УДК 004.4: 004.4 DOi 10.30837/ bi.2022.1(98).08

Ilona Revenchuk[1], Vladyslav Steshko[2]

[1] Kharkiv National University of Radio Electronics, Kharkiv, Ukraine, ilona.revenchuk@nure.ua, ORCID: 0000-0002-5188-9538

[2] Kharkiv National University of Radio Electronics, Kharkiv, Ukraine, vladyslav.steshko@nure.ua

ARCHITECTURAL SOLUTIONS AND OPTIMIZATION METHODS TO IMPROVE THE PERFORMANCE OF NODE.JS AND VUE.JS APPLICATIONS

Software architecture includes a number of important decisions about the organization of a software system, including the choice of structural elements and their interfaces that make up and unite the system into a single whole; the behavior provided by the joint work of these elements; the organization of these structural and behavioral elements into larger subsystems, as well as the architectural style that this organization adheres to. The performance of a web application is an objective measurement and user experience associated with the loading and operation of the program. Performance is about how long a web application takes to load, becomes interactive and responsive, and how smoothly the interaction with content takes place. The architecture and the steps that need to be taken to optimize the program have always been and will be relevant. Any application has its own architecture, but not every application adheres to the rules for building a good architecture, the same applies to optimization. A program with a good architecture is easier to extend and change, as well as to test, configure and understand. As practice shows, people do not like to wait, and even a three-second delay can force a user to close a tab with a slow resource. Therefore, using a number of optimization methods to improve performance will lead to increased usability and will not force the user to leave the resource due to the fact that it is running slowly. Search engines also pay attention to hundreds of parameters when ranking pages in the search. And one of the most important is the speed of data transfer from the server to the client.

ARCHITECTURE, OPTIMIZATION METHODS, PERFORMANCE, WEB SYSTEM, NODE JS, VUE

І.А. Ревенчук, В.Ю. Стешко. Архітектурні рішення і методи оптимізації для підвищення продуктивності додатків на Node.js та Vue.js. Архітектура програмного забезпечення містить у собі низку важливих рішень про організацію програмної системи, серед яких, вибір структурних елементів та їх інтерфейсів, що становлять і об›єднують систему в єдине ціле; поведінка, що забезпечується спільною роботою цих елементів; організацію цих структурних та поведінкових елементів у більші підсистеми, а також архітектурний стиль, якого дотримується ця організація. Продуктивність веб-додатку – це об›єктивні вимірювання та відчуття користувача, пов›язані із завантаженням і роботою програми. Продуктивність – це про те, як довго веб-додаток завантажується, стає інтерактивним та чуйним, про те, як плавно відбувається взаємодія з контентом. Архітектура та кроки, які необхідно зробити для оптимізації додатку, завжди були і будуть актуальними. Будь-який додаток, має свою архітектуру, але далеко не кожен додаток, дотримується правил, щодо побудови гарної архітектури, те саме стосується і оптимізації. Програму з гарною архітектурою легше розширювати та змінювати, а також тестувати, налагоджувати та розуміти. Як показує практика, люди не люблять чекати і навіть трьох секундна затримка може змусити користувача закрити вкладку з повільним ресурсом. Тому застосування низки методів оптимізації для підвищення продуктивності, призведе до підвищення зручності використання і не змусить користувача залишити ресурс через те, що він повільний. Також пошукові системи при ранжируванні сторінок у пошуку звертають увагу на сотні параметрів. І один із найважливіших – швидкість передачі даних від сервера клієнту.

АРХІТЕКТУРА, МЕТОДИ ОПТИМІЗАЦІЇ, ПРОДУКТИВНІСТЬ, ВЕБ-СИСТЕМА, NODE JS, VUE.

Introduction

At the current stage of World-wide-web development, it is important to improve the performance of a web application, scalability and reliability, by building a correct architecture and using optimization methods.

Technology selection: Node.js and Vue.js, for research, is determined by personal preferences. These technologies are also quite popular, both among companies and developers.

Behind any good product, there is a high-quality architecture and the work done is related to its optimization. Each application may, if not immediately, then after some time, encounter performance-related problems. These problems occur in the following situations:

– the app grows and becomes larger (hundreds of custom scripts, dozens of screens, etc.);

– the application begins to operate with a large amount of data;

– a large number of users appear.

