

Distributed Systems - Challenge Task FS2023

Gripguide by Simon Linder & Davide Ferrara



Computer Science
University of Applied Sciences of Eastern Switzerland

1 Content

2	The challenge task	3
2.1	Challenge Task Requirements:	3
2.2	Requirements	3
2.3	Overview Project	3
3	Infrastructure	4
3.1	Overview Infrastructure	4
3.2	Program flow	4
3.3	Structure diagram.....	6
3.4	Workflow	7
4	Design Decisions	9
4.1	IntelliJ	9
4.2	WebStorm	9
4.3	MongoDB.....	9
4.4	Node.js.....	9
4.5	Express.....	9
4.6	TypeScript.....	9
4.7	Docker / Docker Compose.....	9
4.8	Caddy	9
4.9	GitLab	10
5	Summary.....	11
5.1	Learning Caddy	11
5.2	Docker and MongoDB	11
5.3	Conclusion	11

2 The challenge task

2.1 Challenge Task Requirements:

This was the general idea behind the challenge:

Challenge Task FS 2023

Design and implementation of a simple distributed system (of your choice), with at least one service storing data in a database, and where a service instance can fail.

2.2 Requirements

- Load balancing with scalable service
- Failover of a service instance
- Dockerized and Frontend connects via HTTP/3
- Persistent storage (storage does not need to be scalable)
- Use latest stable releases of chosen libraries and frameworks

2.3 Overview Project

So, with the requirements in mind, we wanted to create a little desktop application, which could be used to have a better platform to discuss climbing routes online and keep track which ones you have already mastered and which ones you can still try. Since we've experienced ourselves the difficulty to discuss a boulder route and how to master it during a coffee break. One problem is to get all thinking about the same route. Reconsidering how much time it's planned for the project to implement and our current limited web knowledge; we chose to have a caddy load balancer with a mongo DB persistence and an express / Node.js backend in typescript. But still we were open to try new things which we were interested in, like making different styles of the website and implementing data filtering and sorting. It should be possible to have images of the routes which can be commented by the users and uploaded as new "Boulder Routes" which you can filter based on completion of the route. As second style we made an experiment into 3D parallax where everting moves based on your cursor position, as such websites are currently state of the art for big tech companies.

3 Infrastructure

3.1 Overview Infrastructure

These are all the technologies we used:

- IntelliJ¹
- WebStorm²
- MongoDB³
- Node.js⁴
- Express⁵
- TypeScript⁶
- Docker⁷
- Caddy⁸
- GitLab⁹

3.2 Program flow

The user accesses Gripguide, which is in a Docker-Compose cluster, through their browser. Gripguide will start two node:18-alpine containers that each contains the front-end and back-end. Additionally, a Caddy load balancer container and a MongoDB container will be started to provide the Application with the ability to store data and protect against failiours.

By utilizing a Docker container, the Application can be deployed and managed easily on different systems. A web application, without worrying about the intricacies of setting up the environment manually. Docker and Git provides us with a consistent and reproducible deployment process, ensuring that the application works seamlessly across different systems.

The front-end / back-end containers work together to provide a smooth user experience. The front-end container handles the presentation layer, rendering the Gripguide interface and sending it to the user's browser (Server site rendering / expect of the 3D Style which is client site based for performance reasons). It communicates with the back-end, which contains the business logic and handles data processing and storage. (Model-view-controller Architecture)

To enhance performance and ensure high availability, a Caddy load balancer container is utilized. This load balancer distributes incoming requests across multiple instances of the Gripguide web-server container, effectively distributing the workload and preventing any single point of failure. This setup

