УДК 519.6:004.93 DOI 10.36910/775.24153966.2021.72.29

Абгарян Юра Серьожайович,
спеціаліст кафедри транспортних технологій, Запорізького національно технічного університету, розробник програмного забезпечення SoftServe
ORCID iD 0000-0001-8519-2539

ЩОДО ПИТАННЯ ПОБУДОВИ АРХІТЕКТУРИ СУЧАСНИХ КОРПОРАТИВНИХ ДОДАТКІВ НА NODE.JS

У статті розкрито питання побудови архітектури сучасних корпоративних додатків на Node.js. Визначено, що Node.js - це надзвичайно потужна платформа на основі JavaScript, побудована із застосуванням Google Chrome JavaScript V8 Engine, яка використовується для розробки веб-додатків з інтенсивним введенням-виводом. Наголошено, що програму, яка запускається на сервері та обробляється клієнтським браузером, використовуючи Інтернет для доступу до всіх ресурсів цієї програми, зазвичай легко розбити на три частини: клієнт, це коли користувач взаємодіє з інтерфейсною частиною веб-програми; сервер, який відповідає за прийом клієнтських запитів, виконання необхідних завдань та надсилання відповідей клієнтам та база даних у якій зберігаються дані для веб-програми. Наголошено, що Node.js використовує архітектуру «однопотокового циклу подій» для обробки кількох одночасних клієнтів. Модель обробки Node.js заснована на моделі подій JavaScript разом із механізмом зворотного виклику JavaScript. Зазначається, що частини архітектури Node.js складаються із запитів, які можуть бути блокуючими (складними) або неблокуючими (простими) залежно від завдань, які користувач хоче виконати у веб-застосунку; серверу Node.js, який є серверною платформою, яка приймає запити від користувачів, обробляє їх та повертає відповіді відповідним користувачам; черги подій, яка зберігає вхідні запити клієнтів і передає їх один за одним у цикл подій; пулу потоків, що складається з усіх потоків, доступних для виконання деяких завдань, які можуть знадобитися для виконання запитів клієнта; циклу подій, що необмежено приймає запити та обробляє їх, а потім повертає відповіді відповідним клієнтам та зовнішніх ресурсів, які необхідні для блокування запитів клієнтів. Архітектура сучасних корпоративних додатків на Node.js. ґрунтується на застосуванні двох окремих напрямків: «зверху вниз» за відповідними ролями, що допомагає розробникам об'єднувати та роз'єднувати модулі, а також напрямок «розділяй і володарюй» у відповідності до виконуваних завдань, що допомагає розробникам розділити завдання на простіші модулі, одночасно дозволяючи розробляти декілька модулів. Розроблена архітектура покликана ефективно структурувати дизайн додатків для кращої ремонтопридатності та розширюваності з часом.

Ключові слова: *корпоративний додаток, розробка, архітектура, фреймворк, програмна платформа.*