All this can lead to a loss of performance, especially if the application is not optimized and, as a result, users will prefer to use a competitor's product. The use of a poor-quality application architecture negatively affects its further scaling and, as a result, the loss of understanding of the relationship between system components. Maintaining such an application becomes very difficult. Therefore, designing an architecture and paying attention to optimization is the key to a good product.

1. Basic theoretical information about architectural solutions

Software architecture is a set of approaches for organizing a software and hardware complex. Description of system components and relationships between them. Architecture includes approaches, constraints, rules, and heuristics to follow when writing code [1].

Good architecture helps to design and develop a system so that it is easier and more convenient to extend and change it. Therefore:

– if communication between modules is regulated, it is easier to replace their implementation with another one;

– if communication with the outside world is regulated, then there is less chance of data leakage;

– if the code is separated correctly, the application is easier to test;

– if the code is clearly organized, it takes less time to add new features and search for bugs in old ones;

– if the architecture is widely known, immersion in the project is faster [1].

A web app architecture presents a layout with all the software components (such as databases, applications and middleware) and how they interact with each other.

Typically a web-based application architecture comprises 3 core components (3-tier):

– client-side (Presentation layer / Client Layer) – is the key component that interacts with the user, receives the input and manages the presentation logic while controlling user interactions with the application. User inputs are validated as well, if required.

– server-side (Application Layer / Business Logic Layer) – handles the business logic and processes the user requests by routing the requests to the right component and managing the entire application operations. It can run and oversee requests from a wide variety of clients.

– database server (Data Layer) provides the required data for the application. It handles data-related tasks. In a multi-tiered architecture, database servers can manage business logic with the help of stored procedures.

It is possible to conditionally divide architectural solutions according to their goals and scope.

Some approaches distribute responsibility between modules. They determine which modules will be responsible for what. These approaches are called architectural patterns, namely: MVC, MVVM, MVP.

MVC (Model-View-Controller) — a scheme for dividing program data and control logic into three separate components: model, view, and controller — so that each component can be modified independently:

– the model provides data and responds to controller commands by changing its state;

– the view is responsible for displaying user model data, responding to model changes.

– the controller interprets user actions, notifying the model of the need for changes.

The pattern is shown in fig. 1.

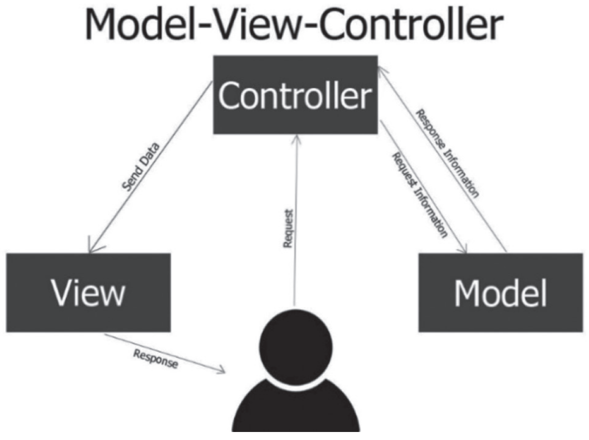

Fig. 1. MVC

MVVM (Model-View-ViewModel) — allows to separate the application logic from the visual part (View). This pattern is architectural, that is, it defines the overall architecture of the application. MVVM consists of three components: Model, ViewModel, and View. The pattern is shown in fig. 2.

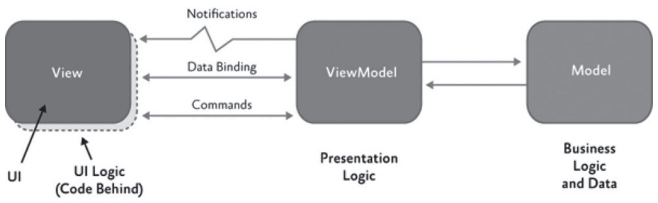

Fig. 2. MVVM

MVP (Model-View-Presenter) is a user interface development pattern. The MVP pattern is derived from MVC, but has a slightly different approach. The main difference is that presenter is not closely related to the model. The presenter takes the place of the controller. The MVP pattern is shown in fig. 3.