¹ <https://www.jetbrains.com/idea/>

² <https://www.jetbrains.com/webstorm>

³ <https://www.mongodb.com/>

⁴ <https://nodejs.org/>

⁵ <https://expressjs.com/>

⁶ <https://www.typescriptlang.org/>

⁷ <https://www.docker.com/>

⁸ <https://caddyserver.com/>

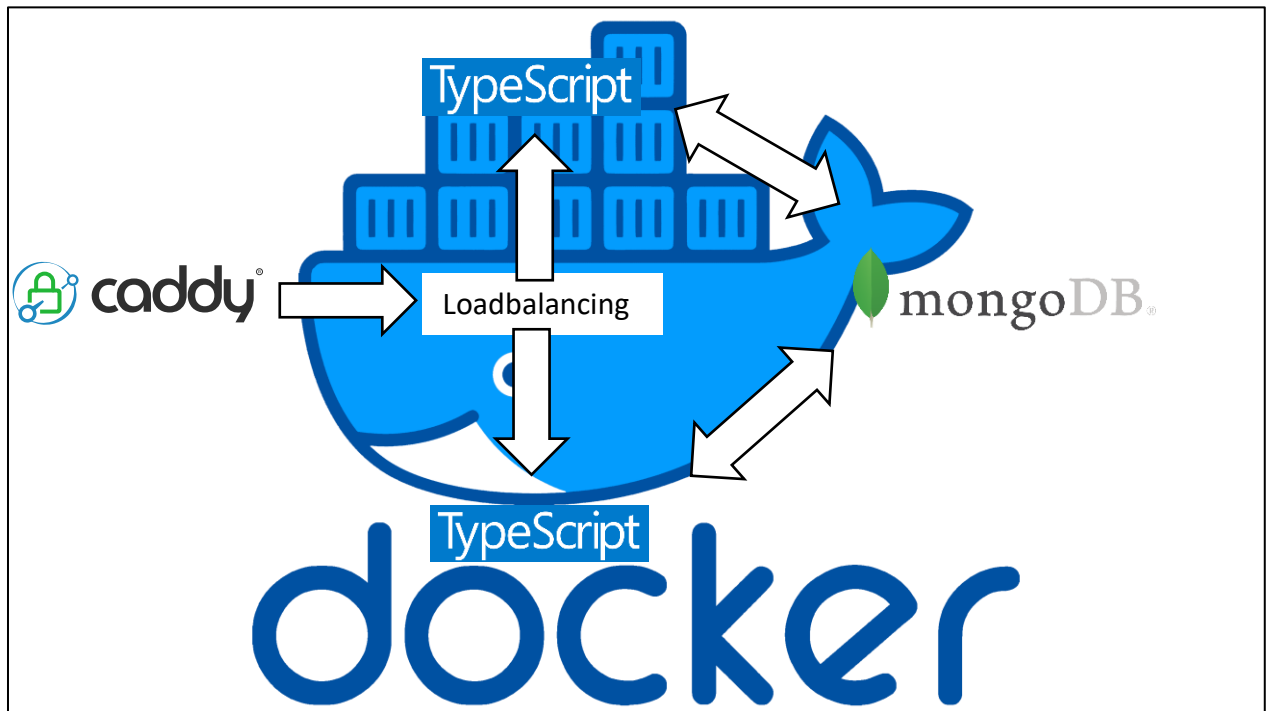
⁹ <https://gitlab.ost.ch/>

increases the application's scalability and resilience, allowing it to handle a larger number of users and maintain stability even during peak periods.

In order to persistently store user data, a MongoDB container is employed. MongoDB is a popular NoSQL database that provides flexibility and scalability. By running the MongoDB container, the user can save and retrieve data securely. This ensures that user information, such as saved items, is stored reliably, even in the event of container restarts or failures.

Overall, this Docker-based setup provides a robust and scalable infrastructure for Gripguide. It streamlines the deployment process, ensures high availability through load balancing, and guarantees data persistence using MongoDB. With this architecture, Gripguide can effectively serve users and handle their interactions, providing a seamless and reliable experience.

