-statements can prevent our components from becoming cluttered (Deming et al., 2018). How can we improve the readability of our code? We can frequently extract many conditions into a single component comprising a switch statement. For instance, we have a Menu component that allows us to toggle and display a variety of tabs, and we have this capability.

As we can see below, we have tabs that can display data regarding users, chat, and rooms:

Because we are utilizing a separate MenuItem component in conjunction with a switch statement, the conditional logic does not clutter up our parent Menu component, and it is simple for us to determine which piece will be displayed based on the current state of the menu.

Want conditionals as components? Try JSX Control Statements

The capability to use plain JavaScript within our React components is quite handy. On the other hand, check out the JSX control statements provided by the React framework if we want conditionals that are even more declarative and straightforward. By using the following command, we'll be able to incorporate it into our React projects (Dekkati & Thaduri, 2017). Additionally, we can include it in the following format in. babel file:

This plugin for Babel enables us to use React components within our JSX code, making it much simpler for readers to comprehend the conditionals we write.

Examining a sample is the most effective method for gaining an understanding of how such a library may be put to use. Let's go back through some of our older examples and rework them with the help of JSX control statements:

No if or ternary statement is anywhere in sight, and our component structure is relatively easy to follow.

We should experiment with JSX control statements in the next React project to determine whether we need a library such as this one.

CONCLUSION

By incorporating some fundamental JavaScript functionalities, the JS framework enables the construction of aesthetically pleasing user interfaces. As a result of the fact that it supplies us with a markup syntax that is highly similar to HTML, it is simple to use and put into practice. The Virtual DOM is the most critical feature we offer, as it eliminates the need to reload a page and significantly boosts the application's overall performance. Because our application is built on JavaScript, we also have access to a package manager called NPM. This gives us an easier way to install external dependencies and simplifies managing these packages. The lifecycle methods that React offers allow us to modify the lifecycle of the class components we use. When it comes to the construction of applications or user interfaces, many developers are turning to one of the most popular frameworks: React. As a result, there is little doubt that, in the not-too-distant future, there will be an increase in demand for this framework and the features it offers. The fact that React is a library that aids businesses in achieving their objectives strengthens the company's position in the market. It ensures its continued relevance in the years to come.

REFERENCE

Capała, Ł., & Skublewska-Paszkowska, M. (2018). Comparison of AngularJS and React.js frameworks based on a web application. *Journal of Computer Sciences Institute, 6*, 82–86. http://dx.doi.org/10.35784/jcsi.645

Ciliberti, J. (2017). Creating Modern User Experiences Using React.js and ASP.NET Core. In *ASP.NET Core Recipes*, 361–409. http://dx.doi.org/10.1007/978-1-4842-0427-6_11

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

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

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

Madaj, T. (2018). Výkonnostní testy pro chytré TV, set-top boxy a herní konzole. Master's thesis, *Vysoké učení technické v Brně*. Fakulta informačních technologií. http://www.nusl.cz/ntk/nusl-386024

Mandapuram, M. (2016). Applications of Blockchain and Distributed Ledger Technology (DLT) in Commercial Settings. *Asian Accounting and Auditing Advancement*, *7*(1), 50–57. Retrieved from 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

Naiki, S., Kohana, M., Okamoto, S., and Kamada, M. (2018). A Graphical Front-End Interface for React.js. In *Advances in Network-Based Information Systems*, 887–896. Cham: Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-98530-5_79

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.

--0--

Archive Link:

https://abc.us.org/ojs/index.php/ei/issue/archive