# PG3402 – Candidate – 2038

MiniECommerce – a microservice driven ecommerce platform.

## Choices of technology

### RabbitMQ with MassTransit

**RabbitMQ** is a widely-used message broker that facilitates asynchronous communication in a distributed system. Although we used this in class, I would like to write down some of the reasons for sticking with RabbitMQ:

* **Decoupling of Services**: RabbitMQ allows services to communicate without being tightly bound to each other, enhancing service independence and resilience
* **Reliability and Consistency**: It ensures message delivery even in cases of temporary service unavailability, preserving the consistency of operations across microservices
* **Scalability**: RabbitMQ effectively handles high throughput and can be scaled as the system grows

**MassTransit** is a lightweight service bus for building distributed applications using .NET. It acts as an abstraction layer over RabbitMQ, simplifying the development and maintenance of message-driven architectures. There was several reasons for the selection of this framework, including:

* **Ease of Use**: MassTransit simplifies the integration with RabbitMQ, providing a more intuitive API for .NET developers
* **Advanced Features**: It offers features like saga coordination, scheduling, and support for various patterns that are beneficial in complex microservice interactions
* **Better Error Handling and Monitoring**: MassTransit provides enhanced error handling and monitoring capabilities, crucial for maintaining system health

## Prometheus and Grafana

**Prometheus** is a powerful monitoring tool that collects and stores metrics as time-series data, allowing you to monitor the health and performance of your microservices.

* **Real-Time Monitoring**: Prometheus's strong querying capabilities allow real-time insight into microservice performance
* **Scalability and Reliability**: It is designed for reliability and scalability, handling large volumes of data efficiently

**Grafana** is an open-source platform for monitoring and observability and integrates seamlessly with Prometheus to provide visualizations of the collected data.

* **Data Visualization**: Grafana allows you to create comprehensive dashboards that visualize metrics from Prometheus, making it easier to understand and respond to the data
* **Alerting**: Grafana's alerting features enable proactive responses to potential issues, ensuring quick resolution of problems

## Serilog

Having opted for .NET over Java for implementation, I needed an alternative logging framework and ultimately selected Serilog for this purpose. Some reasons including:

* Structured Logging: This allows for more efficient and effective log analysis, crucial for debugging and monitoring in a microservice environment.
* Extensibility: Serilog can be extended with various sinks (outputs), enabling logs to be written to multiple destinations and formats, suitable for different analysis tools and scenarios.
* Ease of Integration: Serilog integrates well with ASP.NET Core and other .NET components, simplifying the logging process across different services.

## Docker Compose

In developing my microservice architecture, I opted to utilize Docker Compose as my tool of choice for defining and running multi-container Docker applications. This decision was driven by several key factors:

* Simplified Configuration: Docker Compose allows us to define our multi-container setup in a single, concise YAML file (docker-compose.yml). With Docker Compose, I can easily configure service dependencies, network settings, volume mounts, and more, without the need for complex scripts or commands
* Development Efficiency: It enables to launch an entire microservice stack with a single command (docker-compose up)
* Isolation and Consistency: By containerizing services, Docker Compose ensures that each service runs in an isolated environment with its dependencies
* Scalability for Testing: Docker Compose facilitates easy scaling of services for load testing in a development environment