Абгарян Юра Сережаевич

ПО ВОПРОСУ СТРОЙКИ АРХИТЕКТУРЫ СОВРЕМЕННЫХ КОРПОРАТИВНЫХ ДОПОЛНИТЕЛЕЙ НА NODE.JS

В статье раскрыты вопросы построения архитектуры современных корпоративных приложений на Node.js. Определено, что Node.js – это очень мощная платформа на основе JavaScript, построенная с применением Google Chrome JavaScript V8 Engine, которая используется для разработки веб-приложений с интенсивным вводом-выводом. Отмечено, что программа, запускаемая на сервере и обрабатываемая клиентским браузером, используя Интернет для доступа ко всем ресурсам этого приложения, обычно легко разбить на три части: клиент, это когда пользователь взаимодействует с интерфейсной частью веб-приложения; сервер, отвечающий за прием клиентских запросов, выполнение необходимых задач и отправку ответов клиентам и база данных, в которой хранятся данные для веб-приложения. Отмечено, что Node.js использует архитектуру «однопоточного цикла событий» для обработки нескольких клиентов. Модель обработки Node.js основана на модели событий JavaScript вместе с механизмом обратного вызова JavaScript. Отмечается, что части архитектуры Node.js состоят из запросов, которые могут быть блокирующими (сложными) или неблокирующими (простыми) в зависимости от задач, которые пользователь хочет выполнить в веб-приложении; серверу Node.js, который является серверной платформой, принимающей запросы от пользователей, обрабатывает их и возвращает ответы соответствующим пользователям; очереди событий, сохраняющая входящие запросы клиентов и передающая их один за другим в цикл событий; пулу потоков, состоящий из всех потоков, доступных для выполнения некоторых задач, которые могут потребоваться для выполнения запросов клиента; цикла событий, что неограниченно принимает запросы и обрабатывает их, а затем возвращает ответы соответствующим клиентам и внешним ресурсам, которые необходимы для блокирования запросов клиентов. Архитектура современных корпоративных приложений на Node.js. основывается на применении двух отдельных направлений: «сверху вниз» по соответствующим ролям, что помогает разработчикам объединять и разъединять модули, а также направление «разделяй и властвуй» в соответствии с выполняемыми задачами, что помогает разработчикам разделить задачи на более простые модули, одновременно разрешая разрабатывать несколько модулей. Разработанная архитектура призвана эффективно структурировать дизайн приложений для лучшей ремонтопригодности и расширяемости с течением времени.

Ключевые слова: *корпоративное приложение, разработка, архитектура, фреймворк, программная платформа.*

Abharian Yura

ON THE ISSUE OF BUILDING THE ARCHITECTURE OF MODERN ENTERPRISE APPLICATIONS ON NODE.JS

Abstract. The article reveals the issue of building the architecture of modern enterprise applications on Node.js. Node.js has been identified as an extremely powerful JavaScript-based platform built with the Google Chrome JavaScript V8 Engine, which is used to develop web applications with intensive I / O. It is emphasized that a program that runs on a server and is processed by a client browser using the Internet to access all resources of this program is usually easily divided into three parts: client, when a user interacts with the interface part of a web program; a server that is responsible for receiving client requests, performing necessary tasks and sending responses to clients, and a database that stores data for a web application. It is noted that Node.js uses a "single-threaded event loop" architecture to handle multiple concurrent clients. The Node.js processing model is based on the JavaScript event model along with the JavaScript callback mechanism. It is noted that parts of the Node.js architecture consist of queries that can be blocking (complex) or non-blocking (simple) depending on the tasks that the user wants to perform in the web application; Node.js server, which is a server platform that receives requests from users, processes them and returns responses to the appropriate users; an event queue that stores incoming customer requests and passes them one by one to the event loop; a thread pool consisting of all threads available to perform some tasks that may be required to execute client requests; an event loop that receives and processes requests indefinitely and then returns responses to the appropriate clients and the external resources needed to block client requests. Architecture of modern corporate applications on Node.js. is based on the use of two separate areas: "top-down" for the respective roles, which helps developers to combine and disconnect modules, and the direction of "divide and conquer" in accordance with the tasks performed, which helps developers to divide tasks into simpler modules, allowing to develop several modules at the same time. The developed architecture is designed to effectively structure the design of applications for better maintainability and extensibility over time.

***Key words**: corporate application, development, architecture, framework, software platform.*

Постановка проблеми. В умовах зростаючого ринку інформаційних додатків інтерес викликають різноманітні програми Інтернету речей (IoT), даний факт призводить до створення різних точок даних, зібраних з ряду різнорідних пристроїв, які збираються в єдине сховище для розробки додатків. Розподілений характер цих систем дозволив дослідникам перейти до хмарної інфраструктури для реалізації нових дизайнів додатків за підтримки інтегрованої обробки даних та ефективного управління ресурсами [1].

Однак, незважаючи на свою привабливість, впровадження таких хмарних програм не є тривіальним. Розробка ефективного хмарного додатка вимагає від програміста знання різних технічних областей від роботи сервера, локальних мов і мов веб-додатків до протоколів передачі даних. Наприклад, для ефективної веб-розробки розробникам зазвичай потрібно знання до п’яти різних мов програмування, таких як JavaScript, HTML, CSS, мови сервера, таких як PHP і SQL [2].

