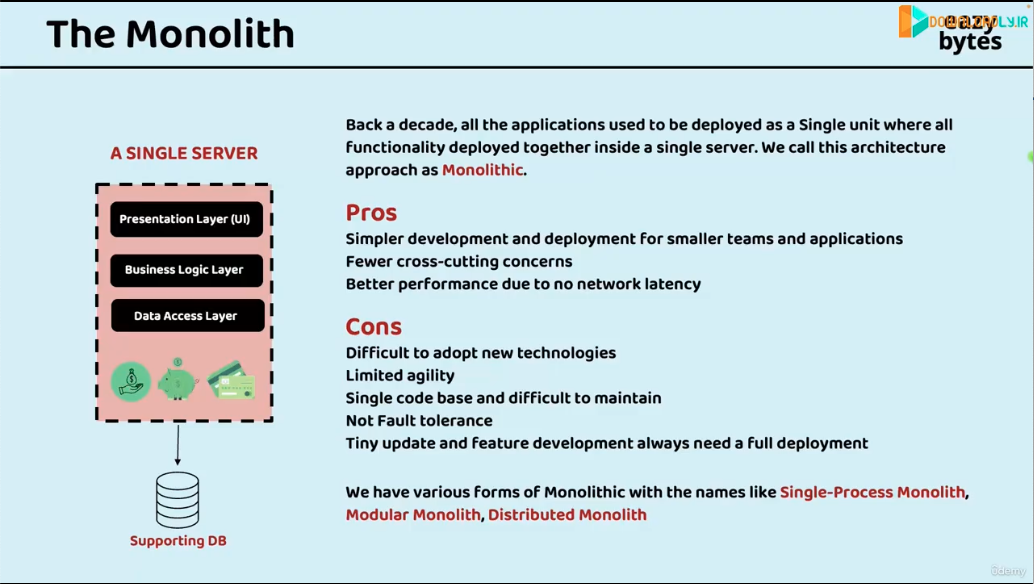
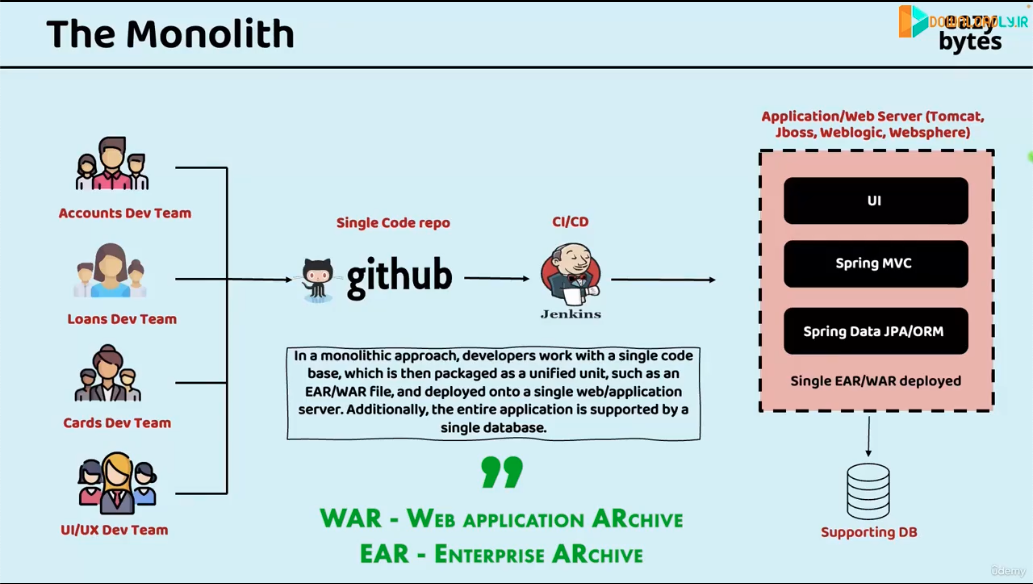
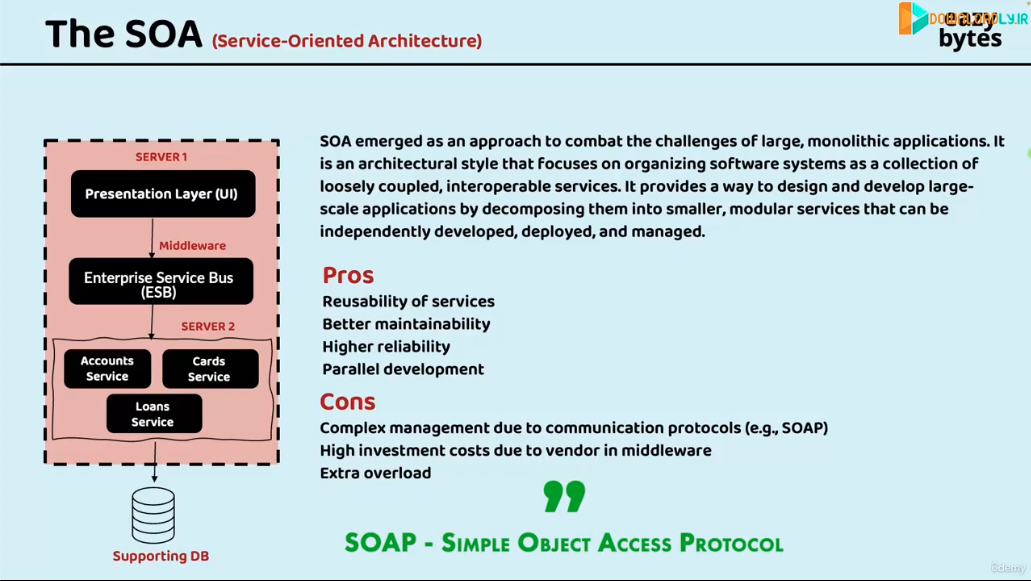
# Intro to Microservice

