

Spring Boot eCommerce Masterclass

Faisal Memon (EmbarkX)

Instructor: Faisal Memon

Company: EmbarkX.com

1. Personal Use Only

The materials provided in this course, including but not limited to PDF presentations, are intended for your personal use only. They are to be used solely for the purpose of learning and completing this course.

2. No Unauthorized Sharing or Distribution

You are not permitted to share, distribute, or publicly post any course materials on any websites, social media platforms, or other public forums without prior written consent from the instructor.

3. Intellectual Property

All course materials are protected by copyright laws and are the intellectual property of Faisal Memon and EmbarkX. Unauthorized use, reproduction, or distribution of these materials is strictly prohibited.

4. Reporting Violations

If you become aware of any unauthorized sharing or distribution of course materials, please report it immediately to [\[embarkxofficial@gmail.com\]](mailto:embarkxofficial@gmail.com).

5. Legal Action

We reserve the right to take legal action against individuals or entities found to be violating this usage policy.

Thank you for respecting these guidelines and helping us maintain the integrity of our course materials.

Contact Information

embarkxofficial@gmail.com

www.embarkx.com

How Does the Web Work?

Faisal Memon (EmbarkX)

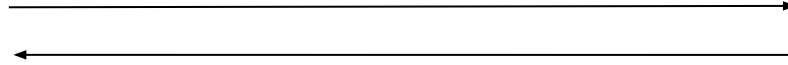
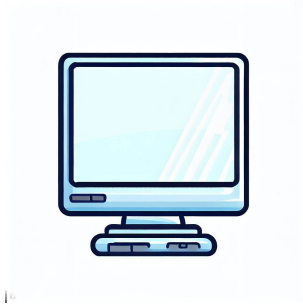
Internet

Internet is a global network of computers connected

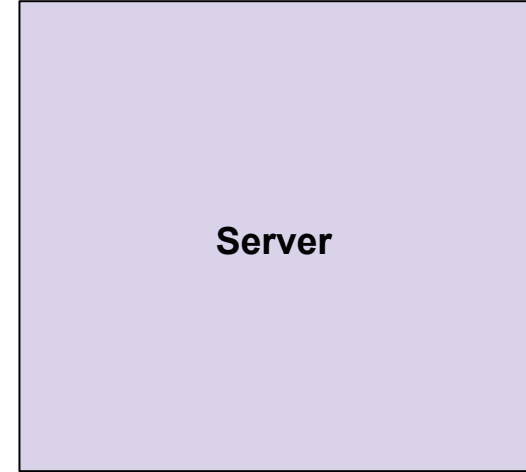
Web

World Wide Web is a way of accessing information over the medium of the Internet

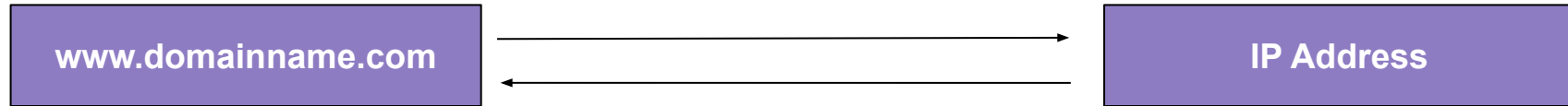
How Web Works



Request



How Web Works



Thank you

What is Client & Server?

Faisal Memon (EmbarkX)

What is a Client?



Client

→ *A device or application that requests services or resources from a server*

→ *A client is typically a web browser that users interact with to access web pages*

→ *A client can also be other types of software like an email client or a mobile app*

Characteristics of a Client



Client

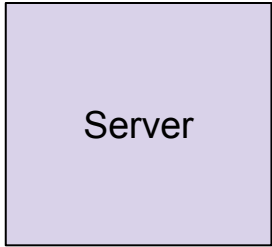
→ *User Interface*

→ *Requests Services*

→ *Receives Data*

What is a Server?

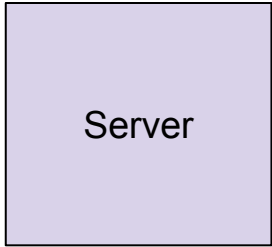
→ *A device or application that provides services or resources to clients*



→ *A Server is designed to handle requests from multiple clients*

→ *A Server hosts websites and respond to requests*

Characteristics of a Server

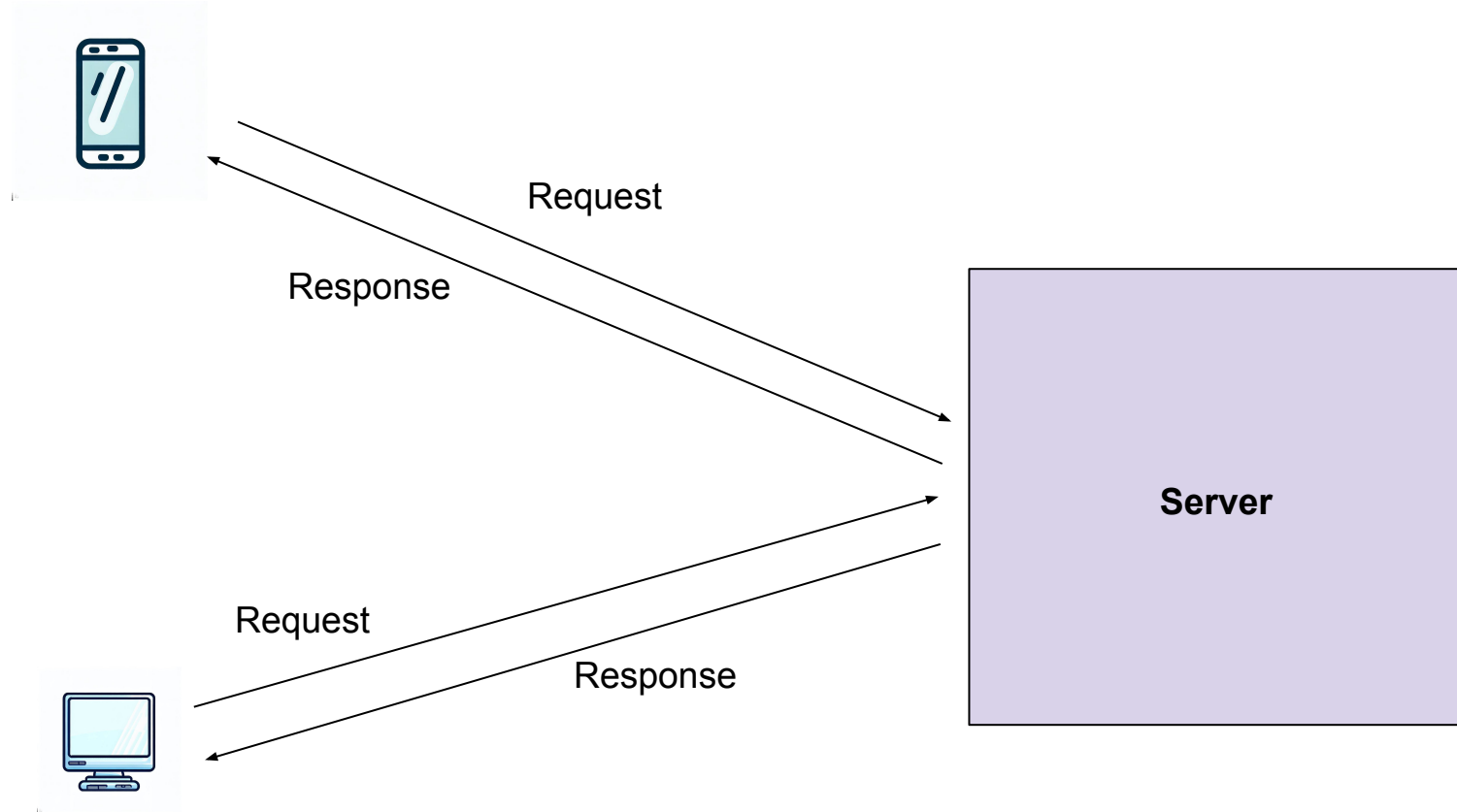


→ *Always On*

→ *Handles Multiple Requests*

→ *Sends Data*

How do they interact?



Examples

→ *Web Browsing*

→ *Email*

Thank you

What Are APIs

Faisal Memon (EmbarkX)

**API stands for Application
Programming Interface**



Imagine you
are at a
Restaurant

Restaurant

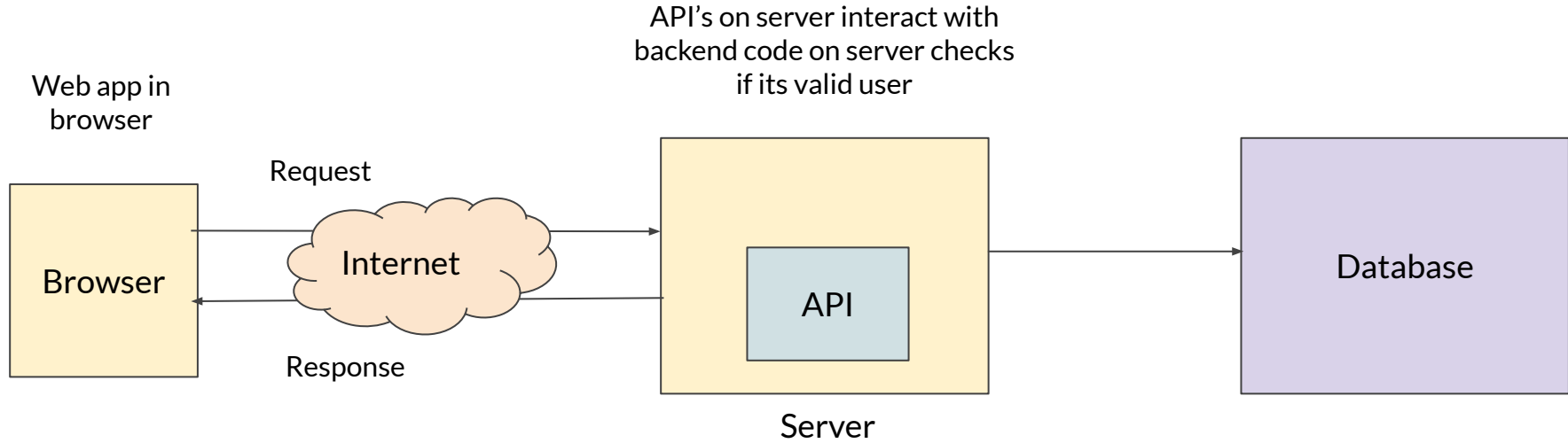
Customer → Application

Kitchen → Another System Service

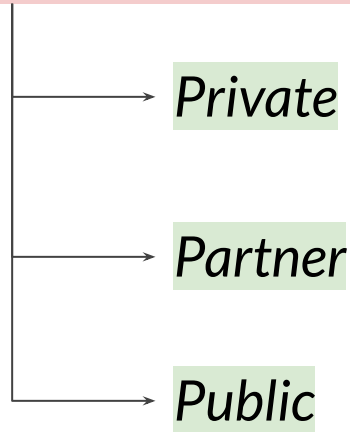
Menu → API Specifications

Waiter → API

Food → Response



API's can be



The Need

→ *Reduces manual effort*

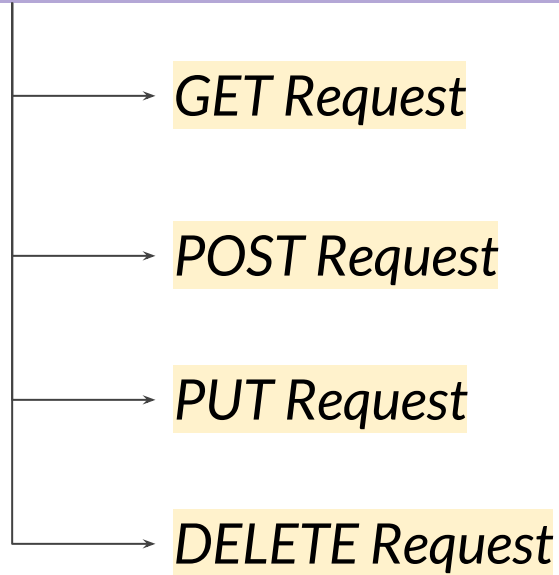
→ *Automates everything*

Thank you

Types of API Requests

Faisal Memon (EmbarkX)