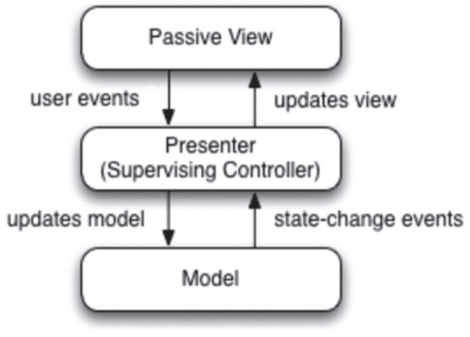

Fig. 3. MVP

The presenter takes over all the logic of data processing, updating the view, and processing user commands. The view in this case is passive: it does nothing but display data as the presenter tells it to. If in MVC the view could take over the formatting of the output, then in MVP the presenter is also responsible for this.

Others determine how close each of the modules is to business logic. For such approaches, it is important which part of the code directly deals with the application task, and which part deals with infrastructure tasks. For example, in a photo processing application, the business logic would be filter functions, and the infrastructure tasks would be calls to the phone's camera API [2].

Still others manage data flows in the app. They define how modules communicate with each other: directly, indirectly, or using special services such as the event bus. In general, data flows can be organized in a huge number of ways, but most often in practice there are two types in frontend: unidirectional and bidirectional. In a unidirectional data stream, each part of the application can receive or transmit data from another part. The direction of such flow does not change. Data in a bidirectional stream can be transmitted between parts of the application in both directions. This is most often used to link the model and view, so that updating, for example, text in the input field immediately updates data in the model — this is called bidirectional data binding. Frameworks that use bidirectional binding are often reactive — that is, they apply changes instantly not only to the UI, but also to the data being computed. Vue is one such framework.

The following approaches determine the layout of the application. It will be one large program (monolith) or a set of several smaller programs (microservices). In general, there are three types of web application architecture: monolith, microservices and serverless.

A monolith is an architectural solution in which all components and modules are closely related and depend on each other [3]. An example of the monolith architecture is shown in fig. 4.

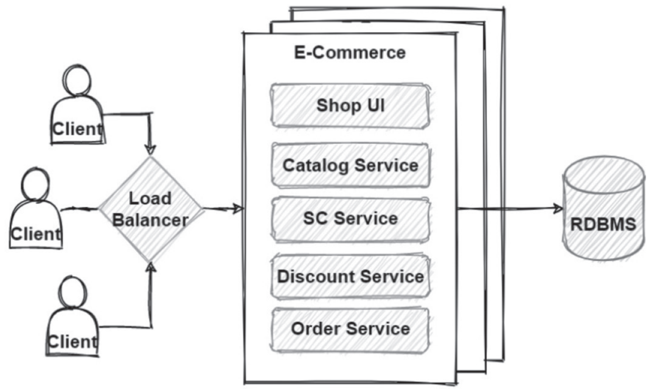

Fig. 4. Monolithic architecture

Monolithic applications work fairly efficiently, as long as they don't get too big and cause scaling problems.

Microservices is an architectural solution based on the distribution of modules into separate systems that communicate with each other using messages [3]. An example of the architecture is shown in fig. 5.

Microservices architecture solves several challenges that are encountered in a monolithic environment. In a microservice architecture, the code is developed as loosely-coupled, independent services. Each microservice contains its own database and operates a specific business logic which means you can develop and deploy independent services with ease. Since it's loosely coupled, microservice architecture provides the flexibility to update/modify and scale independent services. Development becomes easy and efficient and continuous delivery is enabled. For highly scalable and complex applications, microservices is a good choice.

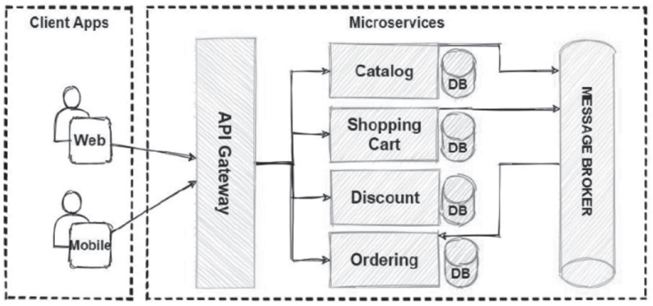

Fig. 5. Microservices architecture

Serverless is an architectural solution that focuses on development, instead of deployment and interaction between services [3]. It is an alternative to microservices that automates the entire deployment thanks to cloud technologies (fig. 6).