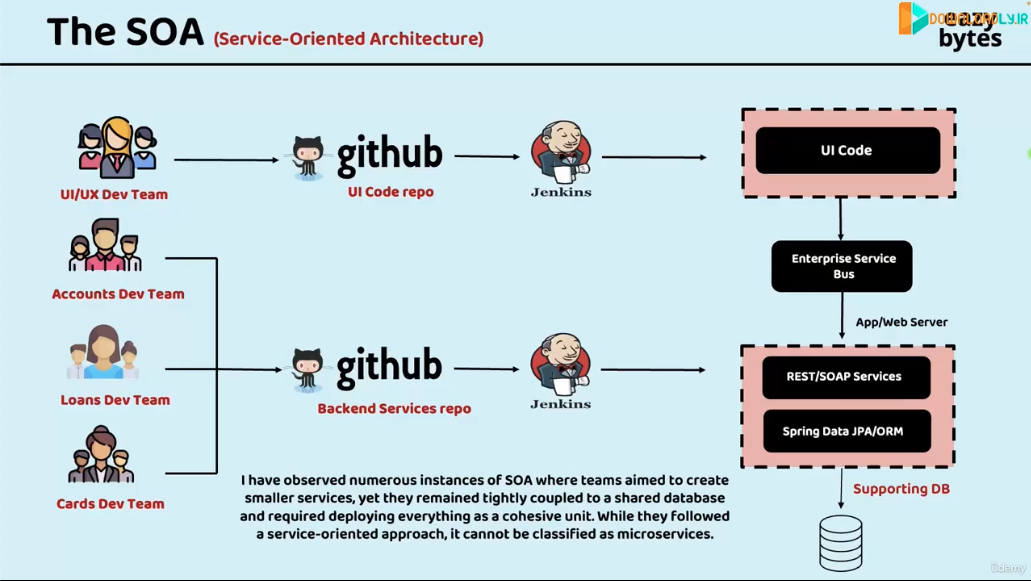
## Intro

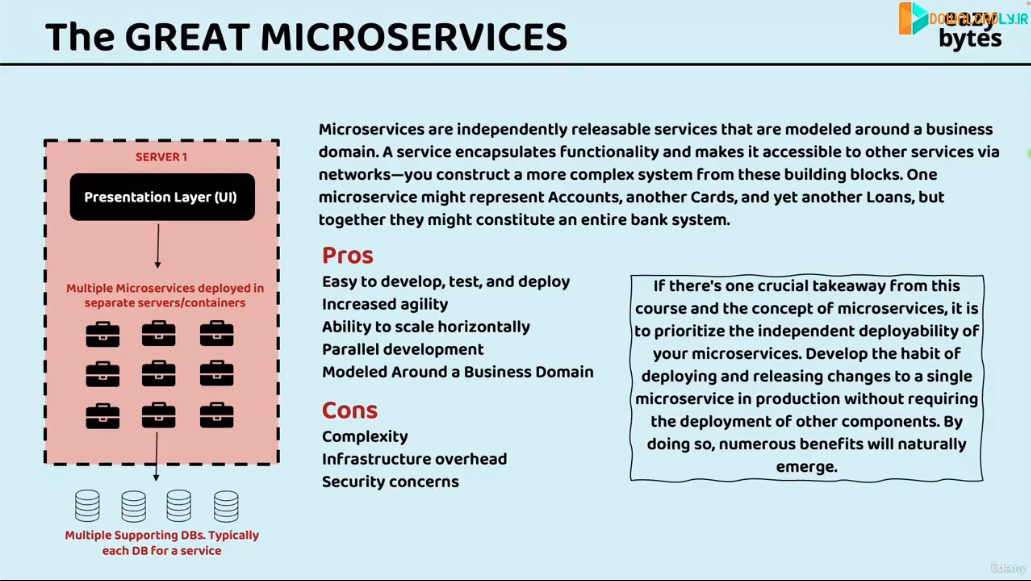