Types of API requests



GET Request

- *Retrieve or GET resources from server*
- *Used only to read data*

POST Request

→ *Create resources from server*

PUT Request

→ *Update existing resources on Server*

DELETE Request

→ *Used to DELETE resources from Server*

Thank you

What is REST API and its Architecture?

Faisal Memon (EmbarkX)

REST API, stands for
**Representational State
Transfer** Application
Programming Interface

REST is **stateless**

Principles of REST API

→ *Client-Server Architecture*

→ *Stateless*

→ *Can be Cached*

→ *Opaque in terms of Layers*

→ *Uniform Interface*

Web services built following the REST architectural style are known as **RESTful web services**

Common Methods

→ *GET*

→ *POST*

→ *PUT*

→ *DELETE*

Benefits

→ *Simplicity*

→ *Scalability*

→ *Flexibility*

→ *Visibility*

Thank you

http vs https

Faisal Memon (EmbarkX)

http stands for
**HyperText Transfer
Protocol**

https stands for
**HyperText Transfer
Protocol Secure**

HTTPS is essentially
HTTP with security

Http and Https

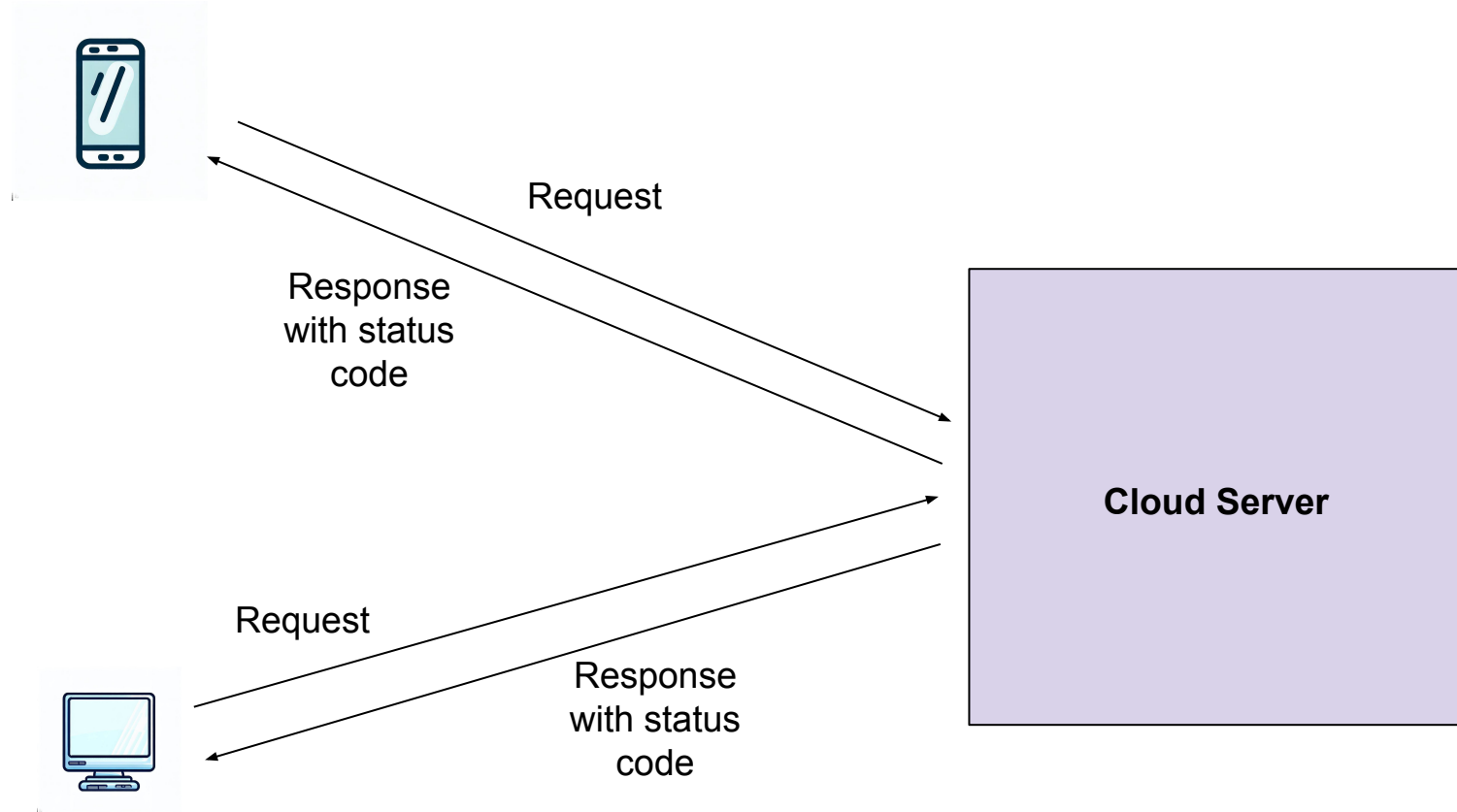
- Both HTTP and HTTPS are protocols designed for transferring hypertext across the World Wide Web.
- They operate based on a client-server model, where a client (web browser) sends a request to the server hosting a website
- Both protocols use similar methods to perform actions on the web server as well as status codes
- HTTP and HTTPS are both stateless protocols, meaning they do not inherently remember anything about the previous web session
- Both HTTP and HTTPS can transfer data in various formats including HTML, XML, JSON, and plain text

Thank You

Status Codes in API

Faisal Memon (EmbarkX)

Need for Status Codes



Classification of Status Codes

→ *1xx (Informational)*

→ *2xx (Successful)*

→ *3xx (Redirection)*

→ *4xx (Client Error)*

→ *5xx (Server Error)*

Commonly used Status Codes

→ 200 OK

→ 201 Created

→ 204 No Content

→ 301 Moved Permanently

→ 400 Bad Request

Commonly used Status Codes

- *401 Unauthorized*
- *403 Forbidden*
- *404 Not Found*
- *500 Internal Server Error*

Thank you

What is Resource, URI and Sub-Resource

Faisal Memon (EmbarkX)

Resource

→ A Resource is any piece of information that can be named or identified on the web.

→ Can represent any type of object, data, or service that can be accessed by clients

→ A resource is not just limited to documents or files; it can be anything from a text file, an image, a collection of other resources, a non-virtual object like a person, and even abstract concepts like a service

→ In a social media application, resources could include a user profile, a photo, a list of friends, or even a specific post or comment.

URI (Uniform Resource Identifier)

- A URI is a string of characters used to identify a resource on the internet either by location, name, or both
- It provides a mechanism for accessing the representation of a resource over the network, typically through specific protocols such as HTTP or HTTPS.
- URIs are a broad category that includes both URLs (Uniform Resource Locators) and URNs (Uniform Resource Names).

Sub-Resource

- A Sub-Resource is a resource that is hierarchically under another resource.
- It's a part of a larger resource and can be accessed by extending the URI of the parent resource.
- Sub-resources are often used in RESTful APIs to maintain a logical hierarchy of data and to facilitate easy access to related resources.
- Example: In a blogging platform, you might have a users resource identified by a URI (/users). A specific user could be a resource accessible at /users/{userId}.
- If each user can have blog posts, a post would be a sub-resource of that user, identified by something like /users/{userId}/posts/{postId}.

Importance in Web Development

→ *Organization*

→ *Accessibility*

→ *Scalability*

Thank you

Spring Boot eCommerce Masterclass

What is a Web Framework?

Faisal Memon (EmbarkX)

Why do you need Web Framework?

- *Websites have a lot in common*
- *Security, Databases, URLs, Authentication....more*
- *Should you do this everytime from scratch?*

Think of building a House

- *You would need Blueprint and Tools*
- *That's how web development works*
- *Developers had to build from scratch*

What if...

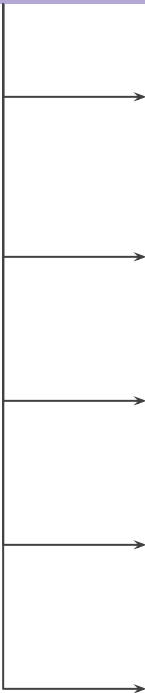
- *You could have prefabricated components?*
- *Could you assemble faster?*
- *Could you reduce errors?*
- *Would that make you fast?*

This is what a **Web
Framework** does!

What is Web Framework

Web Framework is nothing but collection of tools and modules that is needed to do standard tasks across every web application.

