The Hunger Games

Ayushi Agrawal	Deepesh Bathija	Ravi Raj Pedada	Zhuqing Wang
22200035	22200199	22200547	21212716

"Together, we can conquer hunger through sharing and generosity. No man, woman, or infant should go to bed starving in a world with the means to provide for all. Let us be the modern-day Robin Hoods, taking from excess and giving to those in need."

Synopsis

This project's name is inspired by The Hunger Games franchise's title and the name of the annual event held by the Capitol. Although the Capitol's citizens were well-fed and lived a life of luxury, the rest of the Panem constantly dealt with hunger and the threat of starvation. With the rising cost of living and the post-pandemic situation, we have a similar problem in the real world. This project attempts to reduce food waste by connecting food chain franchisees with Angels on the Frontlines, Feeding the Hungry Hearts of Our Communities, aka NGOs.

We were inspired to create this project as a team because we saw a critical need in our community to address food waste and hunger. With the rising cost of living and the impact of the pandemic, we saw firsthand the struggles many people faced to put food on the table. At the same time, we also saw the abundance of food waste generated by food chain franchises. We knew there had to be a way to bridge the gap between these two issues and make a meaningful difference in our community.

The application domain for this system addresses food waste and hunger issues. The design intends to connect food chain franchises with NGOs through restful APIs published on the franchises' websites, allowing the NGOs to access information about excess food at the end of the day. This information can then be used to distribute food to those in need, reducing food waste and addressing hunger. The system will also ensure that the food provided by the franchises complies with allergen laws to ensure the safety of those receiving it.

The system is built with Spring Boot and uses a microservices architecture with RESTful APIs. It also has a MongoDB Atlas database and uses Docker Swarm for container orchestration. It is deployed on the Amazon Elastic Compute Cloud (EC2). It includes a gateway, a broker, food franchises, and branch stores. The system allows NGOs to request food information from multiple franchises and receives information from branch stores before returning a summary of available food to the NGO. The data is stored in the MongoDB Atlas [1] database for record-keeping and analysis.

Technology Stack

Our project uses a technology stack[2], that includes Spring Boot, RESTful APIs, MongoDB Atlas, Docker Swarm, and Amazon EC2.

- **Spring Boot** [3]**:** Spring Boot is a framework used in the project to build a microservices-based application. It simplifies the development process by providing tools and libraries that handle the many complexities of building and deploying microservices. Its ability to automatically configure and wire together components based on dependencies allows the team to focus on writing the application's business logic. Spring Boot also provides helpful tools and features such as an embedded HTTP server, integration with build tools, support for data access technologies, and customization options to tailor the application to specific requirements.

- **RESTful:** Each application is designed as an independent service in a microservices architecture. REST [4] is a valuable architectural style for microservices thanks to its simplicity, flexibility, and scalability.