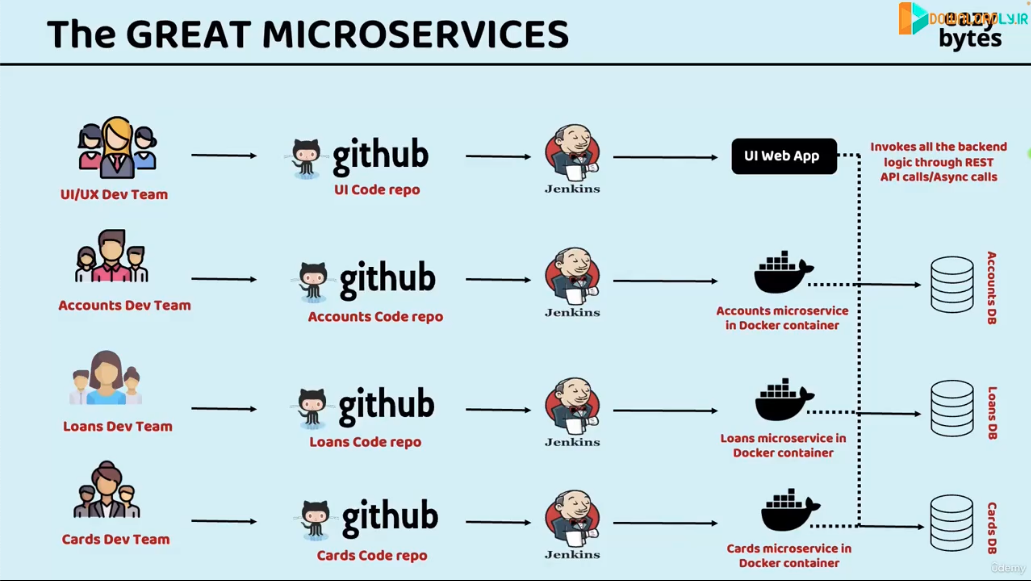




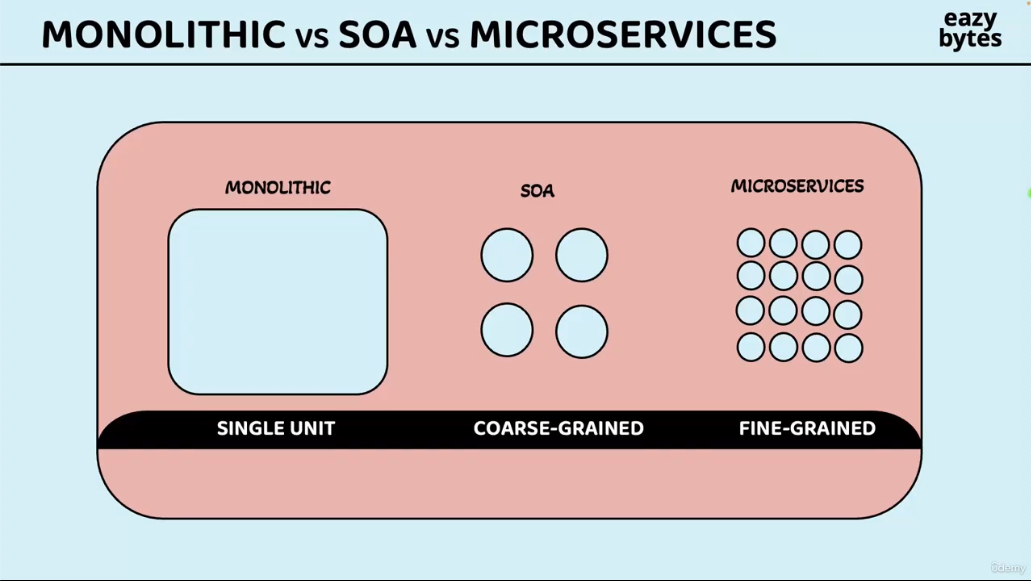
SOAP is legacy compared to REST.