У якості фундаментальної основи для подолання цих проблем представлена програмна платформа Node.js [3] отримала значну популярність у спільнотах розробників з моменту її впровадження у 2009 році. Зокрема, Node.js – це масштабоване однопотокове середовище JavaScript на стороні сервера, реалізовано на C і C++ [4]. Незважаючи на останні розробки, підхід Node.js до розробки веб-додатків зробив його привабливою альтернативою більш традиційним платформам, таким як сервери Apache+PHP і Nginx.

Аналіз останніх досліджень і публікацій. В останні роки з'являється все більше робіт, в яких описуються принципи побудови архітектури сучасних корпоративних додатків.

А. В. Єрукаєв, О. С. Копча та Д. С. Лукенів [5] розглянули сучасний стан розробки веб-додатків за допомогою фреймворків, у тому числі Node.js. Авторами розглядається які можливості додає Node.js до базової мови JavaScript, особливості які цей фреймворк додає, такі як можливість використання файлового менеджера npm, що дає можливість зручно працювати зі встановленням файлів та фреймворків.

Механізми впровадження Node.js у розробку мобільного застосування детально розглянуто у роботі «Розробка мобільного застосування з автоматичним налаштуванням функцій клієнтської частини» [6] авторами якої Л. Є. Кузьмиченко та О. О. Арсірієм визначено основні принципи застосування та особливості розробки.

В. О. Кривенька, О. С. Городецька та Л. А. Савицька [7] запропонували використання технології віртуальних інтерфейсів для зменшення завантаженості кластеру баз даних.

У [8] М. Бартков розглянув GraalVM, За словами автора, це програмне забезпечення, яке забезпечує значне поліпшення продуктивності та ефективності програм, що ідеально підходить для створення IT-продуктів. Науковець стверджує, що GraalVM розроблений для програм, написаних на мовах Java, JavaScript, LLVM, таких як C і C++ та інших динамічних мовах. Це

усуває ізоляцію між мовами програмування та забезпечує сумісність у загальному середовищі виконання.

Питання впливу засобів розробки програмних продуктів на кінцевий результат дослідили О. М. Шинкарук, О. М. Яшина та О. Г. Онишко [9]. Основними результатами дослідження є обгрунтування вибору фреймворків для розробки програмного забезпечення. Потрібно відмітити, що не зважаючи на велику кількість фреймворків, структурований аналіз здійснений досить незначною кількістю дослідників, що і обумовлює наукову новизну даного питання. Практична значущість полягає у можливості застосування тих чи інших фрейморків відповідно до потреб замовника та розробника для досягнення певних результатів при розробці програмного забезпечення

Із зарубіжних робіт варто відзначити такі наукові праці як: Schreck, Hanna-Reetta [10], Panetta, Daniele & Steppert, Michael & Ross, Tobias & Szidat, Sönke & Cutler, Cathy & Del Guerra, A. & Düllmann, Christoph & Eberhardt, Klaus & Edelstein, Norman & Gaeggeler, Heinz & Langrock, Gert & Moenius, Thomas & Morss, Lester & Rösch, Frank & Ruehm, Werner & Trautmann, Norbert & Walther, Clemens & Wendt, Klaus & Zeh, Peter [11], Gonzalez-Morón, Dolores & Kauffman, Marcelo [12], Zverovich, Vadim [13], Beaumont, Perry [14], Sutanto, Sutanto & Gunawan, Waliadi & Faeshal, Faeshal [15], Bhardwaj, Harsh [16], Shcherbakov, E. & Shcherbakova, M [17], Mardan, Azat [18] та інші.

Проте, враховуючи описані наукові набутки, за темою, питання розкриття питання побудови архітектури сучасних корпоративних додатків на Node.js залишається відкритим та потребує детального опрацювання.

Постановка завдання. Розкрити питання побудови архітектури сучасних корпоративних додатків на Node.js.

Викладення основного матеріалу дослідження. Побудова сучасних корпоративних додатків, має на увазі, формування єдиної системи, робота якої направлена на поєднання певної кількості модулів взаємодії чи окремих підсистем. Кількість та склад модулів залежить від кінцевої мети розробки та переліку функцій покладених на корпоративну систему. При значній кількості модулів інтеграція між ними безпосередньо вимагає підтримки багатьох зовнішніх інтерфейсів. В результаті з'являються інтеграційні платформи, які закладаються в основу корпоративної інфраструктури додатків.