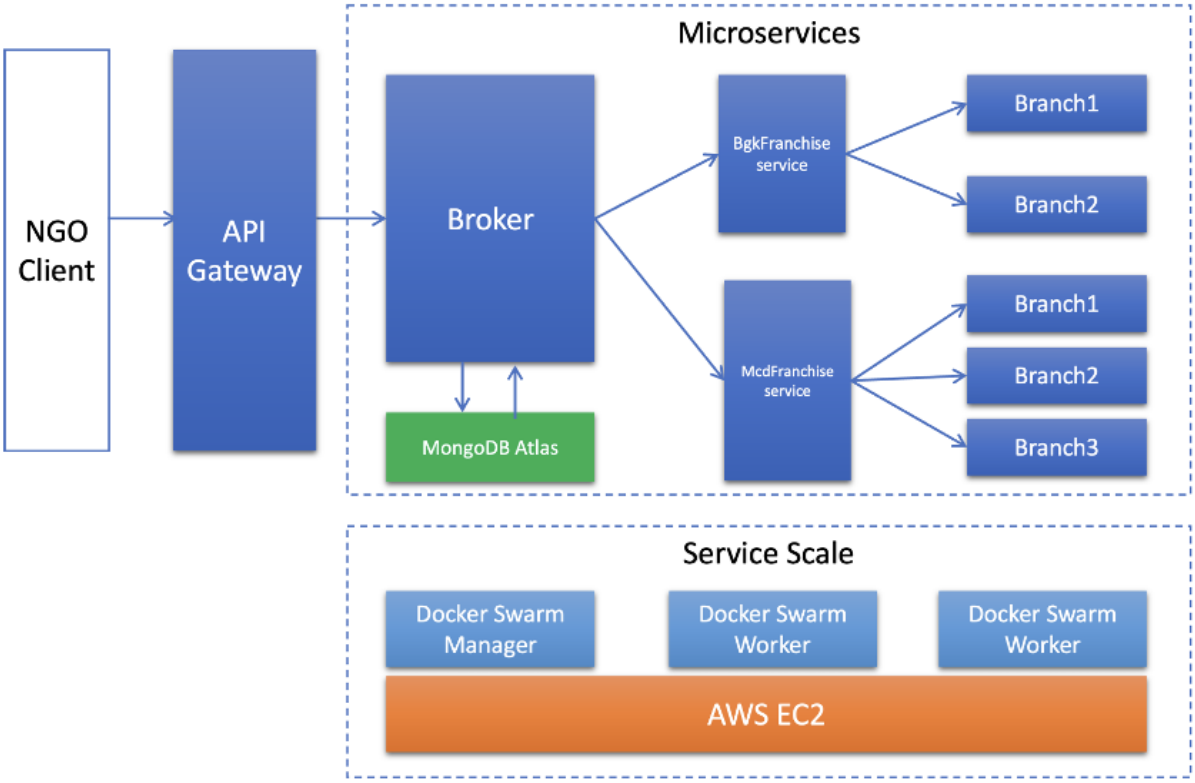
- **MongoDB Atlas:** MongoDB [5] is well suited for microservices architecture with its ability to provide a flexible schema, redundancy, automation, and scalability. MongoDB and microservices can help organisations align teams effectively, achieve faster innovation, and meet the challenges of a demanding new age in application development and delivery.

- **Docker swarm:** Inherently, microservices are distributed systems and need to be distributed and isolated resources. Docker Swarm[6] provides container orchestration clustering capabilities so that multiple Docker engines can work as a single virtual engine. It is similar to the load balancer capabilities in that it creates new instances of containers or deletes them as needed.

- **AWS EC2:** Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable computing capacity in the cloud. Users can launch instances with various OSes, load them with custom application environments, manage network access permissions, and run images on multiple systems. Since the Docker swarm needs to be applied to a specific virtual machine, we chose the Amazon EC2 [7] instance as the application server to deploy the Docker swarm cluster.

System Overview

System Architecture



The following are the main components of our system:

- **Core**: The core of the system is the central processing and functionality that powers the entire project. It provides a structure for handling tasks such as handling requests, processing data, and storing information in the database. It comprises class definitions such as AbstractQuotationService, ClientApplication and Quotation.
- **mcdTempleBar, mcdJervis, mcdStillorgan, McdFranchise:** These refer to McDonald's branch stores and franchises, respectively.
    - **McdFranchise** is responsible for connecting to the respective branches and obtaining quotations for food donations when a request is made.
    - **mcdTempleBar, mcdJervis, and mcdStillorgan** are responsible for providing information about excess food at their stores at the end of the day to the system through RESTful APIs. This information includes details such as the type and quantity of food available and the location of the franchise or branch store.
- **bgkGrafton, bgkBaggot, BgkFranchise**: These refer to Burger King branch stores and franchises, respectively.
    - **BgkFranchise** is responsible for connecting to the respective branches and obtaining quotations for food donations when a request is made.
    - **bgkGrafton and bgkBaggot** are also responsible for providing information about excess food at their stores at the end of the day to the system through RESTful APIs.

- **Broker**: The broker component is an intermediary between the NGOs and the food franchises. It receives requests for food information from NGOs and uses the RESTful APIs published on the franchises' websites to access information about excess food from multiple franchises. The broker then receives this information from the franchises, which get the information from the branch stores and returns a summary of the available food to the NGO.
- **NGO Client:** The NGO client component represents the NGOs using the system to access information about excess food and distribute it to those in need. The NGO client may send requests for food information to the broker, which will access the information from the food franchises and return a summary to the NGO. The NGO client may then use this information to plan food distribution to those in need.