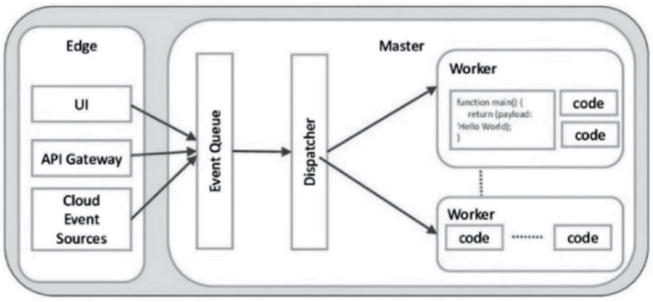

Fig. 6. Serverless architecture

Serverless computing lowers costs as resources are only used when the application is in execution. The scaling tasks are handled by the cloud provider. Moreover, backend code gets simplified. It reduces development efforts, costs and brings faster time to market.

All these architectures are top-level and each is good in its field. To implement the application, can choose any of the architectures listed above, but it can be ineffective either in terms of development or in terms of operation. Therefore, it is necessary to rely on the application requirements and on the advantages and disadvantages of each of these architectures.

2. Web application performance issues

At the moment, performance problems are among the most common. Their solution requires compliance with a number of rules formed as a result of studying the construction of web applications. Globally, web application performance issues can be divided into two categories: data transfer and runtime.

– data transfer — loading any resources necessary for the application to work;

– runtime — operation of the application, rendering and processing user input.

Each of these categories contains nuances that distinguish high-quality programs from low-quality ones. Most popular web application performance metrics:

– TTFB — time to receive the first byte;

– FCP — time until the first content display;

– FMP — time to the first significant display;

– TTI — time to start interactivity of page elements.

Among other things, can also identify common causes of performance problems:

– poorly written code can lead to many problems with web applications, including inefficient algorithms, memory leaks, and application deadlocks. Older software versions or integrated legacy systems can also reduce performance;

– non-optimized databases. An optimized database provides the highest level of security and performance, while an unoptimized database can destroy an application. Missing indexes slow down the execution of SQL queries, which can lead to a decrease in the performance of the entire application;

– DNS, firewall and network connection. DNS queries make up the majority of web traffic. That’s why a DNS problem can cause so many problems, preventing visitors from accessing your site and leading to errors, 404 errors, and wrong paths. Similarly, network connectivity and firewall performance are critical to access and performance [4];

– external services. The problem with using external services is that they are out of control;

– slow server;

– poor load distribution. This can lead to an increase in response time due to incorrect distribution of new site visitors between servers [5].

3. Problems of choosing software solutions for Node.js

Node.js is a JavaScript runtime environment based on the V8 engine that can compile JavaScript code into machine code. It is an open source cross-platform environment for developing server and network applications that uses an event-driven, non-blocking I / O model, making it simple and efficient for real-time data-intensive applications running across distributed devices.

Node.js has many frameworks, including: express.js, nest.js, koa.js, adonis.each of the presented solutions has its own characteristics and therefore it becomes difficult to choose a suitable solution.

2.1 Express.js

Express.js is a flexible and minimalistic application framework. It’s not built around specific components, which gives developers the opportunity to experiment. They get lightning-fast customization and a clean JS experience [6].

Features include: fast server-side development; NoSQL database support out of the box; supports MVC architecture; allows developers to create a RESTful API faster.

Express.js is suitable for quickly creating web applications and services, as it has API generation tools available.

2.2 Nest.js

Nest is a server platform built to support developer productivity. Nest, a framework written entirely in TypeScript (it also supports JS), is easy to test and includes everything you need.

Nest, by default, makes use of the Express library under the hood. Nest significantly extends its functionality, adds declarativeness, and also helps the developer to build the application according to best architectural practices.