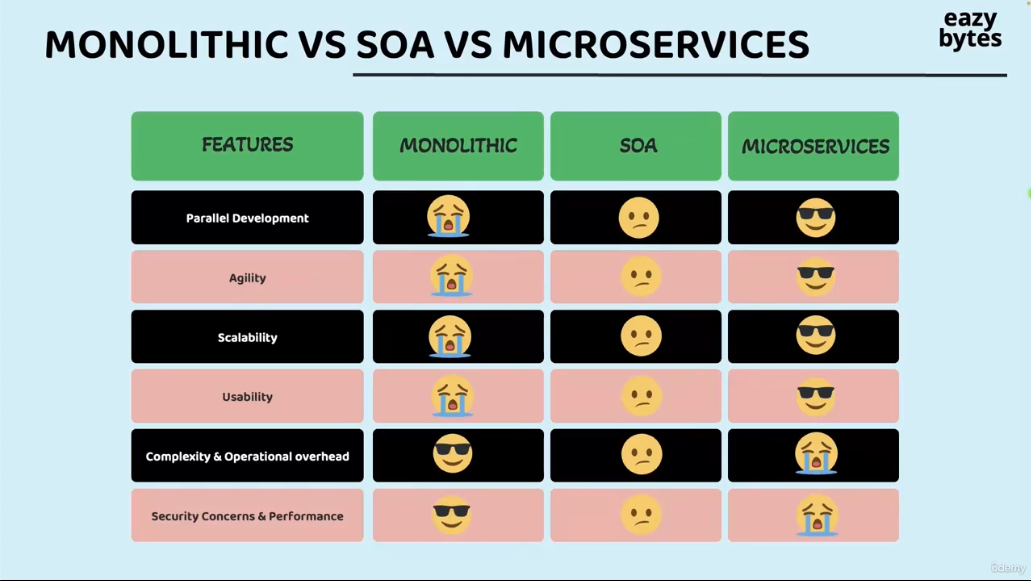




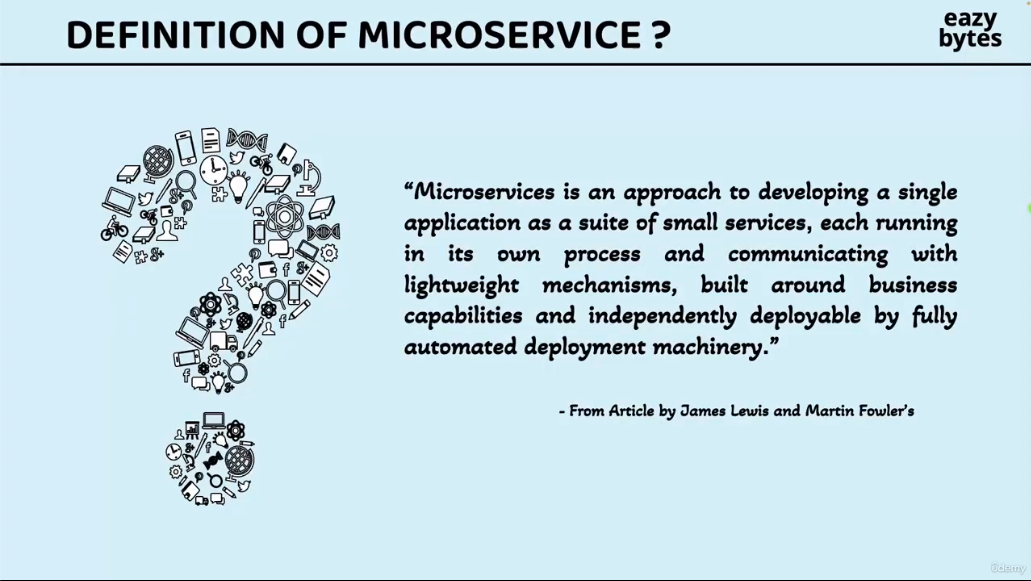


## Comparison Between Monolith vs SOA vs Microservices

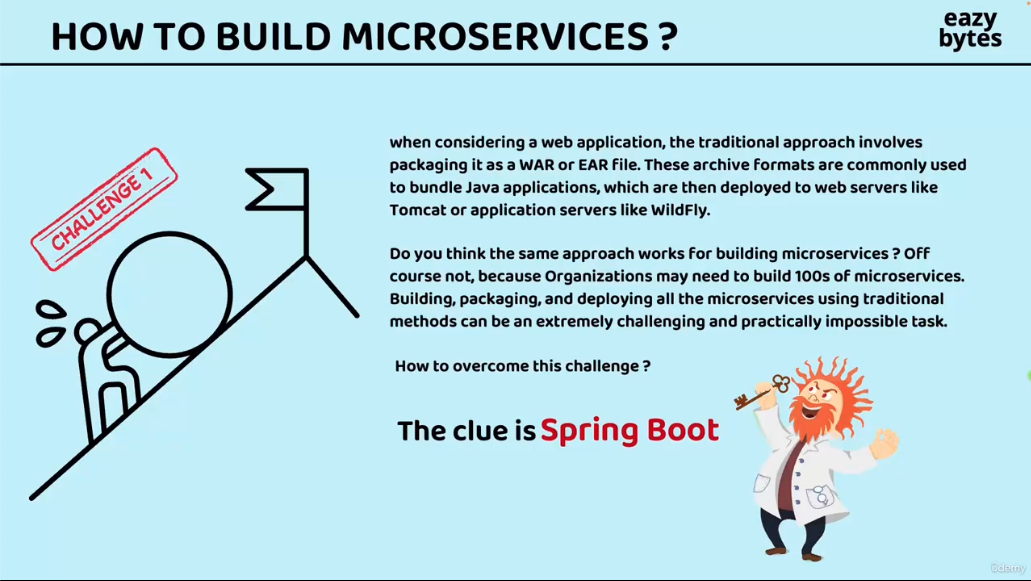


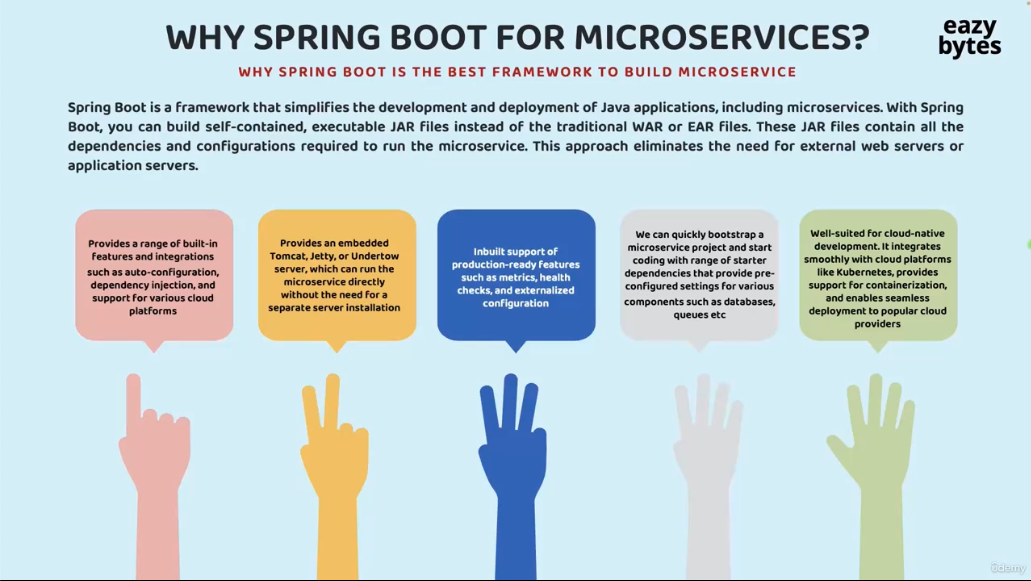


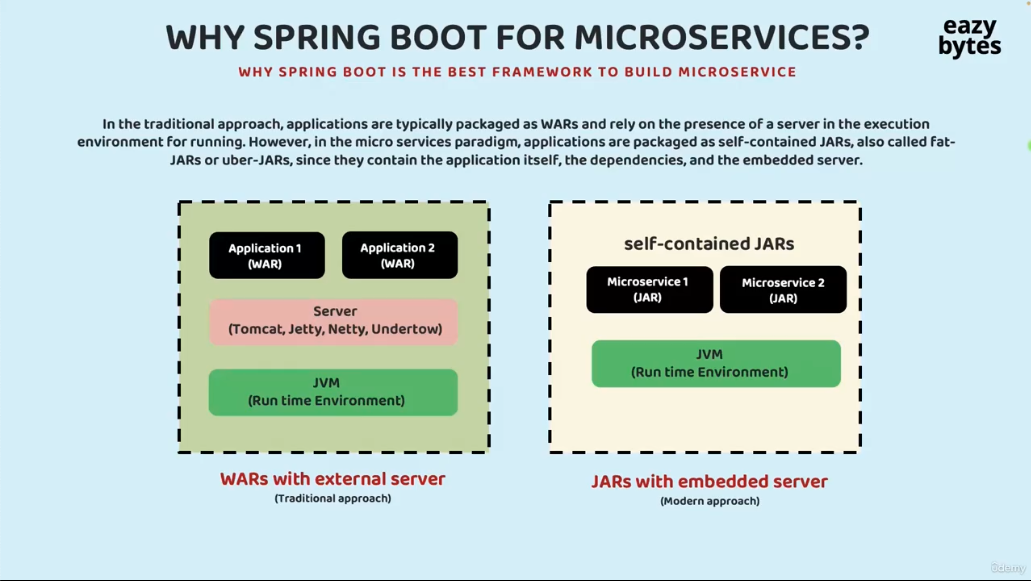
## Definition of Microservice

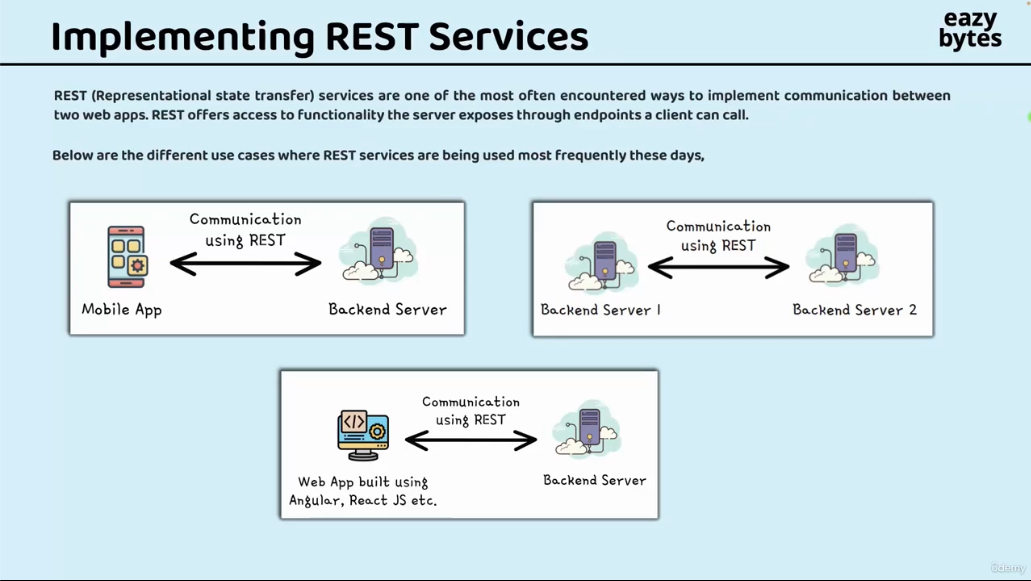


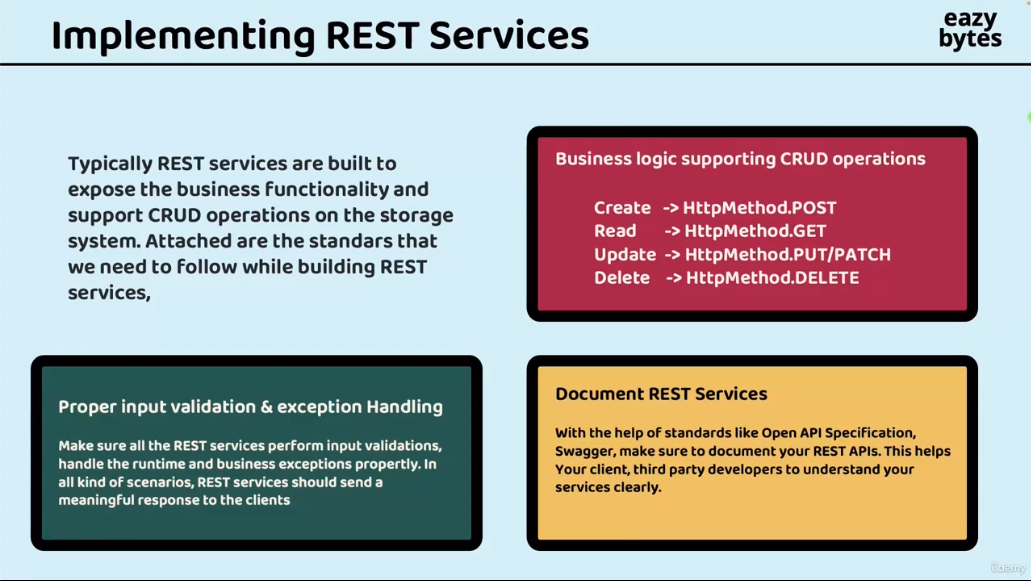
# Build Microservices using Spring Boot











## Initialize Account Microservice

Go to start.spring.io, and add dependencies:

Spring Web

H2 Database

Spring Data JPA

Spring Boot Actuator

Spring Boot DevTools

Lombok

Validation

## Config application.yml file

First we modify application.properties to application.yml. Both are valid, properties file uses key-value pairs to define app configs, yml file uses structured way to do so. We choose yml file because later Docker and Kubernetes both use yml file.

