01 Section 1 Why do we need configuration?

📚 Introduction

Configuration in any application acts as **boilerplate code** that is not part of the core logic but is crucial for the smooth functioning of the application. These configurations are typically **externalized** to maintain flexibility and scalability.

📊 What are these configurations?

- **Database Connection** – Managing connection details for different environments.
- **Feature Flags:** this is the type of configuration that we want to put it behind the flags. Saying that we are going to enable the configuration only for certain hours and see the user engagement. And if the engagement is good we are going to deploy this configuration on large scale. This is also done according to the geographical location.
- **Credentials** – Securely storing API keys, authentication credentials, and sensitive information.
- **Business logic configuration:** – Adjusting business rules dynamically, such as discounts or promotions, sometimes managed via databases or configuration files.
- **Scenario Testing (A/B Testing)** – Similar to Feature Flags, this allows testing variations of a feature by directing different traffic percentages (e.g., 10% to Version A, 90% to Version B) and scaling up the best-performing one.
- **Spring Boot Configurations** – Various configurations required for a Spring Boot application, including profiles, properties, and externalized settings.

🌟 Goals of Configuration

- **Externalization** – Keeping configurations outside the application code for flexibility.
- **Environment-Specific Configurations** – Managing settings across different environments (Development, QA, Production) while keeping the structure consistent but changing values as needed.
- **Consistency** – Ensuring configuration uniformity across microservices and deployments.

📢 Next, we will explore how to implement these configurations efficiently! 🚀