Features include: easily extensible — can be used with other libraries; combines features of functional programming, OOP and functional reactive programming; provides framework APIs that help use various third-party modules available for any platform; has detailed and supported documentation;

Nest.js is used to write: clean and reusable code; code using high-level constructs: interceptors, filters, pipes, guards; scalable and tested applications. It provides the right balance of structure and flexibility to effectively manage code.

2.3 Koa.js

Koa.js is an open source web framework written by the developers of Express.js. With Koa, they aimed to create a smaller and more reliable platform for web applications and APIs. It offers a wide range of effective methods to speed up the process of creating servers. [7].

Features include: a modern and promising solution; built-in error catchall that prevents crashes; using a context object that contains request and response objects.

Used for creating servers, routes, and handling responses and errors.

2.4 Adonis.js

Adonis is a Node.js MVC framework for consistent, stable and expressive code. Adonis takes care of many of the time-consuming details of development by offering a stable ecosystem for server-side web applications. It was designed to bring joy and ease to developers in their work, so it neglects a consistent and expressive API to develop full-featured web applications. This framework follows the same principles as Laravel [8].

Features include: API and session-based authentication system; excellent security system; a powerful ORM that helps create secure SQL queries; validation of user input data.

Ideal for building RESTful APIs.

Each framework has its own capabilities and approaches. Architectural solutions and optimization methods can be applied to each of them. But the most attractive framework is Nest.js. It uses good architectural approaches and, along with optimization methods, will allow to get a scalable, structured and productive backend application.

4. Determination of optimization methods

Common methods for optimizing web applications include the following.

Data transfer

CDN — helps speed up loading for geographically distributed clients.

Resource prioritization — acceleration of page loading using the correct resource loading strategy. Browsers allow you to set priorities for different types of resources and load earlier what is important for the first drawing.

Static compression — is a compression algorithm that reduces the weight of static and, accordingly, increases the download speed.

WebP vs Png vs Jgp. WebP is a great alternative to Png. In addition to the lower weight of the images, WebP is practically the same quality and has a fast loading time [9].

TTFB. Time to receive the first byte of the application page after sending a request from the client. It is an important metric. This is a complex metric that primarily depends on what operations are performed on the server during Request processing. A long response time can be related to dozens of factors: application logic, slow database operation, routing, software platform, libraries, lack of processor power or memory.

HTTP2 can speed up page loading by multiplexing or compressing headers [10].

Runtime

requestIdleCallback is a function that allows to execute code at the end of a frame (tick) or when the user is inactive.

requestAnimationFrame allows to schedule animations correctly and maximize the chances of rendering at 60 frames per second.

DOM manipulations are expensive, and they need to be performed carefully and meaningfully. Vue, provides optimized work with the DOM due to the use of a lighter version — VirtualDOM.

Virtual scrolling and pagination. These methods are used to display a list with a large number of items. In this case, either infinite loading with virtual scrolling or pagination should be used. If you do not use this approach or use it partially, for example, only infinite loading, then performance can be significantly reduced [11].

60 FPS by pointer-events: none — with this feature you can achieve 60 FPS when scrolling the page. It works on this principle: all mouse handlers are disabled while scrolling.

Build

Webpack is a module bundler. One of the most powerful and flexible tools for building frontend. It analyzes application modules, creates a dependency graph, and then builds the modules in the correct order. It has the ability to optimize the build, which in turn improves performance.

Code splitting — by splitting the code into chunks, you can optimize the first load [12].

Minification — reducing the size of the final build by removing unnecessary characters from HTML, JS, CSS that are not needed to display the page: comments, indents [12].

Dead code — unused code is removed from the final build, thereby speeding up page loading.

If your application uses a library that needs only a few methods, then you need to extract only the methods you need when importing, not all of them. This affects the size of the final bundle and, as a result, performance.

When you download fonts locally, you need to make sure that you are using compressed font formats for modern browsers, such as WOFF and WOFF2 [13].

Lazy loading of modules / routes is a tool that is available in all popular frameworks and libraries. Allows you to “lazily” load chunks of the app’s functionality.