### Yml file

The configs in yml file are:

server:  
 port: 8080  
spring:  
 datasource:  
 url: jdbc:h2:mem:testdb  
 driverClassName: org.h2.Driver  
 username: sa  
 password: ''  
 h2:  
 console:  
 enabled: true  
 jpa:  
 database-platform: org.hibernate.dialect.H2Dialect  
 hibernate:  
 ddl-auto: update  
 show-sql: true

First we need to use space not tab in yml file.

**Spring datasource: database connection**

spring.datasource.url → JDBC connection string.

jdbc:h2:mem:testdb means:

h2 → using the H2 database engine.

mem:testdb → use an in-memory database named testdb (lives only while the app runs).

spring.datasource.driverClassName → the Java driver to use (org.h2.Driver).

spring.datasource.username → username for DB (default for H2 is sa).

spring.datasource.password → here it’s an empty string ('').

**h2.console.enabled**

Enables the H2 web console.

You can visit http://localhost:8080/h2-console to interact with the database in a browser.

**JPA settings**

database-platform → Hibernate dialect for H2 database. Tells Hibernate how to generate SQL for H2 (different databases have slightly different SQL syntax).

ddl-auto: update

Controls schema management (what happens to DB tables on startup):

create → drops & recreates tables each run.

create-drop → same as create, but also drops tables on shutdown.

update → updates schema without dropping existing data.

validate → just checks schema matches entities, no changes.

Here: update means Hibernate will auto-create or alter tables based on your JPA entities, while keeping existing data if possible.

show-sql: true

Logs SQL statements executed by Hibernate to the console.

Useful for debugging and learning what queries are being generated.

**What are JPA, Hibernate, JDBC?**

JPA means Java Persistence API, part of Jakarta EE. It defines how Java objects (entities) map to relational database tables.

It provides standard annotations such as @Entity, @Table, @Id, @Column.

Hibernate is a popular implementation of JPA.

It provides:

Actual code that executes JPA operations on a real database.

Extends JPA with extra features (like caching, batch fetching, lazy loading).

Automatically generates SQL from your entities and executes it.

JPA → defines entity mapping, annotations, repository interfaces.

Hibernate → generates SQL, executes it, manages DB state.

Here’s the full stack:

Your Code (User objects)

↓

Spring Data JPA (repository.save(user))

↓

JPA API (maps entity to database table)

↓

Hibernate (generates SQL from your entity)

↓

JDBC (executes SQL on the database)

↓

Database (H2, MySQL, PostgreSQL, etc.)