Popular Web Frameworks



Spring Boot (Java)

Django (Python)

Flask (Python)

Express (JavaScript)

Ruby on Rails (Ruby)

Thank you

Introduction to Spring Framework

Faisal Memon (EmbarkX)

History

- *Initially developed by Rod Johnson in 2002*
- *First version released in March 2004*
- *Since then, major developments and versions released*

Spring simplifies enterprise
application development

Key Principles



Simplicity

Modularity

Testability

Key Components of Spring

→ *Core Spring Framework*

→ *Spring Boot*

→ *Spring Data*

→ *Spring Security*

→ *Spring Cloud*

Use Cases

→ *Enterprise Applications*

→ *Microservices Architecture*

→ *Web Applications*

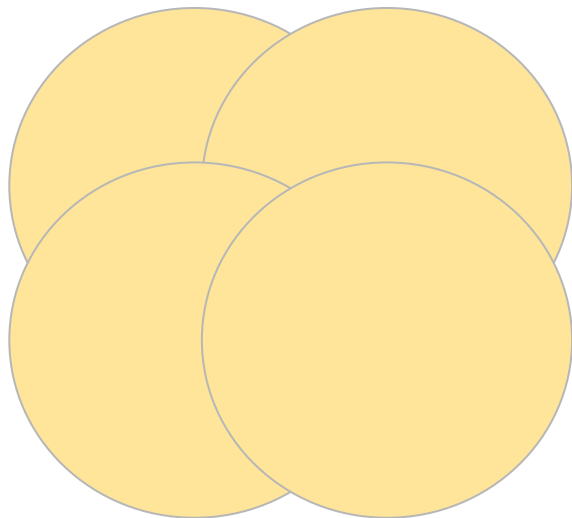
Thank you

Tight Coupling and Loose Coupling

Faisal Memon (EmbarkX)

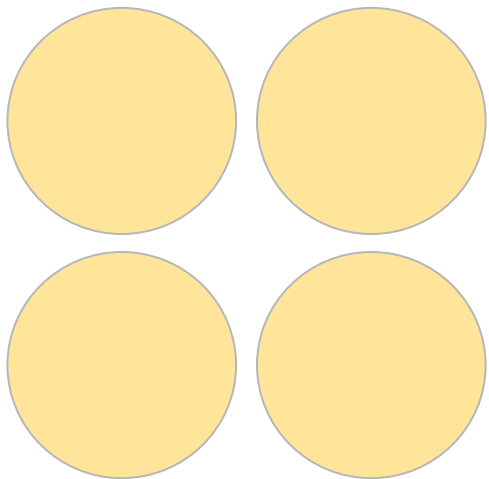
Coupling refers to how closely connected different components or systems are

Tight Coupling



Tight coupling describes a scenario where software components are **highly dependent** on each other

Loose Coupling



Loose coupling describes a scenario where software components are **less dependent** on each other

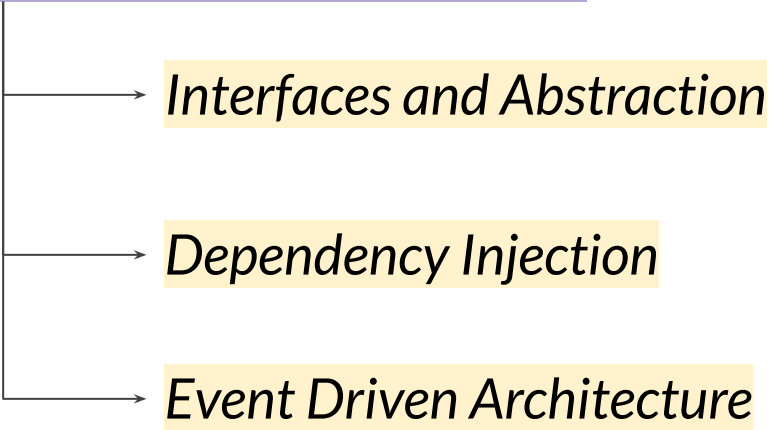
Importance in Software Design

→ *Flexibility and Maintainability*

→ *Scalability*

→ *Testing*

Achieving Loose Coupling



```
graph LR; A[Achieving Loose Coupling] --> B[Interfaces and Abstraction]; A --> C[Dependency Injection]; A --> D[Event Driven Architecture];
```

Interfaces and Abstraction

Dependency Injection

Event Driven Architecture

Thank you

Core Concepts of Spring

Faisal Memon (EmbarkX)

Loose Coupling

Loose Coupling is a design principle that aims to reduce the dependencies between components within a system

Inversion of Control (IoC)

Inversion of Control is a design principle where the control of object creation and lifecycle management is transferred from the application code to an external container or framework

Dependency Injection [DI]

Dependency injection is a design pattern commonly used in object-oriented programming, where the dependencies of a class are provided externally rather than being created within the class itself

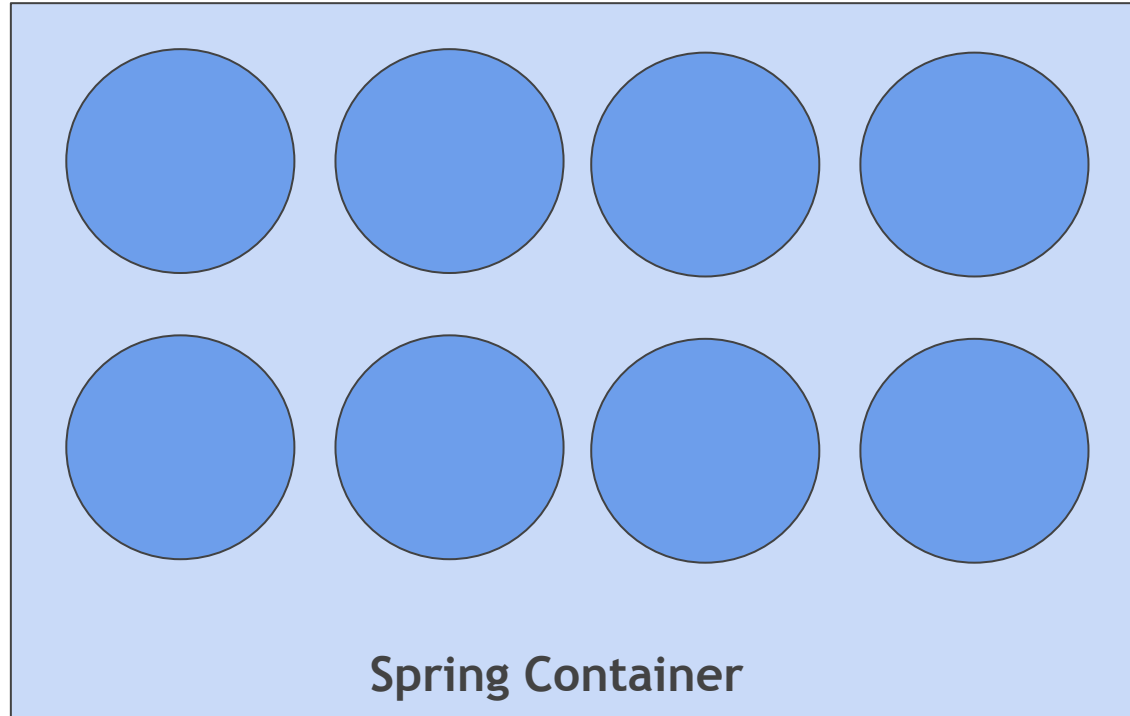
Beans

Objects that are managed by frameworks are known as Beans

Thank you

Spring Container and Configuration

Faisal Memon (EmbarkX)