3.3 Structure diagram



3.4 Workflow

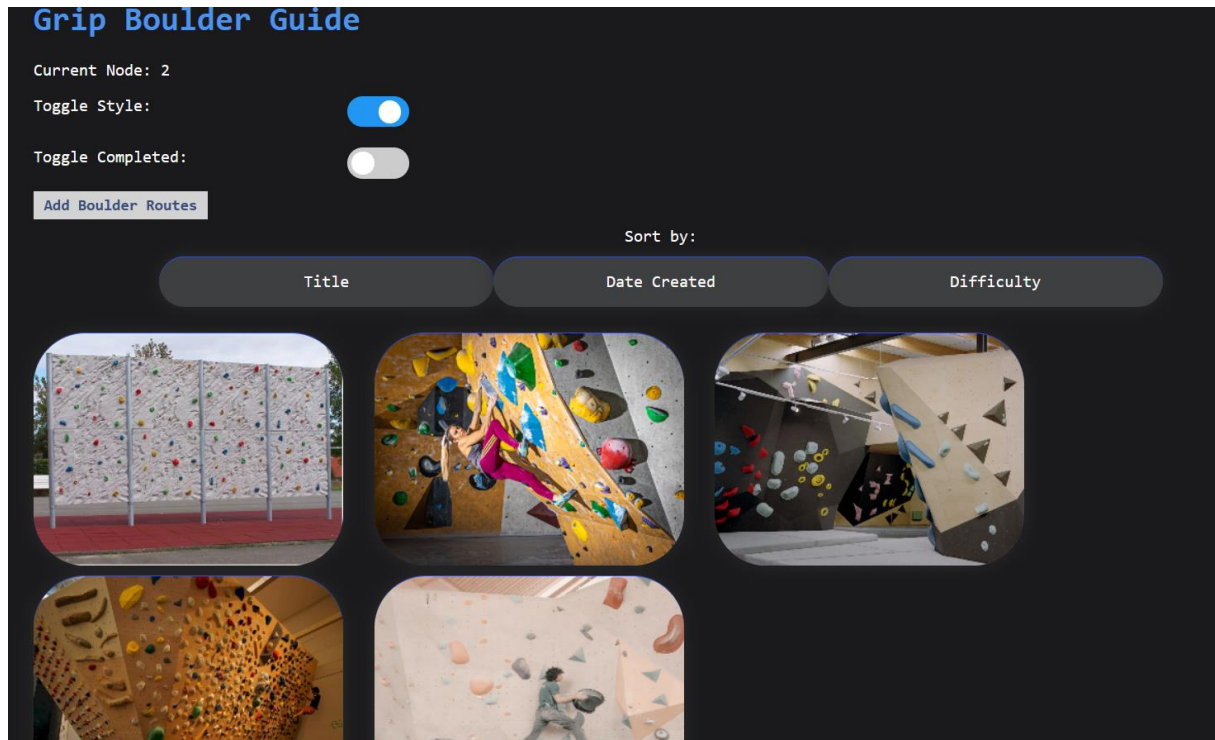


Figure 1 Dark Mode



Figure 2 3d Style

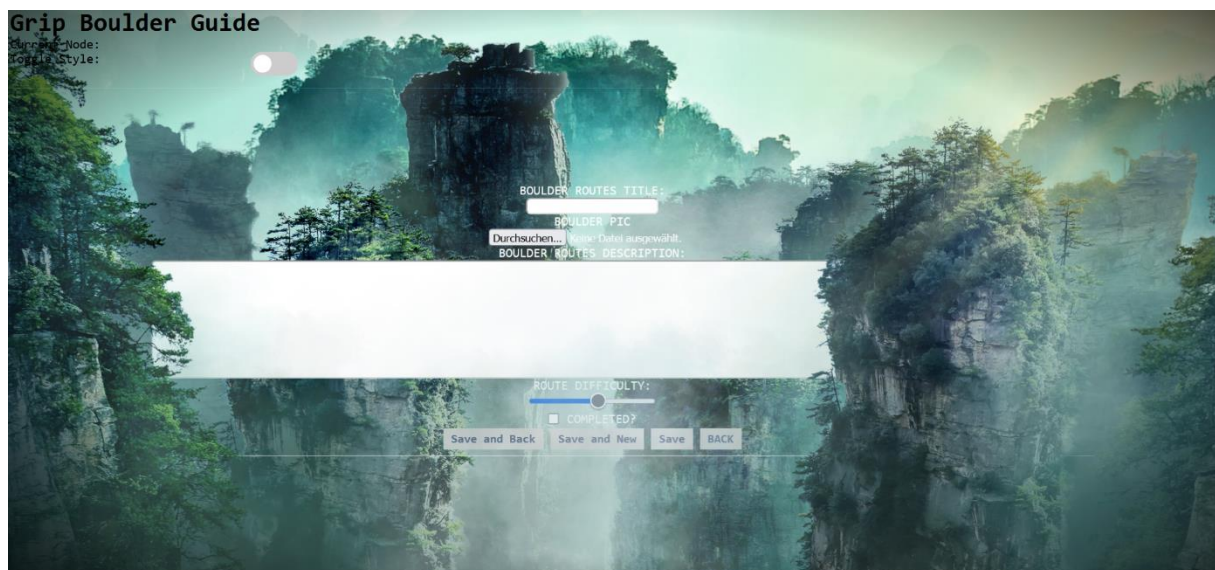


Figure 3 2. site editmode

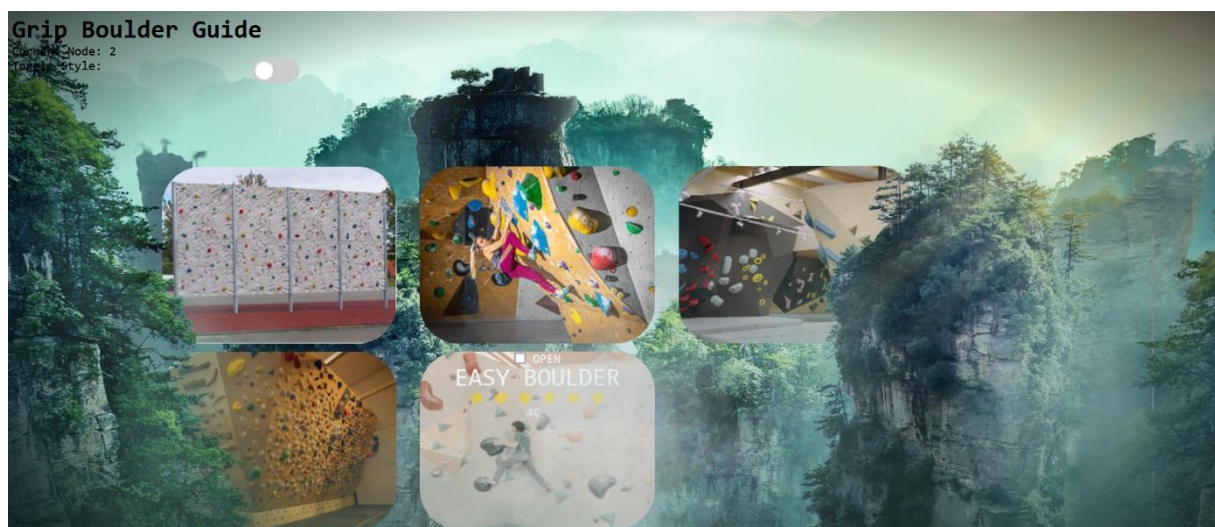


Figure 4 showing turnover of the images

4 Design Decisions

Before we started, we choose some concepts, how and with which technologies we will work.

4.1 IntelliJ

We wanted one IDE that could manage all different technologies together. IntelliJ didn't have a problem to handle Docker, TypeScript, JavaScript, Node.js, git and mongo DB so it was a good fit to finish our development.

4.2 WebStorm

It is the recommended IDE for web technologies, so it was in heavy use in the beginning of the project when creating the front/backend with Node.js, JavaScript and TypeScript.

4.3 MongoDB

We wanted to have security for data persistence. The NoSQL mongo DB was chosen because of the flexible document schemas, widely supported and code-native data access, powerful querying and analytics and we wanted to learn a new technology, since we never used MongoDB in such a project. Its simple installation and cost-effectiveness made it to a perfect fit for us. Additionally, it would easily allow us scale-out horizontally with sharing.

4.4 Node.js

Node.js is a recommended frontend / backend from the OST. We picked it because we are simultaneously learning it in the Web Engineering 2 course. It enables the App with high performance and scalability. It also enabled us to develop the website with responsive Design.

4.5 Express

Express is also a recommended backend from OST and we already knew it. These made it easier to implement it with the other technologies. Express is fast to link it with databases like MongoDB.

4.6 TypeScript

It is recommended to have a type secure programming language to write good and stable code. Nevertheless, we wanted to apply our knowledge from the visited courses from our university.

4.7 Docker / Docker Compose

Docker gave us an easy-to-use base to have a deployment tool, so we as developers can move the Application to a different place (even into a cloud) and start it with one command. It also makes it hardware independent and ensures our productivity as we don't waste time configuring each component on different development Machines. We package everything into a Docker Compose to have one package with all components needed. We have chosen alpine-18 as image for our web app since it is a light-weight image with good performance.