Node.js виступає платформою з'єднання повного переліку модулів та підсистем, яка надає можливість JavaScript взаємодіяти з пристроями вводу-виводу через свій API, мовою написання якого є C++, підключати інші зовнішні бібліотеки, написані різними мовами, забезпечуючи виклики до них із JavaScript-коду.

Однією з багатьох переваг Node.js є його архітектура, яка дозволяє легко використовувати його як виразну, функціональну мову для програмування на стороні сервера [17]. Хоча розробка на Node.js може бути тривіальною, коли розробник повністю осягає мову, для початківців, щоб ефективно використовувати Node.js, існує багато перешкод перед тим, як вони зможуть реалізувати реальні хмарні програми. Розробнику варто обовьзково враховувати, що Node.js застосовується переважно на сервері, виконуючи роль веб-сервера, але є можливість розробляти на Node.js та десктопні віконні програми (за допомогою NW.js, AppJS або Electron для Linux, Windows та macOS) і навіть програмувати мікроконтролери (наприклад, tessel, low.js та espruino).

Node.js, як функціональну одиницю, розроблено знизу вгору, щоб обробляти асинхронний запит на введення-вивід, оскільки він побудований за допомогою JavaScript, а JavaScript розроблений як цикл подій. У клієнтському JavaScript подія натискання кнопки є циклом подій. Хоча інші середовища пропонують цю функцію, вона виконується за допомогою сторонніх бібліотек або не розробляється з нуля для тієї ж мети, що й Node.js, тому вона часто працює повільно або відстає і не належить до стандартної функції. Деякі приклади включають Event Machine, який був створений для Ruby, Twisted, та був представлений для Python і доступний з Python 2 і далі, і Apache MINA, який також відомий як «Бібліотека мережевих сокетів» і є ще одним прикладом надання подій, керований та асинхронний, обмежений лише API. Таким же чином Apache AsyncWeb створюється за допомогою Apache MINA та Perl Any Event. Аналогічно, однією з переваг Node.js перед іншими буде його здатність обробляти багато запитів, одночасно діючи як клієнт сторонніх служб, керуючи лише одним потоком. Інші мови, у цьому сенсі, припинять обробку, доки віддалений сервер не відповість на їхній початковий запит, що вимагатиме багатопотокової обробки для виконання. Для порівняння, все, що буде використано у

Node, є асинхронним, тому що писати в ньому неасинхронний код буде складно. Крім того, Node.js ніколи не вимагатиме буферизації даних перед їх виведенням, тоді як інші, такі як Event Machine, вимагатимуть обов'язкової буферизації враховуючи масштабність обставин.

Будучи на стороні сервера JavaScript, ще одна помітна перевага Node.js перед іншими полягає в тому, що від розробника потрібно мати знання та досвід лише однієї мови, а саме JavaScript, незалежно від того, чи створює він сценарії на стороні клієнта чи на стороні сервера. Розробник не зобов'язаний перемикати свій мозковий цикл з однієї мови на стороні клієнта на іншу мову на стороні сервера. Крім того, оскільки Node.js молодий, він має перевагу, що уникає помилок, які робили інші мови в минулому, такі як помилка зворотної сумісності. Згідно зі статистикою, близько 47 % онлайн-серферів очікують, що сторінка завантажиться протягом 2 секунд, а затримка в 3 секунди знижує задоволеність споживачів на 16 %. Тут лідирує Node.js, оскільки його інтерпретатор менший і швидший, ніж в інших мовах, таких як PHP. На відміну від інших мов, де кожен запуск програми буде виконувати цикли, що потребують таких процесів, як налаштування завантаження, після чого підключення до бази даних, отримання важливої інформації і, нарешті, відтворення мови розмітки, тут програми на стороні сервера завжди підтримуються увімкненими. Node.js, з іншого боку, мінімізує ці кроки, залишаючи програму завжди увімкненою.