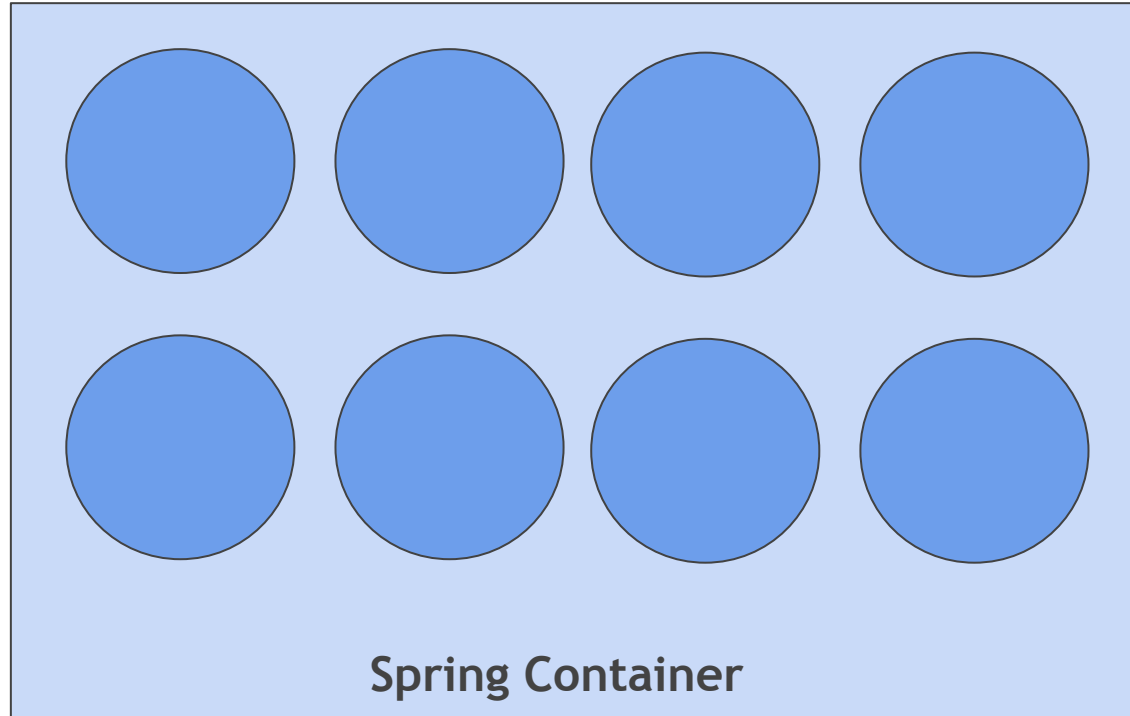
Types of Spring Containers

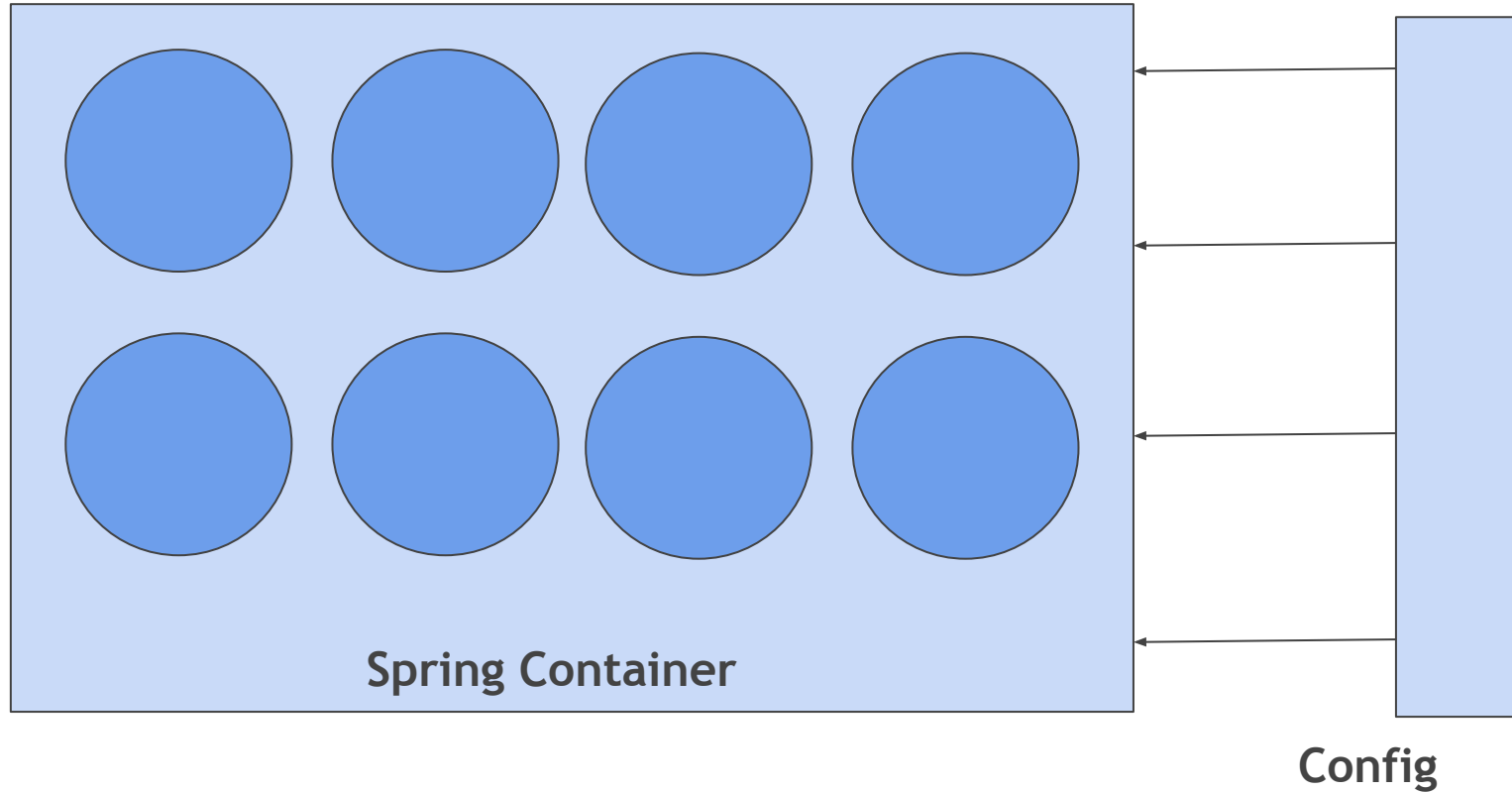


```
graph TD; A[Types of Spring Containers] --> B[ApplicationContext]; A --> C[BeanFactory]
```

ApplicationContext

BeanFactory





Configuration contains
bean definition

Thank you

Lifecycle of Bean

Faisal Memon (EmbarkX)

Beans

Objects that are managed by frameworks are known as Beans

Beans

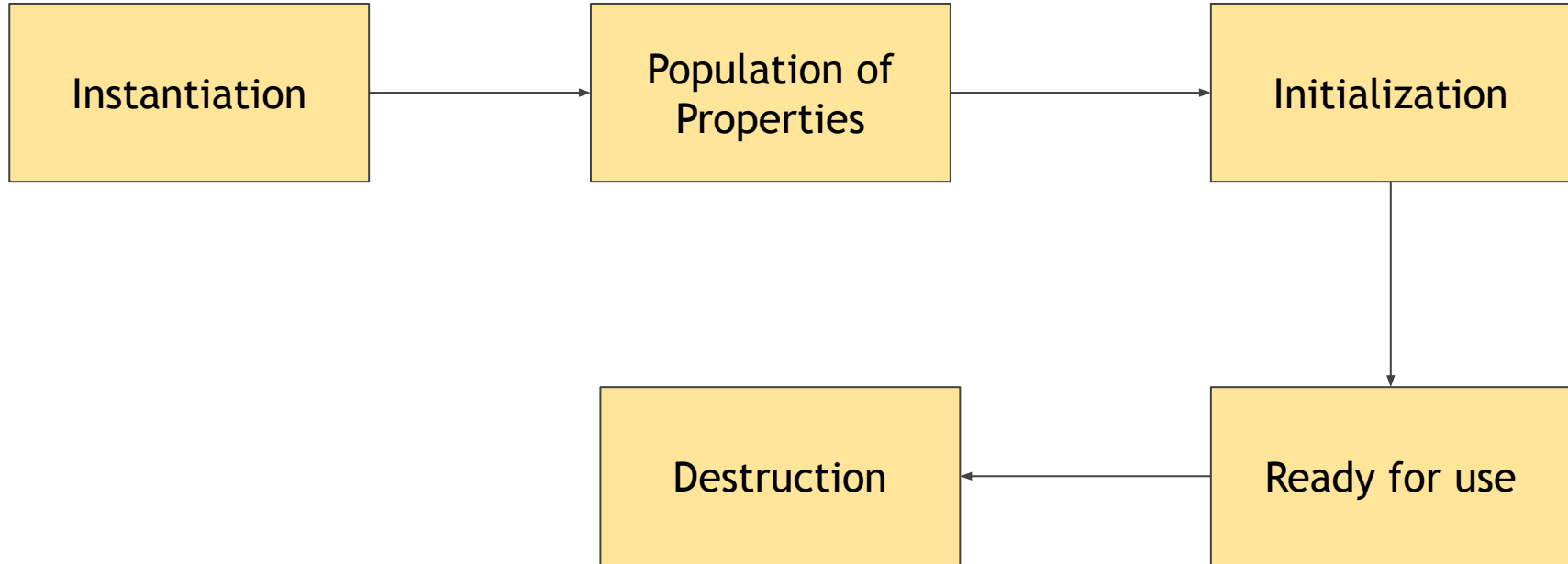
Bean Definition

- *A bean definition includes configuration metadata that the container needs to know to create and manage the bean*

Bean Configuration

- *Bean definitions can be provided in various ways, including XML configuration files, annotations, and Java-based configuration.*
- *Beans are configured using XML files, where each bean is defined within <bean> tags with attributes specifying class, properties, and dependencies.*
- *Beans can be configured using annotations like @Component, @Service, @Repository, etc., which are scanned by Spring and managed as beans.*

Lifecycle of Beans



Dependency Resolution

→ *Dependency Injection*

→ *Autowiring*

Thank you

Dependency Injection (DI)

Faisal Memon (EmbarkX)

Dependency Injection (DI) is a design pattern used in software development to achieve loose coupling between classes by removing the direct dependency instantiation from the dependent class itself

Types

→ *Constructor Injection*

→ *Setter Injection*

Thank you

Constructor Injection

Faisal Memon (EmbarkX)

Constructor Injection

- *Dependencies are provided to the dependent class through its constructor*
- *Dependencies are passed as arguments to the constructor when the dependent class is instantiated*
- *Constructor injection ensures that the dependencies are available when the object is created*

Thank you

Setter Injection

Faisal Memon (EmbarkX)

Setter Injection

- *Dependencies are provided to the dependent class through setter methods*
- *Dependent class exposes setter methods for each dependency that needs to be injected*
- *Setter injection allows for flexibility as dependencies can be changed or updated after the object is instantiated*

Thank you

Introduction to Annotations

Faisal Memon (EmbarkX)

Annotations in Java
provide a way to add
metadata to your code

@Override

Commonly Used Spring Annotations

→ *@Component*

→ *@Autowired*

→ *@Qualifier*

→ *@Value*

→ *@Repository*

Commonly Used Spring Annotations

→ *@Service*

→ *@Controller*

→ *@RequestMapping*

→ *@SpringBootApplication*

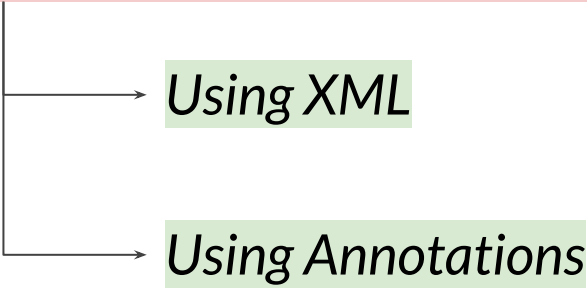
Thank you

Understanding Components and ComponentScan

Faisal Memon (EmbarkX)

Component refers to a Java class that is managed by the Spring IoC container

Defining Components in Spring



```
graph LR; A[Defining Components in Spring] --> B[Using XML]; A --> C[Using Annotations];
```

Using XML

Using Annotations

Using XML

```
<bean id="myComponent" class="com.example.MyComponent" />
```

Using Annotations

```
import org.springframework.stereotype.Component;  
  
@Component // Marks the class as a Spring component  
  
public class MyComponent {  
    // Class implementation  
}
```

Component scanning is a feature helps to automatically detect and register beans from predefined package paths.

Using XML

```
<!-- Enable component scanning -->
```

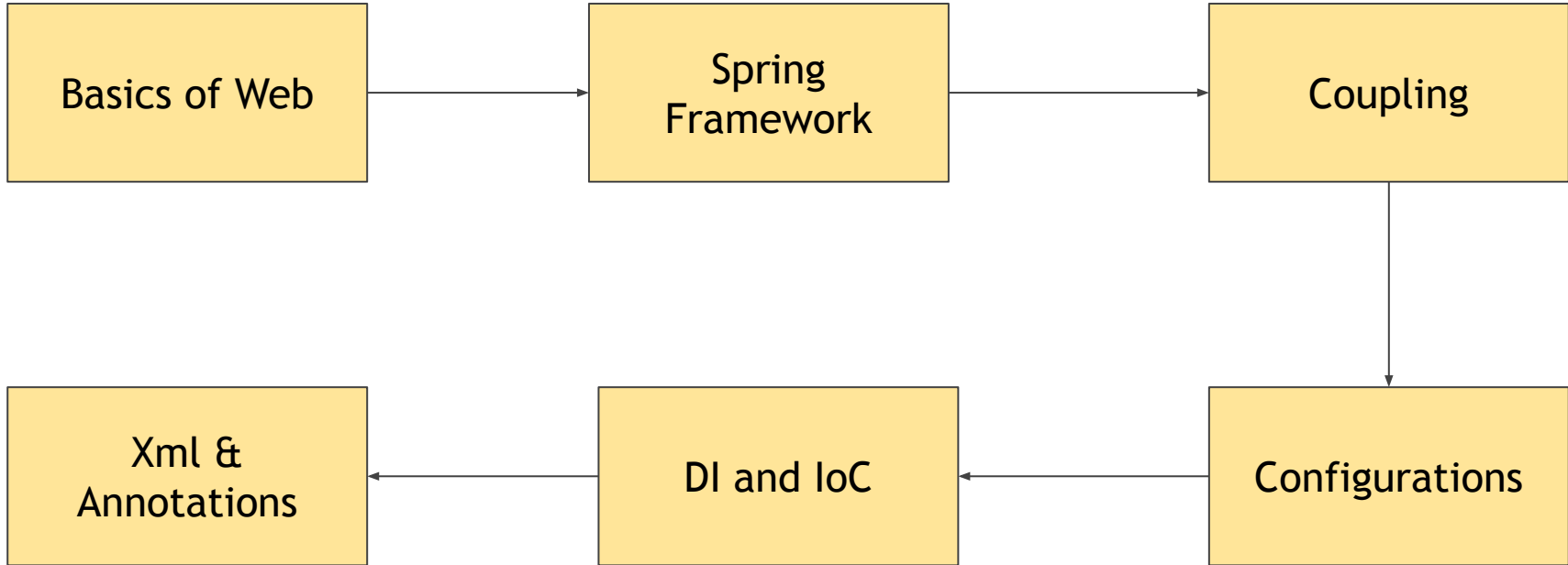
```
<context:component-scan base-package="car.example.componentscan" />
```

Thank you

Progress and Review So Far

Faisal Memon (EmbarkX)

Review



Review

→ *Explicit Bean Configuration*

→ *No Embedded Server*

→ *Component Scanning*

→ *Boilerplate code*

Thank you

What is Spring Boot?

