https://doi.org/10.35945/gb.2023.15.004

SYNTHESIS OF CONTEMPORARY APPROACHES USED IN THE DEVELOPMENT OF THE CLIENT-SIDE IN TECHNOLOGICAL PROJECTS

BESIKI TABATADZE

Associate Professor, European University, Georgia

GIORGI ASANIDZE

Master, Georgian American University, Georgia

ABSTRACT. The advancement of modern technologies and the evolution of the web have led to a multitude of technological innovations. Concurrently, web projects have become increasingly diverse and iterative. The emergence of complex systems has presented challenges that existing approaches cannot adequately address, thereby necessitating the development of novel architectures, technologies, and methodologies. Notably, adopting microservice architecture has effectively resolved issues encountered in large-scale projects and significantly enhanced their manageability and flexibility. Similarly, in response to the requirements of client-side applications, a demand for adopting a novel architectural approach has arisen.

The work presents the usefulness of using Monorepo and Micro Front-end architecture in technological projects and private web development in a single workspace. This method is suitable for the development of large-scale, complex projects. The architecture of client-side Angular and server-side NestJS technologies will be discussed in this article. Huge corporations such as Google, Facebook, and others employ these tactics. The paper clearly demonstrates why this architecture is the ideal answer for online projects and the benefits it provides developers over other existing and experienced alternatives. The work is useful both theoretically and practically.

KEYWORDS AND PHRASES: TECHNOLOGY PROJECT MANAGEMENT, PROJECT DEVELOPMENT ARCHITECTURE, WEB TECHNOLOGIES, CLIENT-SIDE TECHNOLOGIES.

INTRODUCTION

Recently, the function of programming has been expanding as software products have become more complicated and large-scale. In this instance, practical project architecture is critical. The program performs much better with a properly chosen architecture, the software code is ordered, and the error resistance is great.

It is preferable to examine and design the project architecture ahead of time before beginning a new project. In this situation, the project's complexity, scale, version control, and other elements should be defined so that future development is not hampered. After all of this, the architecture should be properly selected regardless of the project's aim.

Fortunately, numerous methods to project architecture are available today [2, 4]. It is pretty easy to distinguish between them, so we can easily select the suitable architecture for a certain project. The project's structure defines the architecture chosen, the future

development plan, the number of developers involved, and other elements that will contribute to the program's future development.

It should be remembered that as programming evolves, so does the project's architecture, which improves even more and becomes more aligned with the framework of a given direction. This enables us to start a new project with a carefully chosen programming language and an architecture tailored to it.

REVIEWING THE MONOLITH AND MICRO SERVICE ARCHITECTURE

Companies are currently deciding between two service architectures: monolithic and micro-service architecture [1, 2, 3]. The first was a monolithic architecture because the services that were still being generated at the end of the twentieth century did not have a distinct structure; the majority of the software code was included in one project, and the service was implemented in accordance with the supplied project. In fact, such a configuration implies the substance of monolithic design. Later, programming development necessitated the creation of new designs, such as microservice architecture, service-oriented architecture, and others[7]. The primary principle of microservice architecture is to break the project into portions based on logic, which facilitates control, updating, and maintenance of independent services.

Correctly picked architecture is critical to the future development of the project. If we know that the service will not be burdened with several business logic and that the modification will be minimal, we should select a monolithic design because the development process will be much easier and faster, which will help us save resources. And, if we are dealing with a huge, dynamic, multifunctional project on which numerous teams are working concurrently, it is preferable to utilise a microservice design, which ensures stability against changes. It should also be noted that this design is connected with substantially higher expenditures in terms of server infrastructure, as well as a much more complex implementation procedure.

SYNTHESIS OF MONOREPO AND MIRCO FRONTEND

Monorepo is a repository that houses all of the organisation's services, whether they are server-side or client-side. Rather than storing services independently, a single repository orchestrates all services. A developer can simply access all of their company's service code with the help of Monorepo. A developer can easily employ current software code in a task in another service using a similar approach. This strategy avoids developing the same software code many times; instead, it makes use of existing software code in the project.

To create Micro Frontend architecture [3, 5, 6], similar to the microservices architecture designed for the server, massive projects on the client side must also be compartmentalised. The creation and success of microservice architecture was critical in the evolution of this strategy. However, compared to microservices, the Micro Frontend design is significantly more complex because it is important to partition the visual component of the project in such a way that user integrity is not breached.

Although the Micro Frontend design provides many benefits to the development team and is handy for project development using rapid, flexible, and modern ways, several issues go against best practices. For example, having logic replicated in both a server-side service and a client-side application when both client-side and server-side modules utilise a corresponding project-wide common business logic model.

This issue can be adequately solved by integrating Monorepo with Micro Frontend architecture. This method enables us to join numerous services in one area, whether it's a web application or an api, and exchange codes and components that different services share. To do all of this, we must employ the NX system, which provides a synthesis of monorepo and Micro Frontend architectures, as well as shared libraries.

IMPLEMENTATION OF MICRO FRONTEND ARCHITECTURE

Considering the Micro Frontend architecture, we can add distinct features in separate web apps. With the appropriate command, this strategy can be applied in the NX environment [4].

Considering the Micro Frontend design, we have

the ability to add numerous capabilities to specific web apps. With the appropriate command, this strategy can be applied in the NX environment.

npx nx g @nrwl/angular:host main – – remotes=home,products

where **main** denotes the main project, which comprises all remote web apps, and **home** and **products** denote remote applications. When you run this command, the schematics in NX will automatically generate the relevant projects and connect them. In the case of remote apps, a module-federation.config.js file is created that contains the remote's name and project adress.

```
module.exports = {
  name: 'products',
  exposes: {
    './Module': 'apps/products/src/app/remote-entry/entry.module.ts',
  },
};
```

Figure 1. Configuration in Dependent Files.