Аналізуючи сучасні дослідження [11, 14, 15], варто наголосити, що платформі в цілому бракує стандартів якості коду, і аргументація через файл, який був відредагований різними розробниками, може бути складною. Більше того, оскільки фреймворк є відносно молодим, а програмне забезпечення все ще досягає зрілості, порівняно з традиційними фреймворками, розробники стикаються з відсутністю підтримки документації і можуть зіткнутися з проблемами при отриманні підтримки від спільноти розробників, що дійсно є проблемою. Оскільки він використовує парадигму, керовану подіями/зворотним викликом, код Node.js швидко розвивається, а також його важко налагоджувати. Відсутність готового хостингу для середовищ Node.js наразі є серйозним недоліком. Складні теми в мові JavaScript, такі як успадкування прототипів, анонімні функції та зворотні виклики, ускладнюють вивчення мови, і, як наслідок, її краще вивчати після оволодіння іншою простою мовою. Node.js все ще молода мова, і, як наслідок, професійні програмісти не можуть приєднатися. Інша проблема полягає в тому, що оскільки система є однопоточною, інші запити припиняються, якщо центральний процесор зайнятий навіть на частку секунди. В результаті розробники також змушені мислити в асинхронних термінах, до яких важко адаптуватися.

На сьогодні, ще не існує єдиного стандарту, щодо архітектури програмного забезпечення Node.js чи керівництва щодо впровадження, за яким розробники мали б впливати на створення добре структурованих додатків. Проте, беззаперечним фактом є те, що організація дизайну програми з чіткими межами має вирішальне значення для успішного розгортання в реальному світі. Розробка неструктурованої моделі спричинить логічну складність і труднощі в підтримці й оновленні програми з часом. Більше того, без затверджених інструкцій для налаштування базового середовища розробки додатків потрібен значний час і зайві зусилля при розробці.

Підхід до розробки архітектури сучасного корпоративного додатку на Node.js ґрунтується на застосуванні двох окремих напрямків. Перший напрямок, це «зверху вниз» за відповідними ролями, допомагає розробникам об'єднувати та роз'єднувати модулі, дана структурна організація сформована у чіткому ієрархічному порядку. Це дозволяє розробникам спостерігати за перебігом програми від основного модуля, який відіграє роль основного класу в Java, до інших підмодулів зверху вниз. По суті, це розбивка сучасного корпоративного додатку, щоб отримати уявлення про його композиційні підмодулі. З іншого боку, підхід «розділяй і володарюй» у відповідності до виконуваних завдань, допомагає розробникам розділити завдання на простіші модулі, одночасно дозволяючи розробляти декілька модулів. Початковим підходом до розробки сучасного корпоративного додатку на Node.js є підхід «зверху вниз», а потім підхід «розділяй і володарюй» на наступному кроці, так що на початку визначаються ролі між модулями, перш ніж завдання призначаються кожній ролі, як показано на рисунку 1.

Існують різні типи хмарних додатків, які можуть скористатися перевагами запропонованої архітектури сучасного корпоративного додатку на Node.js та рекомендацій щодо впровадження. Приклади включають програми, які просто реєструють інформацію в хмарі через Інтернет, обмінюються медичною інформацією або інформацією про медичне обслуговування за допомогою мобільного додатка, а також виконують дії фізичних компонентів у локальній мережі ad-hoc.

Загалом, запропонована розробка може бути застосована до програмування та формування хмарних додатків, які вимагають основи встановлення фундаментальних вимог. Ці вимоги включають надсилання та отримання даних, наявність бази даних для зберігання інформації, виклик зовнішніх виконуваних файлів для алгоритмів машинного навчання та забезпечення простору для розширення програми.