Faisal Memon (EmbarkX)

What is Spring Boot?

Open-source, Java-based framework used to create stand-alone, production-grade Spring-based Applications

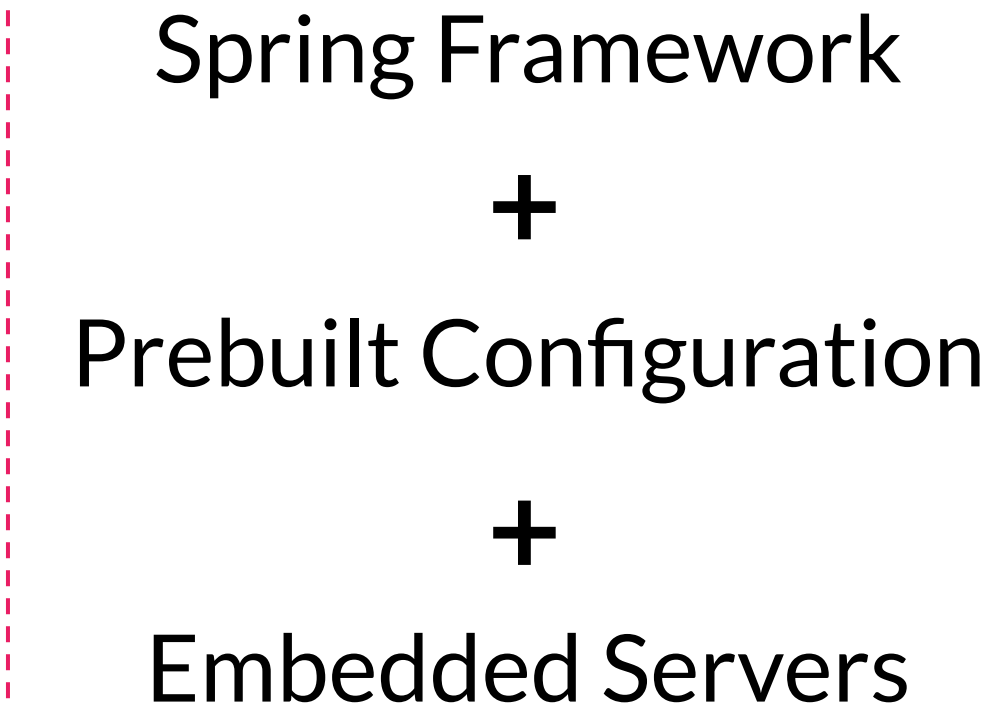
Spring

VS

Spring Boot

Lots of steps involved in setting up, configuration, writing boilerplate code, deployment of the app

Offers a set of pre-configured components or defaults, and eliminating the need for a lot of boilerplate code that was involved in setting up a Spring application

Spring boot =  Spring Framework
+
Prebuilt Configuration
+
Embedded Servers

Components of Spring Boot

- *Spring Boot Starters*
- *Auto Configuration*
- *Spring Boot Actuator*
- *Embedded Server*
- *Spring Boot DevTools*

Advantages of Spring Boot

- *Stand alone and Quick Start*
- *Starter code*
- *Less configuration*
- *Reduced cost and application development time*

Why do developers love Spring Boot?

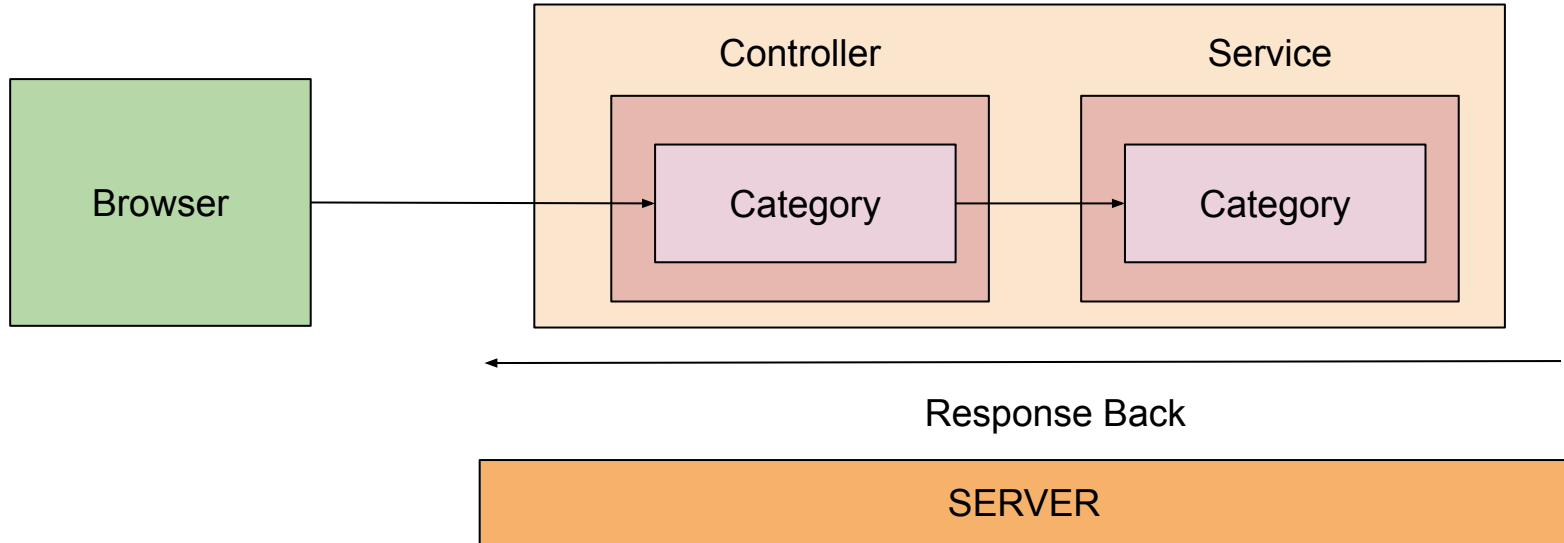
- *Java based*
- *Fast, easy*
- *Comes with embedded server*
- *Various plugins*
- *Avoids boilerplate code and configurations*

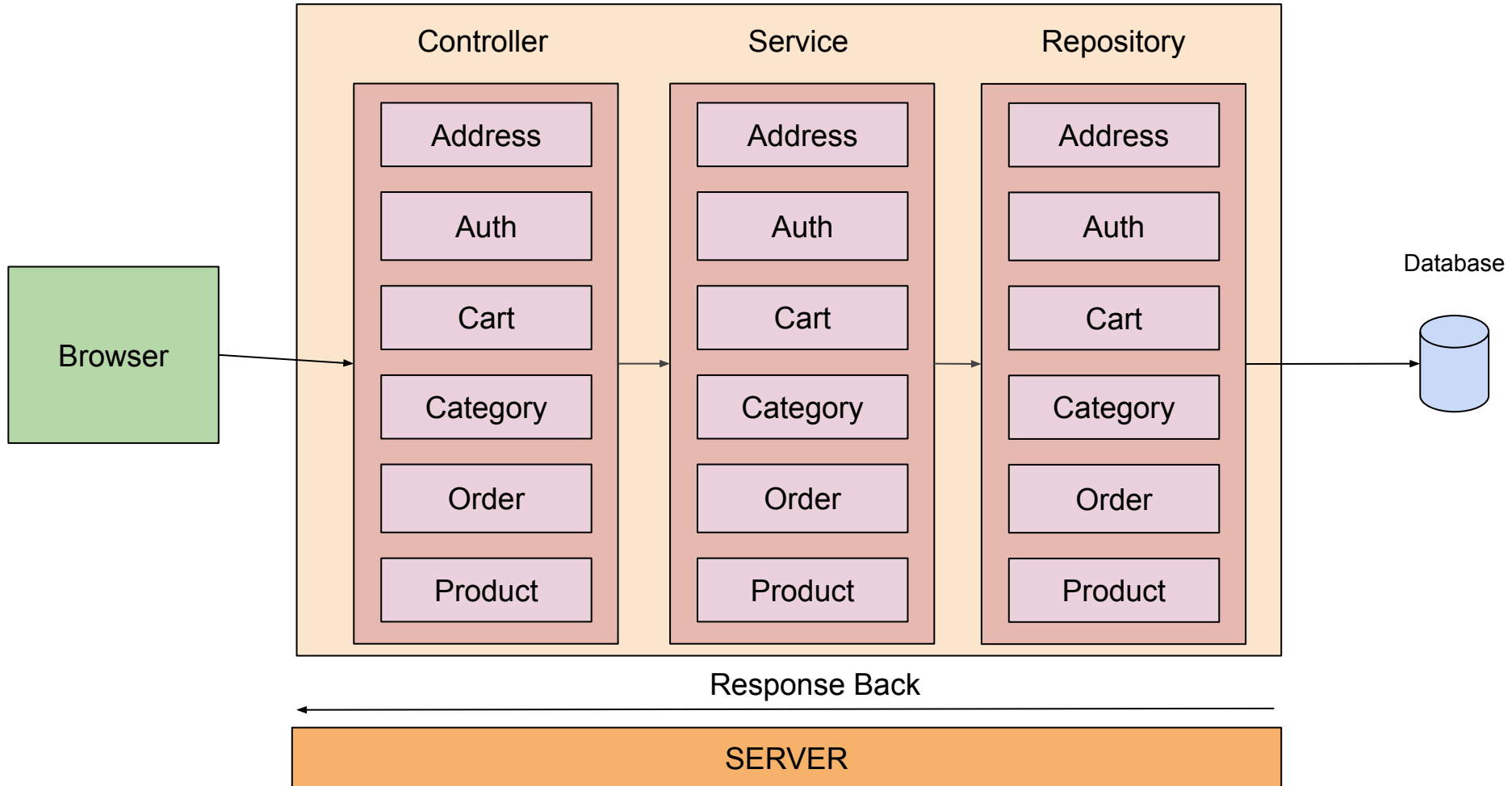
Thank you

Structuring Thoughts

Faisal Memon (EmbarkX)