The names of the remote applications are already specified in the module-federation.config.js file for the host, the main application.

```
module.exports = {
  name: 'main',
  remotes: ['home', 'products'],
};
```

Figure 2. Configuration in Main File.

In terms of how Micro Frontend architecture works, remote applications are hosted on multiple servers alongside the main program and are loaded as needed. If the user navigates to the Products page in the following project example, the main project will load and visualise the matching distant Products. To avoid complicating the development process, NX ensures that projects can be executed on different ports during development. The ports' configuration is provided in the project.json file of each remote project.

```
"options": {
  "port": 4201,
  "publicHost": "http://localhost:4201"
}
```

Figure 3. Port of Main Modules.

When starting the main project, which is begun after the **nx serve** command, the NX system starts the main project on the default port first, followed by the remote projects on other ports.

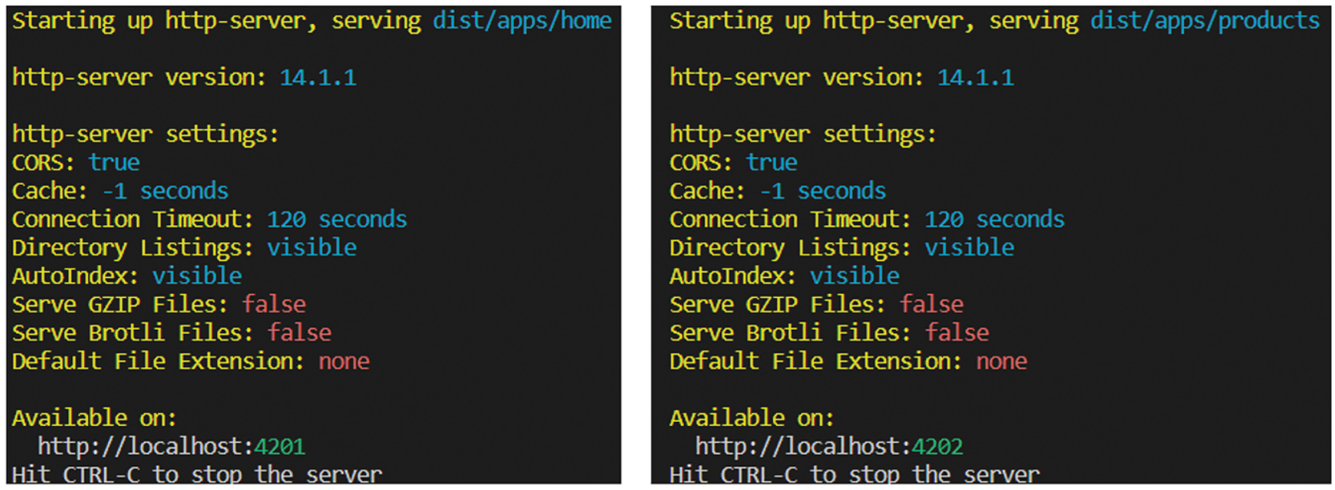

Figure 4. Ports of Dependent Modules.

These graphics demonstrate which projects are running on which ports. The remote Home project is running on port 4201, while the Products project is running on port 4202.

The web api connection in the monorepo is accomplished through the use of a proxy server; all requests made from the web application are routed through the proxy server. A proxy.conf.json file is produced in the web project that uses the web api to obtain and process information. This file contains the corresponding configuration. /api is a header used in web projects to perform api queries, e.g. /api/auth/login.

```
{
  "/api": {
    "target": "http://localhost:3333",
    "secure": false
  }
}
```

Figure 5. Rout of Main API.

CONCLUSION

The expansion of modern technology and the web space has resulted in numerous technological advancements in this field. Web projects have grown fairly broad and iterative, with the emergence of new paths that, in addition to the development of web apps, give developers and users the finest project management methodologies.

When complex systems appeared, traditional technologies and so-called frameworks could no longer do the intended tasks. Therefore, new architectures, technologies, and techniques emerged. Microservice design, for example, has easily solved challenges in large-scale projects and made their management much easier and more flexible. Similar to the Microservice architecture developed for the server-side programming language, a new architecture was required for the client-side application. As a result of all of this, Micro Frontend architecture emerged, which played a pivotal part in the development of huge projects such as Upwork, IKEA, and others.

The existing methodologies can deal with the supplied projects now, although changes in the aforementioned structures and technologies are probable in the future.

REFERENCES:

1. Lumetta, J. (2018, January 18). Monolith vs microservices: which architecture is right for your team? Retrieved from Medium: https://medium.com/free-code-camp/monolith-vs-microservices-which-architecture-is-right-for-your-team-bb840319d531
2. Lorenzo De Lauretis (2019, October 30). From Monolithic Architecture to Microservices Architecture: https://ieeexplore.ieee.org/abstract/document/8990350
3. Mezzalira, L. (2022). Building Micro-Frontends. Sebastopol: Luca Mezzalira. https://www.sciencedirect.com/science/article/pii/S0950584921000549#sec1
4. NX Docs. (n.d.). Micro Frontend Architecture. Retrieved from NX Docs: https://nx.dev/more-concepts/micro-frontend-architecture
5. Pavlenko, A. (2020, May 2). Micro-frontends: application of microservices to web front-ends. Journal of Internet Services and Information Security, pp. 49-66. https://jisis.org/wp-content/uploads/2022/11/jisis-2020-vol10-no2-04.pdf
6. Peltonen, S. (2021, August 13). Motivations, benefits, and issues for adopting Micro-Frontends. p. 2.4.
7. Johannes Thönes(2015, January). Microservices https://ieeexplore.ieee.org/abstract/document/7030212