JDBC is the bridge between Java and the database.

Hibernate/Prisma/JPA are higher-level ORMs that save you from writing raw SQL by using objects and annotations/models.

### Create table using schema.sql

In the same directory as application.yml, create a schema.sql file and add two create table SQL statements.

Then we can log in from the browser to see the h2 database.

## Create BaseEntity, Customer, Account class and Repository

### BaseEntity

To avoid code boilerplate, we wrap createdAt, createdBy, updatedAt, updatedBy in a separate class.

Press cmd + F12 to show File Structure window.

Add MappedSuperclass to it, that tells Hibernate: “This class is not an entity by itself, but its fields should be inherited by subclasses that *are* entities.”

For createdAt and createdBy, annotate them with @Column(updatable = false), means: Hibernate will set this column when inserting a new row, but won’t touch it when updating

For updatedAt and updatedBy, annotate them with @Column(insertable = false), means: Hibernate won’t set this column when inserting a new row, but it will allow updates later

### Customer

To make Hibernate automatically generate Id, we add @GeneratedValue(strategy = GenerationType.IDENTITY) to id field.

Hibernate can map camelCase name to snake\_case name automatically so if we don’t need to specify special rules like updatable = false, we don’t need to add @Column everywhere.

### Account

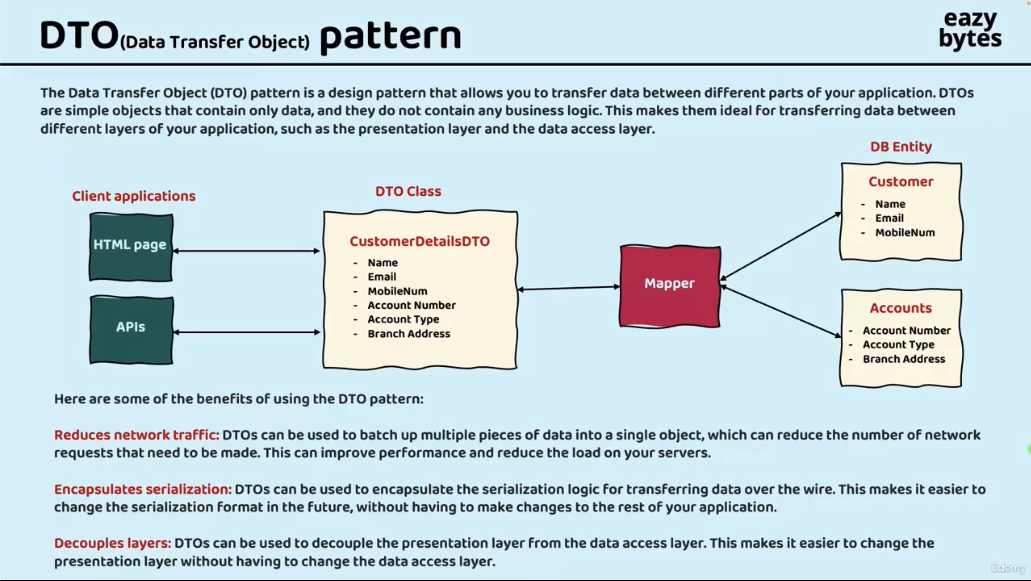
In Account class, we didn’t write auto-generate annotation to id because we will handle id generation later.

### Repository

Create two repositories representing Customer and Accounts.

Annotate them with @Repository and extends JpaRepository.

## DTO pattern



**Common layers in RESTful project**

|  |  |
| --- | --- |
| Presentation Layer | Controller  Handle HTTP requests and responses |
| Service Layer (Business Layer) | Service  Contain business logic, “what should happen” in the app |
| Data Access Layer (Persistence Layer) | JpaRepository such as CustomerRepository |
| Database Layer | H2, MySQL database |

Create four DTO classes: AccountsDTO, CustomerDTO, ErrorResponseDTO and ResponseDTO.

Annotate all DTOs with @Data, which is equivalent to:

Getter, Setter, RequiredArgsConstructor, ToString, EqualsAndHashCode, Value

We don’t use @Data annotation in entities because we don’t want the equals and hashcode method, which sometimes lead to error in Spring data JPA framework.

Only annotate latter two with @AllArgsConstructor, because in practice, developers usually prefer setters (or builders) for DTOs with many fields, and constructors for short utility DTOs.