OUR APPLICATION





API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Create Category	/api/admin/category	POST	Create a new category	Category	None	CategoryDTO
Get Categories	/api/public/categories	GET	Retrieve a list of categories	None	pageNumber, pageSize, sortBy, sortOrder	CategoryResponse
Update Category	/api/admin/categories/{categoryId}	PUT	Update an existing category	Category	categoryId	CategoryDTO
Delete Category	/api/admin/categories/{categoryId}	DELETE	Delete an existing category	None	categoryId	CategoryDTO

API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Create Category	/api/admin/category	POST	Create a new category	Category	None	CategoryDTO
Get Categories	/api/public/categories	GET	Retrieve a list of categories	None	pageNumber, pageSize, sortBy, sortOrder	CategoryResponse
Update Category	/api/admin/categories/{categoryId}	PUT	Update an existing category	Category	categoryId	CategoryDTO
Delete Category	/api/admin/categories/{categoryId}	DELETE	Delete an existing category	None	categoryId	CategoryDTO

API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Create Category	/api/admin/category	POST	Create a new category	Category	None	CategoryDTO
Get Categories	/api/public/categories	GET	Retrieve a list of categories	None	pageNumber, pageSize, sortBy, sortOrder	CategoryResponse
Update Category	/api/admin/categories/{categoryId}	PUT	Update an existing category	Category	categoryId	CategoryDTO
Delete Category	/api/admin/categories/{categoryId}	DELETE	Delete an existing category	None	categoryId	CategoryDTO

API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Create Category	/api/admin/category	POST	Create a new category	Category	None	CategoryDTO
Get Categories	/api/public/categories	GET	Retrieve a list of categories	None	pageNumber, pageSize, sortBy, sortOrder	CategoryResponse
Update Category	/api/admin/categories/{categoryId}	PUT	Update an existing category	Category	categoryId	CategoryDTO
Delete Category	/api/admin/categories/{categoryId}	DELETE	Delete an existing category	None	categoryId	CategoryDTO

API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Create Category	/api/admin/category	POST	Create a new category	Category	None	CategoryDTO
Get Categories	/api/public/categories	GET	Retrieve a list of categories	None	pageNumber, pageSize, sortBy, sortOrder	CategoryResponse
Update Category	/api/admin/categories/{categoryId}	PUT	Update an existing category	Category	categoryId	CategoryDTO
Delete Category	/api/admin/categories/{categoryId}	DELETE	Delete an existing category	None	categoryId	CategoryDTO

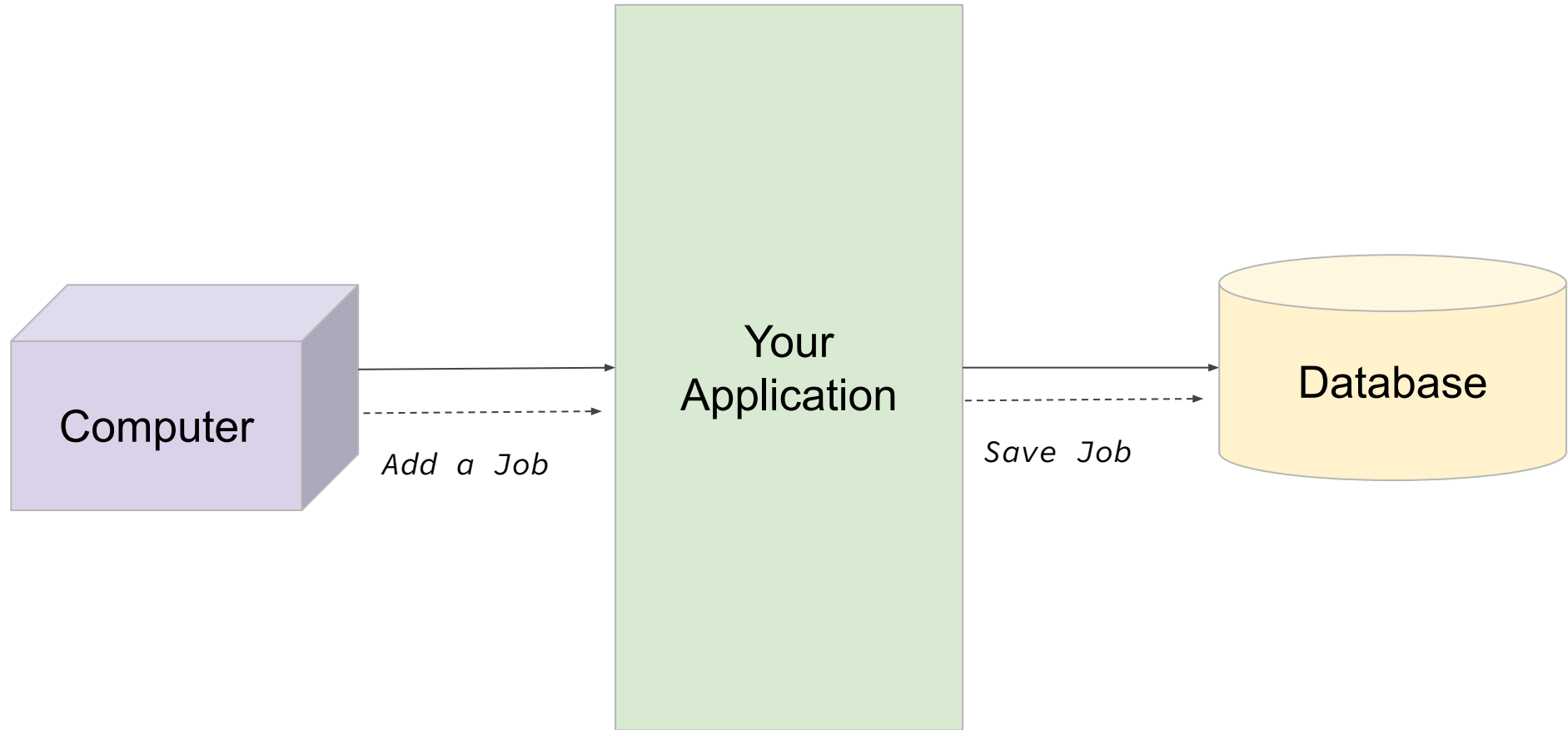
API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Create Category	/api/admin/category	POST	Create a new category	Category	None	CategoryDTO
Get Categories	/api/public/categories	GET	Retrieve a list of categories	None	pageNumber, pageSize, sortBy, sortOrder	CategoryResponse
Update Category	/api/admin/categories/{categoryId}	PUT	Update an existing category	Category	categoryId	CategoryDTO
Delete Category	/api/admin/categories/{categoryId}	DELETE	Delete an existing category	None	categoryId	CategoryDTO

Thank you

Understanding Data and Databases

Faisal Memon (EmbarkX)





What is a Database?

Database is a place where data related to your users and product is stored.

Databases Types

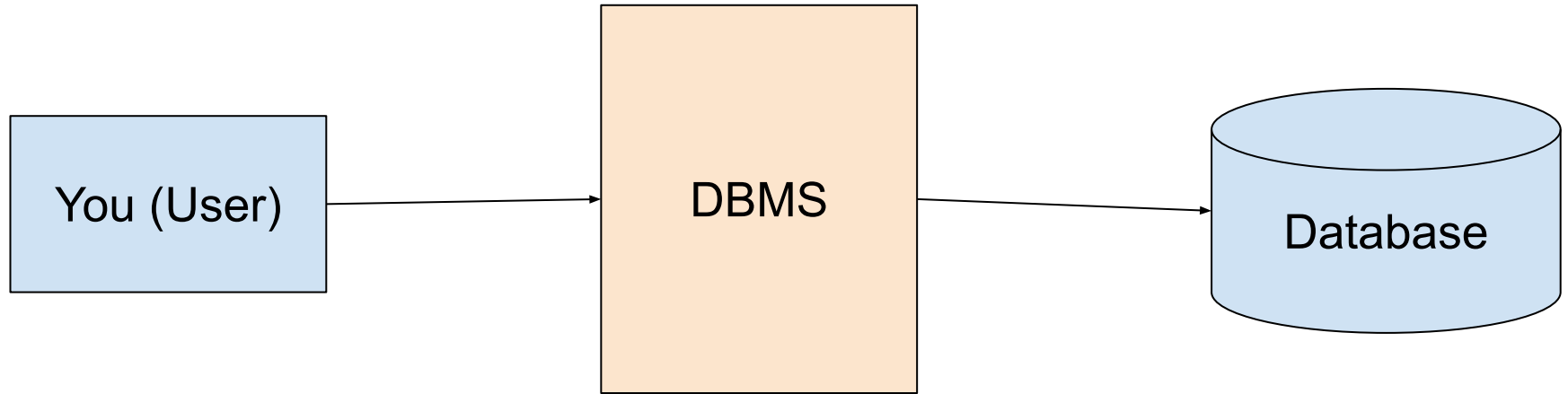
→ *Relational*

→ *Non-Relational*

Thank you

What is DBMS

Faisal Memon (EmbarkX)



Examples of DBMS

→ *MySQL*

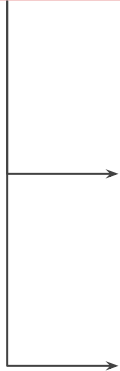
→ *Oracle Database*

→ *SQL Server*

→ *MongoDB*

→ *Cassandra*

Types of DBMS



*Relational Database Management System
(RDBMS)*

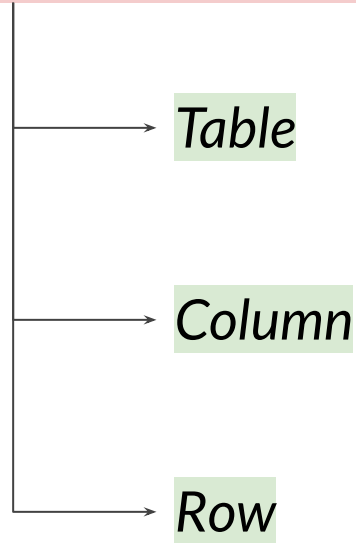
NoSQL Database Management System

Thank you

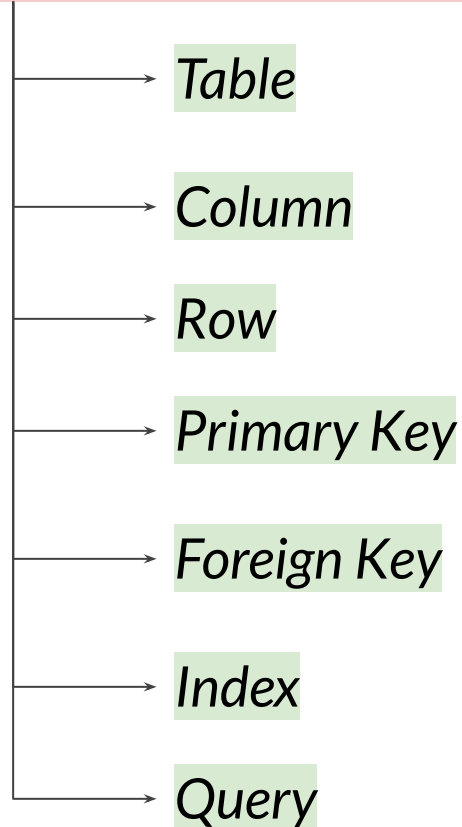
Introduction to Relational Databases Concepts