4.8 Caddy

We chose caddy as a load balancer because it has a straightforward approach to implement a functional failover. This feature helped us to create a valuable load balancer. We've chosen to use the load balancer with Ip-Hashing so a user can have a stable session with the server and his preferences like the style is consistent. Additionally we have chosen to use passive health checks as it is not curtail that in case of failure the user is not prompted with an error. He can simply reload the page and gets forwarded to the second node.

4.9 GitLab

We needed a version control tool and some sort of task organization. Using git during our normal semester we knew how to use it and make good use of it.

5 Summary

5.1 Learning Caddy

Firstly, to understand Caddy it would have been to our advantage to get more knowledge about what's the differences between http2 and http3. Additionally, the caching of the browser to what is on our localhost wasn't really to our pleasing. However, when it was running in the end the solution was very clear, how and why it works.

5.2 Docker and MongoDB

Though we already have worked with docker and mongo DB it wasn't our greatest challenge to implement this. Mainly, integrate the database into our express into perspective with our dockerize application got us some trouble to solve.

5.3 Conclusion

Working with a sort of distributed system will have great advantages in regard of scalability and accessibility. Irreplaceably, no modern website server or web-based application can run without a failover implementation. Especially for big infrastructure it will be necessary to have implemented a scalable network structure.

Referencing to our application we are quite happy in the way it worked out with our application and take this project as a great learning lesson for our future professional career. We saw, for the case of scalability, how to extend our program to be able to have way more services running parallel. Importantly, this project gave us a technical visualization, how in general an application with distributed would be implemented and we appreciate this experience. For our concern we are happy to take this as another completed task and as a step further in our journey as software engineers.