Our system is on a Docker swarm cluster based on AWS EC2. When the NGO request reaches the gateway, the gateway forwards the request to the broker module, and the broker initiates a food information request to two food franchises (BgkFranchise and McdFranchise). Each franchise requests food information from its branch stores. The branch stores generate food balance information and return it to the franchise, and the franchise summarises the food information and returns it to the broker. The broker returns the food information of the two franchises to the NGO client and stores the data in the MongoDB database.

To achieve the scalability of the service, we can either configure the scale of the service in the docker-compose.yml file or use the "docker service scale <SERVICE-ID>=<NUMBER-OF-TASKS>" command line to configure the corresponding service scaled.
To take advantage of swarm mode's fault-tolerance features, we can implement an odd number of nodes according to our high-availability requirements. With multiple managers, we can recover from the failure of a manager node without downtime. Docker recommends a maximum of seven manager nodes for a swarm. Although we have not fully implemented so many manager nodes due to the need to pay for AWS resources, in future, our system can be configured with enough manager nodes to strengthen the fault-tolerance capability.

Contributions

Zhuqing configured the system's core module and deployed it to AWS instances using a Docker swarm. **Ayushi** created and maintained the Franchise microservices, which connected to its branches to get food donation quotes when needed. **Deepesh** and **Ravi** created the broker module and NGO client, which allowed NGOs to communicate with food franchises and gave NGOs access to the system to seek information about available food.

Individual Contributions

Zhuqing Wang
Zhuqing was primarily in charge of coding the system core module, configuring MongoDB Atlas, and implementing the system scaling solution. Zhuqing was also involved in the design and development of franchise modules. He finished the specification of basic data classes in the core module, designed associated interfaces, introduced related dependent packages, resolved conflicts between dependent packages, and so on. In addition, he oversaw configuring the docker-compose document and correcting Docker configuration mistakes in each module. Zhuqing set up three AWS EC2 instances and installed Docker and related tools on each one. One of the instances is set up as a manager node, while the other two are set up as worker nodes. He configured equivalent security group rules for

each instance to allow external access to system services. Git is used for pulling the project code to the manager instance, and the cloud command-line tool is used to complete the system deployment.

Ayushi
Ayushi's contribution was instrumental in developing the Burger King and McDonald's franchise microservices and the five branches of our system. These microservices are responsible for connecting to the respective branches of each franchise and obtaining quotations for food donations when a request is made. She implemented the logic for sending requests to the branches and encapsulating their responses into a single response to be returned to the customer. This was done using REST APIs, with the Franchise Service as a bridge between the customer and the food branches. I also created a unique URI for each branch that only the franchise service knew about, allowing the service to send requests to the correct branch. The Burger King franchise microservice was connected to the bgkGrafton and bgkBaggot branch stores, while the McDonald's franchise microservice was connected to the mcdTempleBar, mcdJervis, and mcdStillorgan branch stores. She was also responsible for developing and maintaining these microservices and ensuring they functioned properly within the more extensive system.

Deepesh
Deepesh's main contribution was to design and implement the functionality that allowed the broker to initiate food information requests to the food franchises and receive and process the responses. This included implementing the RESTful APIs that the broker used to communicate with the franchises and the NGO client and integrating with the MongoDB Atlas database to store and retrieve data. In addition, Deepesh contributed to developing the system's broker module and the NGO client. The broker module is responsible for managing communication between the NGOs and the food franchises, and the NGO client is the interface that NGOs use to access the system. In the broker module, Deepesh implemented the functionality for receiving NGO requests and forwarding them to the appropriate food franchises. He also handled the process of aggregating the responses from the franchises and returning them to the NGO. To facilitate this communication, Deepesh implemented RESTful APIs using Spring Boot, which allows the broker and the franchises to exchange information in a structured and standardized way. Deepesh also contributed to the implementation of the NGO client, which allows NGOs to access the system and request information about food from the franchises.