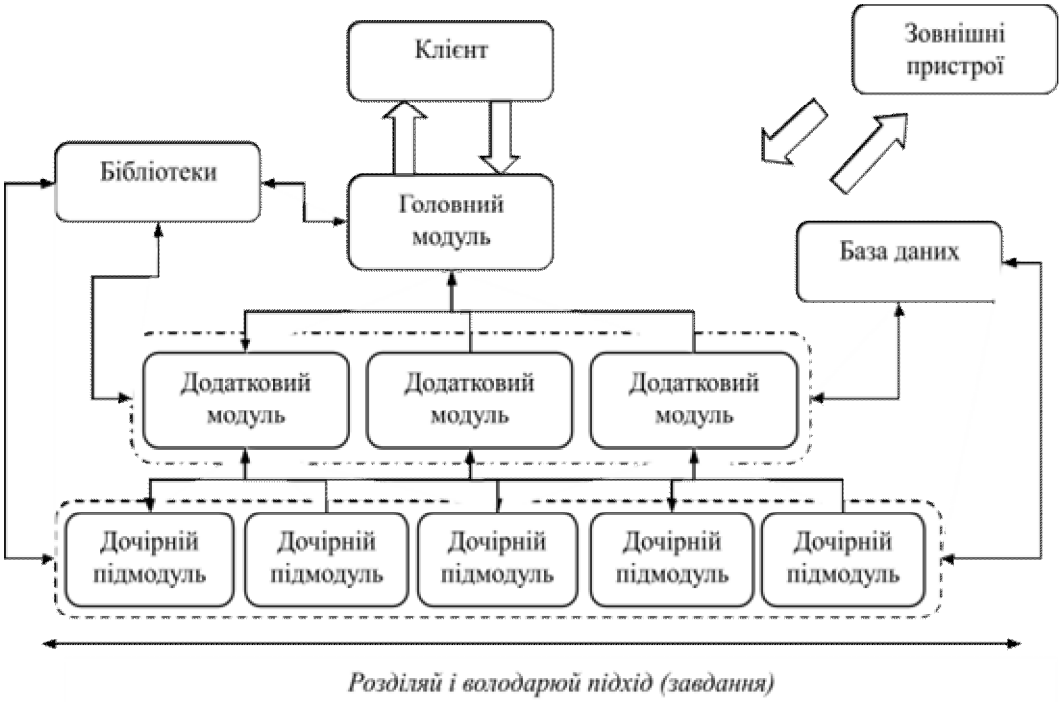

Рис. 1. **Огляд архітектури сучасного корпоративного додатку на Node.js**

Наукове підґрунтя даної роботи полягає у:

1) Архітектура сучасного корпоративного додатку на Node.js, яка вирішує проблеми підтримки та оновлення хмарних додатків, розроблених за допомогою Node.js. Цей макет дизайну підтримує отриману програму, щоб вона складалася з чітко визначених незалежних компонентів, які покращують ремонтопридатність. Крім того, нові можливості можуть бути додані до програми без серйозних змін в базовій архітектурі.

2) Інструкція з впровадження, яка містить чіткі, але прості інструкції для розробки прототипу Node.js загального призначення. На базі архітектури вказується, як: створювати прототипи за допомогою обмеженого набору API;

- розділяти модулі та організовувати функції;
- дозволяти клієнту спілкуватися з сервером;
- використовувати базу даних хмарних додатків;
- обробляти завантаженні файли.

3) Сформована архітектура програмного забезпечення та рекомендації з впровадження можуть бути основою для розробки різноманітних хмарних додатків.

Висновки і перспективи подальших досліджень. У роботі розкрито питання побудови архітектури сучасних корпоративних додатків на Node.js. Запропоновано архітектуру програмного забезпечення та рекомендації щодо впровадження для розробки хмарних додатків Node.js. Розроблена архітектура сучасних корпоративних додатків на Node.js використовує підходи «зверху-вниз», «розділяй і володарюй», щоб ефективно структурувати дизайн додатків для кращої ремонтопридатності та розширюваності з часом. З іншого боку, описані принципи та особливості впровадження, які розкривають процедури встановлення та використання. Також представлено практичні сценарії того, як розробники можуть використовувати запропоновану архітектуру сучасних корпоративних додатків та рекомендації щодо впровадження для різних типів додатків.

Перспективи подальших досліджень базуються на розширенні сфери дій пропонованої архітектури до різних типів додатків, щоб показати ефективність розробки.