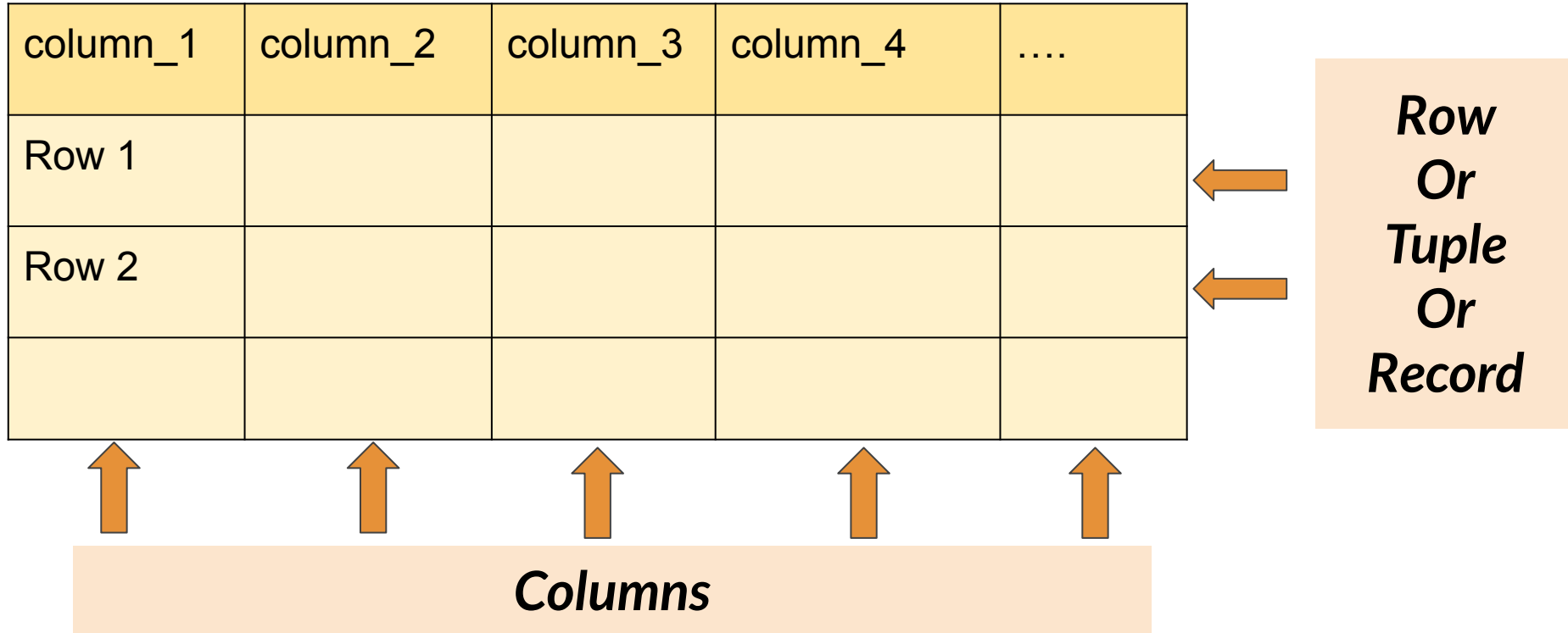
Faisal Memon (EmbarkX)

DBMS Terminologies



DBMS Terminologies





Rental ID	Car Make	Car Model	Rental Period	Rental Price
1	Maruti Suzuki	Vitara	3 days	\$150
2	Jaguar	F7	2 days	\$120
3	Renault	Duster	5 days	\$400
4	Chevrolet	Impala	7 days	\$700
5	TATA	Nexon	4 days	\$200
6	Kia	Seltos	1 day	\$50
7	Mahindra	XUV700	10 days	\$1000

**Row
Or
Tuple
Or
Record**

These are columns- may have diff datatypes

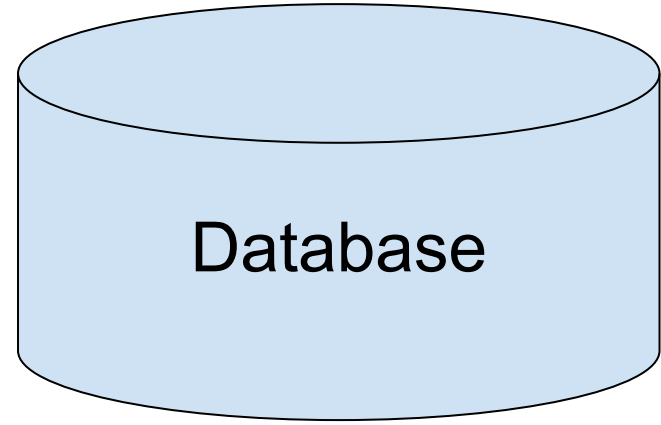
Rental ID	Car Make	Car Model	Rental Period	Rental Price
1	Maruti Suzuki	Vitara	3 days	\$150
2	Jaguar	F7	2 days	\$120
3	Renault	Duster	5 days	\$400
4	Chevrolet	Impala	7 days	\$700
5	TATA	Nexon	4 days	\$200
6	Kia	Seltos	1 day	\$50
7	Mahindra	XUV700	10 days	\$1000

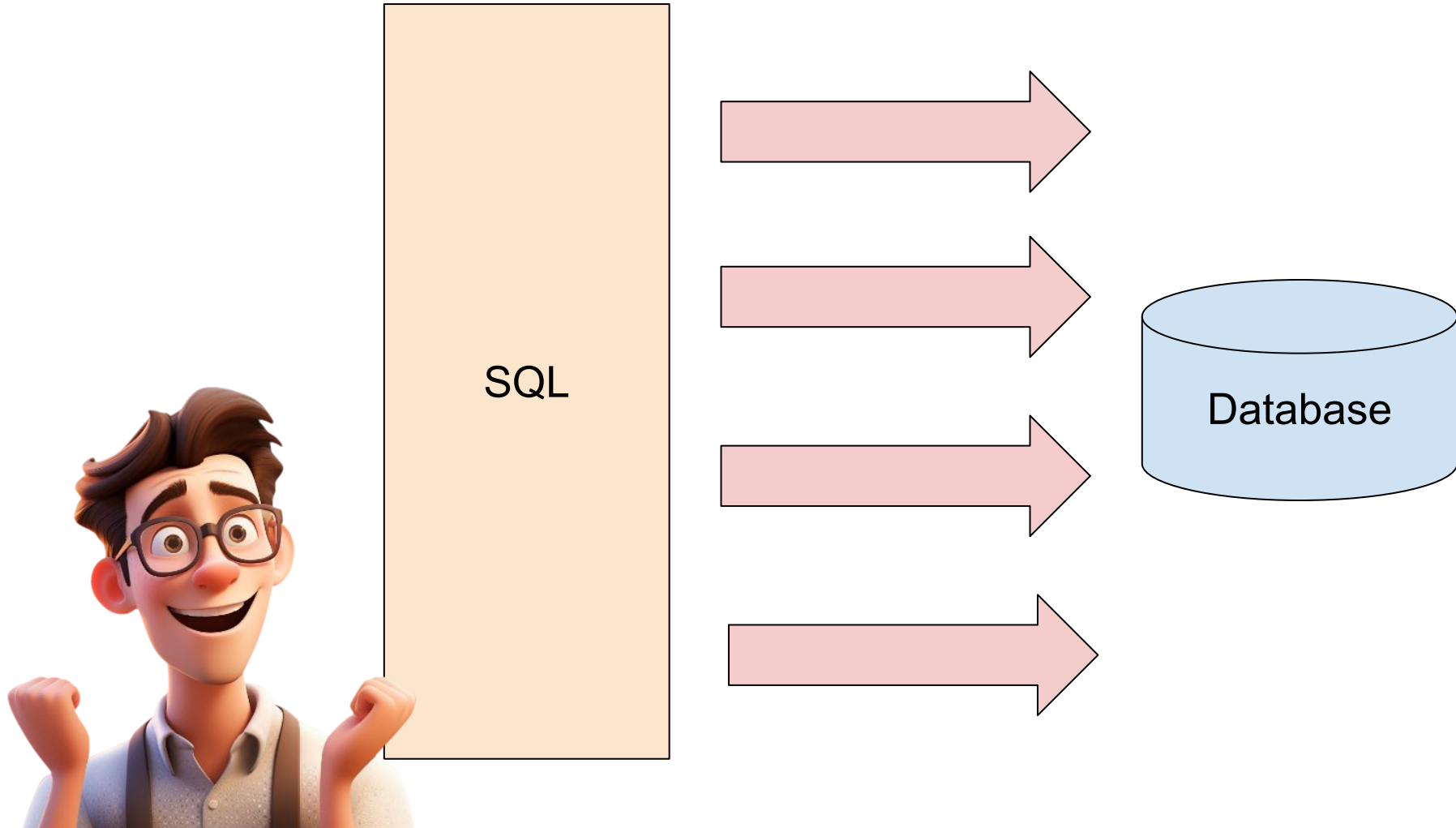
Thank you

Overview of SQL

Faisal Memon

How do I get the data?





What is SQL

*SQL stands for **S**tructured **Q**uery **L**anguage. Used to retrieve, manage and update data in Database*

SQL Queries

Data from Database is retrieved with the help of SQL queries.