Ravi
Ravi was responsible for developing certain parts of a project. The broker module and NGO client are specific components of Ravi's project. He was involved in designing and implementing the functionality that enabled the broker to communicate with food franchises and NGOs and store and retrieve data from the MongoDB database. This suggests that the Broker module and NGO client are systems or tools that facilitate communication and data management in the project. Ravi also contributed to configuring the database and worked with another team member, Deepesh, on integrating the broker and NGO client modules. Based on this description, Ravi played a crucial role in the development of the communication and data management aspects of the project.

Reflections

One of the main challenges we encountered during this project was the need to learn and adapt to new technologies and frameworks rapidly.

We originally intended to use Akka[8] with RESTful technology for the technical architecture but later found that RESTful could not effectively interact with the Akka cluster. As a result, we had to abandon

using Akka and adopt pure RESTful to build the system. This necessitated learning about and mastering Spring Boot and microservice architecture.
In hindsight, if we start again, we recommend spending more time researching and learning about different technologies and architectures before making a final decision. This would enable us to make a more informed decision and avoid the need to switch technologies during the project.

Working on this project was a valuable learning experience for us. We gained practical experience developing microservices-based applications using Spring Boot and RESTful APIs, working with container orchestration using Docker Swarm, and cloud deployment using AWS EC2. We also learned about the importance of food waste reduction and technology's role in addressing this issue. The experience has also given us a greater appreciation for the challenges faced by NGOs and the work they do to address hunger and food insecurity in our communities.

Looking back, if we were to start this project again, we would prioritize acquiring more in-depth knowledge of technologies such as gRPC and Akka to design a more flexible and diverse microservice module. We also consider the potential limitations of technologies like RESTful and MongoDB and explore alternative solutions that better meet the needs of our system.
Despite these challenges, we are proud of what we have accomplished and our project's positive impact on our community.

Pros & Cons of the Technologies Used:

Spring Boot is a framework for building Java-based microservices and distributed systems, making stand-alone, production-grade applications accessible. It simplifies creating and configuring Spring applications and provides many starters to help manage dependencies. However, its opinionated approach may be more flexible, and the number of starters can make the application unnecessarily large. It also has a steep learning curve for those unfamiliar with Spring.

REST (Representational State Transfer) is a good fit for microservices because it allows services to communicate without requiring internal knowledge. This means that the code for each service can evolve independently without affecting other services. However, one disadvantage of RESTful APIs is that it can be challenging to maintain state, such as within sessions, and it may be harder for newer developers to use.

MongoDB is a document-based database that stores most of its data in RAM for faster query performance. In addition, it has attributes such as replication and gridFS that increase data availability. However, it does not support joins like a traditional relational database and can be tedious. It also requires a large amount of storage due to data duplication and redundancy.

Docker Swarm is a lightweight solution that is good for short development projects. It allows applications to be deployed as microservices and uses YAML files to specify multi-container configurations. It is easier to install than Kubernetes but offers less customizability. It also allows new nodes to join an existing cluster as managers or workers.

References:

[1] M. Ward, "NoSQL database in the cloud: MongoDB on AWS," *Amazon Web Services,* 2013.

[2] H. Dinh-Tuan, M. Mora-Martinez, F. Beierle, and S. R. Garzon, "Development frameworks for microservice-based applications: Evaluation and comparison," in *Proceedings of the 2020 European Symposium on Software Engineering*, 2020, pp. 12-20.

[3] M. Naily, M. R. A. Setyautami, R. Muschevici, and A. Azurat, "A framework for modelling variable microservices as software product lines," in *International Conference on Software Engineering and Formal Methods*, 2017: Springer, pp. 246-261.

[4] J. Edstrom and E. Tilevich, "Reusable and extensible fault tolerance for RESTful applications," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012: IEEE, pp. 737-744.

[5] B. Dipina Damodaran, S. Salim, and S. M. Vargese, "Performance evaluation of MySQL and MongoDB databases," *Int. J. Cybern. Inform.(IJCI),* vol. 5, 2016.

[6] N. Naik, "Performance evaluation of distributed systems in multiple clouds using docker swarm," in *2021 IEEE International Systems Conference (SysCon)*, 2021: IEEE, pp. 1-6.

[7] M. A. Azeez, "Autoscaling webservices on Amazon EC2," 2010.

[8] S. Frølund, *Coordinating distributed objects: an actor-based approach to synchronization*. MIT Press, 1996.