Література:

1. Козак Є. Б. Програмні методи організації транспортних потоків у рамках концепції Internet of Vehicles. Науковий журнал «Комп'ютерно-інтегрована технології: освіта, наука, виробництво», 2021. № 44. С. 94-100. https://doi.org/10.36910/6775-2524-0560-2021-44-15.
2. Науменко Д. HTML, CSS, PHP, JavaScript, SQL – что и зачем?. – Режим доступу. – http://codeharmony.ru/materials/125
3. About Node.js, and why you should add Node.js to your skill set? – Режим доступу. – http://blog.training.com/2016/09/about-nodejs-and-why-you-should-add.html
4. Todd Veldhuizen «Techniques for scientific C++» – Режим доступу. – https://web.archive.org/web/20061013134847/http://osl.iu.edu/~tveldhui/papers/techniques/
5. Порівняння фреймворків express та react на платформі Node.js / А. В. Єрукаєв, О. С. Копча, Д. С. Лукенів // Тези доповідей восьмої міжнародної науково-практичної конференції «Управління розвитком технологій». Тема: Інформаційні технології розвитку змісту освіти. // Відповідальна за випуск завідувач кафедри ІТ С.В. Цюцюра, – К. : КНУБА, 2021. С. 25-26.
6. Розробка мобільного застосування з автоматичним налаштуванням функцій клієнтської частини / Л. Є. Кузьмиченко, О. О. Арсірій // Матеріали Десятої Міжнародної наукової конференції студентів та молодих вчених «Сучасні інформаційні технології – 2020» «Modern Information Technology - 2020» (14-15 травня 2020 р., м.Одеса) / МОН України; Одес. Нац. політех. ун-т ; Ін-т комп'ют. систем. – Одеса : Наука і техніка, 2020. С. 154-155.
7. Використання технології віртуальних інтерфейсів для підтримки онлайн гри / В. О. Кривенька, О. С. Городецька, Л. А. Савицька // Науково-технічна конференція факультету інформаційних технологій та комп`ютерної інженерії. ВНТУ – 2021. –
8. https://conferences.vntu.edu.ua.
9. Maxim Bartkov (2021). GRAAL AS A MULTILINGUAL PLATFORM. New York. TK Meganom LLC. Innovative Solutions in Modern Science. 3(47). doi: 10.26886/2414-634X.3(47)2021.9
10. Управління якістю програмних веб-систем засобами розробки / Шинкарук, О.М., Яшина, О.М., Онишко, О.Г. // Вісник Хмельницького національного університету, №6, 2020 (291). С. 39-44.
11. Schreck, Hanna-Reetta. (2021). Modern Corporeality. 10.4324/9780367823672-10.
12. Panetta, Daniele & Steppert, Michael & Ross, Tobias & Szidat, Sönke & Cutler, Cathy & Del Guerra, A. & Düllmann, Christoph & Eberhardt, Klaus & Edelstein, Norman & Gaeggeler, Heinz & Langrock, Gert & Moenius, Thomas & Morss, Lester & Rösch, Frank & Ruehm, Werner & Trautmann, Norbert & Walther, Clemens & Wendt, Klaus & Zeh, Peter. (2016). Modern Applications. 10.1515/9783110221862.
13. Gonzalez-Morón, Dolores & Kauffman, Marcelo. (2020). Modern applications of neurogenetics. 10.1016/B978-0-12-819178-1.00013-7.
14. Zverovich, Vadim. (2021). Modern Applications of Graph Theory. 10.1093/oso/9780198856740.001.0001.
15. Beaumont, Perry. (2019). Theoretical foundations and modern applications. 10.4324/9780429053047-4.
16. Sutanto, Sutanto & Gunawan, Waliadi & Faeshal, Faeshal. (2021). ARSITEKTUR CONTAINER DOCKER PADA APLIKASI EXPERT ASSIST DENGAN TEKNOLOGI NODE.JS, EXPRESS FRAMEWORK & CLOUD DATABASE NoSQL MONGODB ATLAS. Jurnal Sistem Informasi dan Informatika (Simika). 4. 73-89. 10.47080/simika.v4i1.1189.
17. Bhardwaj, Harsh. (2021). Challenges with Implementation of Node.Js. International Journal for Research in Applied Science and Engineering Technology. 9. 1086-1090. 10.22214/ijraset.2021.37464.
18. Shcherbakov, E. & Shcherbakova, M.. (2021). Event dispatching algorithms in Node.js. Scientific news of Dahl university. 10.33216/2222-3428-2021-20-14.
19. Mardan, Azat. (2018). Intro to Node.js: Learn Backbone.js, Node.js, and MongoDB. 10.1007/978-1-4842-3718-2_6.