Chaching. Cache API, ServiceWorkers, App Shell model. Cache API is a storage for network responses that we have full control over. Service workers are specialized JavaScript assets that act as proxies between web browsers and web servers. They aim to improve reliability by providing offline access, as well as boost page performance. An indispensable aspect of service worker technology is the Cache interface, which is a caching mechanism wholly separate from the HTTP cache. App Shell architecture is one of the most efficient ways of building web apps that are loading almost instantly [14].

Lazy loading of images and videos.

Using indexes.

Application speed testing.

In addition to general methods, there are a number of those that use Vue technology:

– functional components. Let’s assume that there is a simple, small component whose task is to display a particular tag, depending on the passed value. You can optimize this component by adding the functional attribute – functional. A functional component is compiled into a simple function and has no local state.

– keep-alive is a wrapper component over the router. All components in the router will be created and destroyed when switching between routes. If the components are heavy, the interface may hang at the time of switching. Vue provides the ability not to destroy, but to cache and reuse, thereby preserving the state of the component. This optimization will lead to increased memory consumption as Vue needs to keep them alive. This approach should not be applied thoughtlessly [15].

– lazy loading of components. In a traditional component import, the child component is loaded as soon as the browser reaches the import instruction. However, if the component is large, then it makes sense to load it asynchronously. Vue provides this feature out of the box. It will only load the component when needed and display it when it is ready. When using a bundler such as Webpack, the child component will be extracted into a separate file, which will reduce the weight of the file on initial download. Lazy loading can also be applied to components used for routes. Each of the routes will be loaded when requested.

– vuex. Work with commits. Commits, unlike actions, are synchronous. If we assume that there is a need to save a large array of data, then its processing will block the interface during operation. To solve the problem, you can split the array into parts and add them one by one, giving the browser time to display. With this approach, you can add a load indicator, which will improve the UX.

– disabling reactivity. The store contains an array of objects with a high level of nesting, and Vue, according to its behavior, will perform a recursive bypass of all nested fields. If the application is built in such a way that it depends on the top-level nesting object and does not refer to reactive data several levels lower, then this reactivity can be removed, making it easier for Vue to work.

Conclusion

To create apps that people want to use that attract and retain a user, it's necessary to create a good user experience, and part of that experience is fast content loading and responsiveness to user interaction (response time). Therefore, the use of performance improvement methods is an important part of the product being developed. Well-built architecture and good performance are the key to a high-quality product.

References:

[1] Керівництво Microsoft з проектування архітектури, 2 видання. URL: https://dut.edu.ua/uploads/l_1507_99407341.pdf.

[2] *Роберт Сесіл Мартін*, Чиста архітектура – містецтво розроблення програмного забезпечення // 2019 – 368 с.

[3] Як обрати архітектуру для Web-додатку. URL: https://blog.ithillel.ua/articles/web-application-architecture.

[4] The 10 most common web app performance problems. URL: https://www.tricentis.com/blog/10-most-common-web-app-performance-problems.

[5] Web Application Performance: 7 Common Problems. URL: https://stackify.com/web-application-problems/.

[6] Архітектура JS Backend: підводні каміння, принципи роботи, лайфхаки. URL: https://dou.ua/forums/topic/33590/.

[7] Comparison of Node.js Frameworks. URL: https://inventorsoft.co/blog/top-14-node-js-frameworks-comparisson.

[8] Node.js Frameworks. URL: https://www.geeksforgeeks.org/node-js-frameworks/.

[9] Оптимізація веб-сторінок та додатків. URL: https://codeguida.com/post/189.

[10] 18 Tips for Website Performance Optimization. URL: https://www.keycdn.com/blog/website-performance-optimization.

[11] 6 Ways to speed up your Vue.js application. URL: https://betterprogramming.pub/6-ways-to-speed-up-your-vue-js-application-2673a6f1cde4.

[12] *Jeremy L. Wagner,* Web Performance in Action // 2016 – 376 p.

[13] 9 tricks to eliminate render blocking resources. URL: https://blog.logrocket.com/9-tricks-eliminate-render-blocking-resources/#load-custom-fonts-locally.

[14] Mastering browser cache. URL: https://vueschool.io/articles/vuejs-tutorials/vue-js-performance-mastering-cache/.

[15] KeepAlive, Rendering Mechanism, Performance Vue. URL: https://vuejs.org/guide/introduction.html.

The article was delivered to editorial stuff on the 05.09.2022