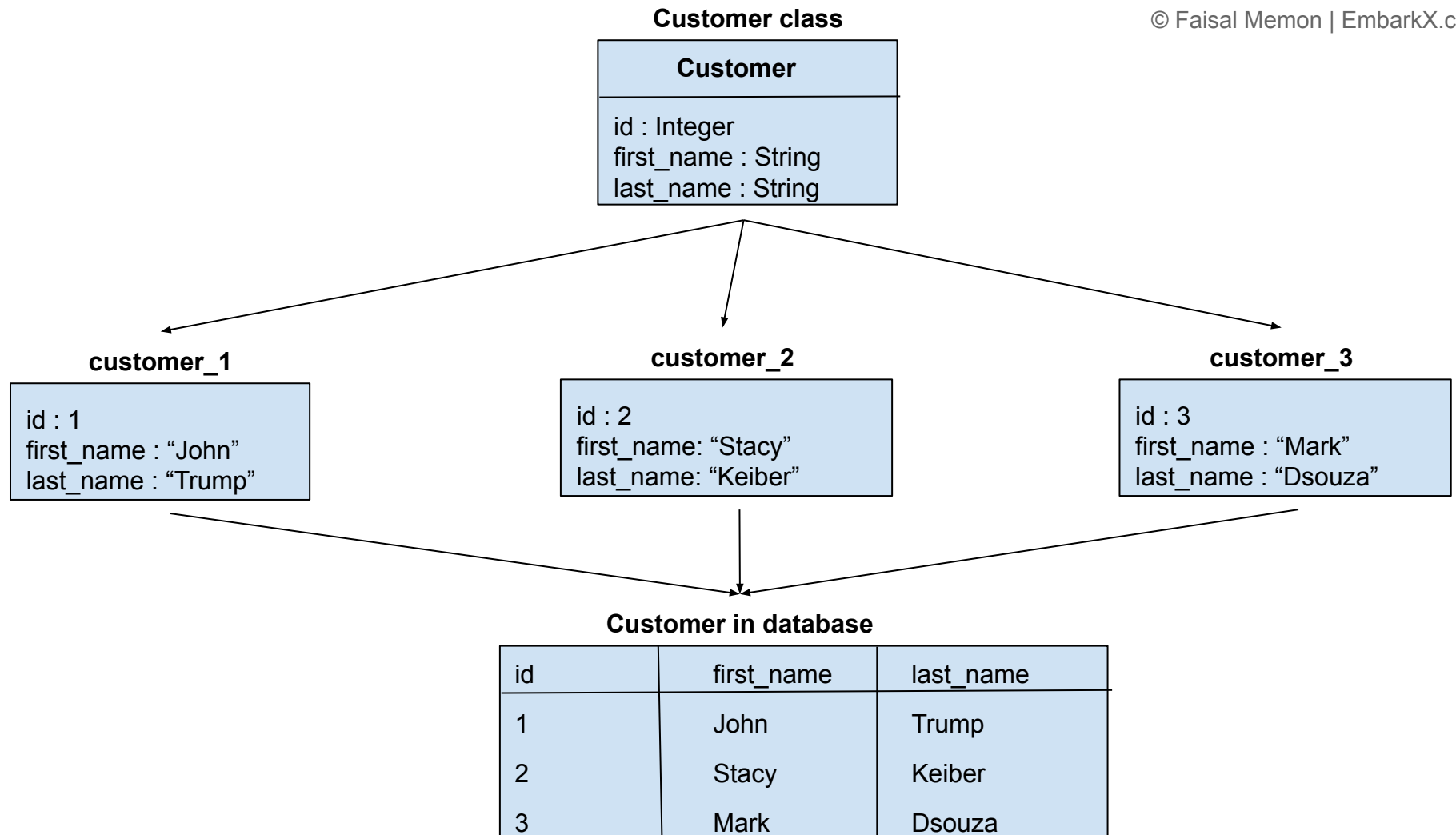
Different Types of SQL Queries

- *Data Query Language [DQL]*
- *Data Manipulation Language [DML]*
- *Data Definition Language [DDL]*
- *Data Control Language [DCL]*
- *Transaction Control Language(TCL)*

Thank you

What is ORM?

Faisal Memon (EmbarkX)



ORM

- *Whenever there is a class, that class can be automatically converted to a table with its attributes being converted to columns*
- *So now the developer does not have to write queries for table creation, it's created automatically*
- *Whenever an object is created, its data can be saved in the database as row in table, this is automatically handled by ORM*

ORM

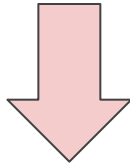
- *ORM as a concept makes developers lives easier and lets developers focus on application logic rather than SQL queries*
- *Because of ORM developers don't need to learn how to write SQL queries since the translation from application to SQL is handled by ORM itself*
- *It's a powerful technique in programming which also minimizes mistakes since developers are not writing queries on their own*

Thank you

What is JPA?

Faisal Memon (EmbarkX)

```
class Category {  
    Long categoryId;  
    String categoryName;  
}
```



categoryId	categoryName
1	Senior Software Engineer

Advantages of using JPA

- *Easy and Simple*
- *Makes querying easier*
- *Allows to save and update objects*
- *Easy integration with Spring Boot*

Thank you

Let's Understand Data Layer

Faisal Memon (EmbarkX)

Presentation Layer

Presentation layer presents the data and the application features to the user. This is the layer where in all the controller classes exist.

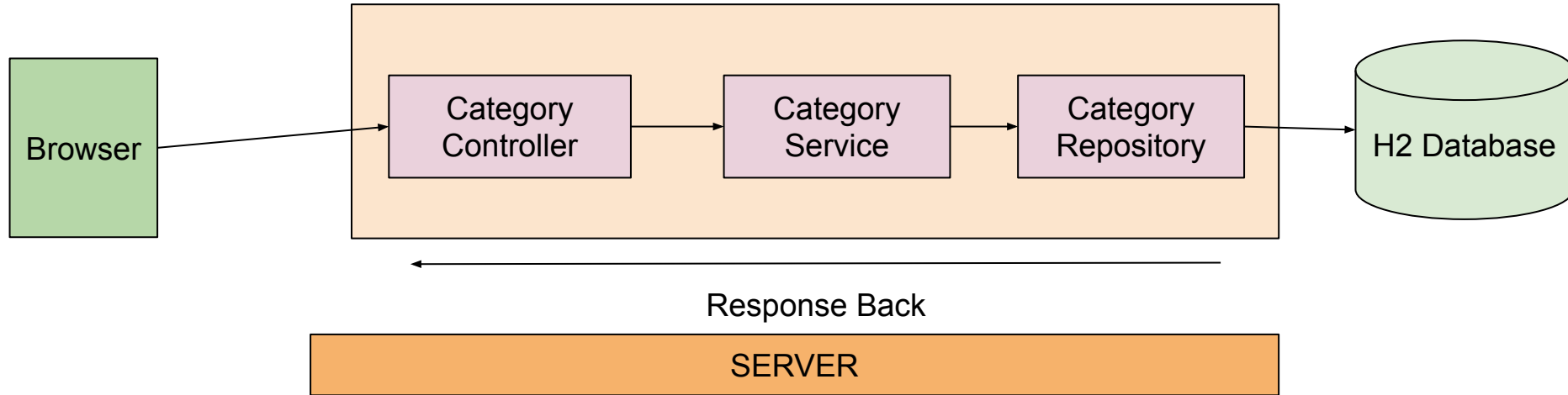
Service Layer

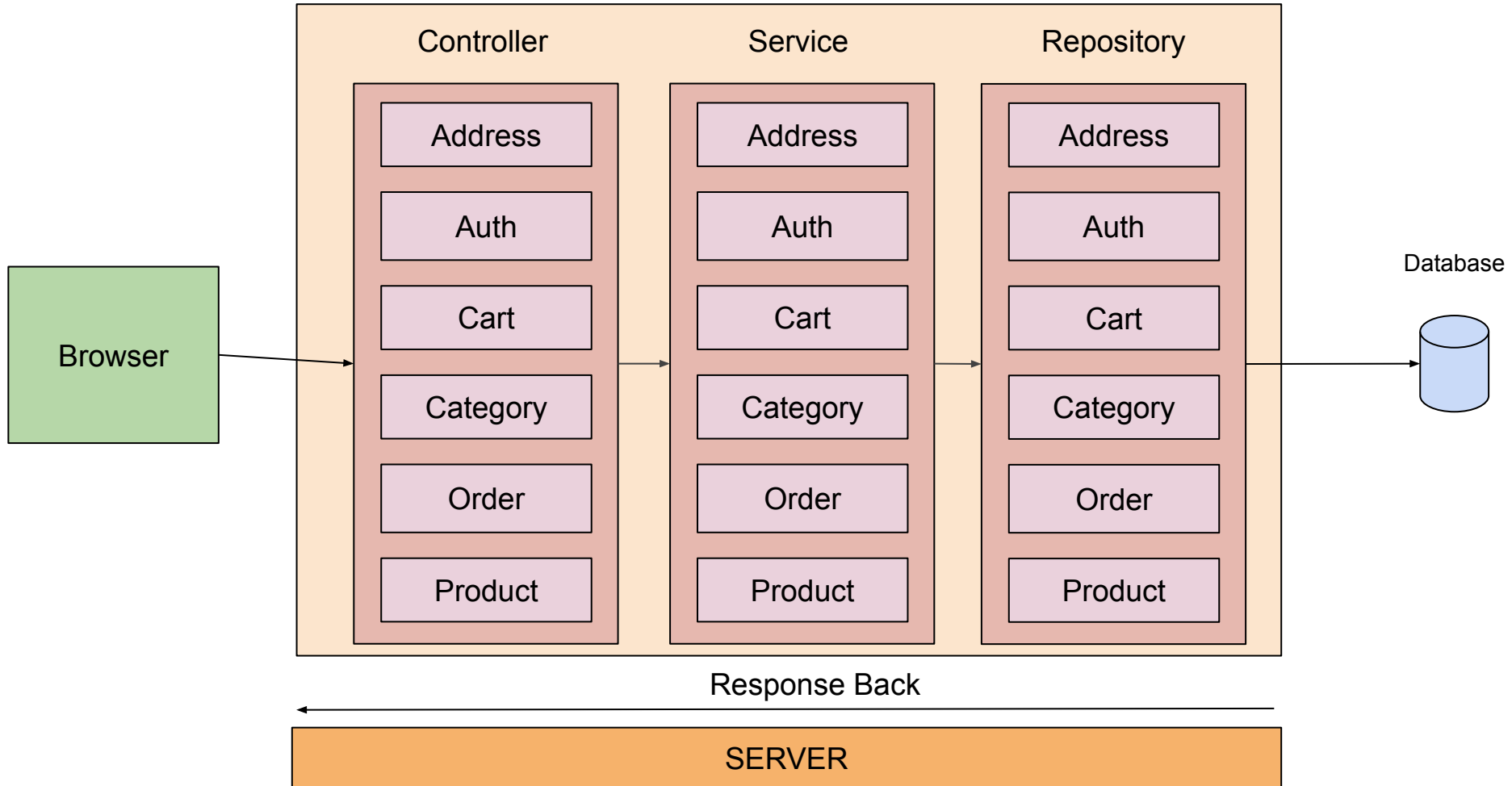
Service layer is where business logic resides in the application. Tasks such as evaluations, decision making, processing of data is done at this layer.

Data Access Layer

Data access layer is the layer where all the repository classes reside.

OUR APPLICATION



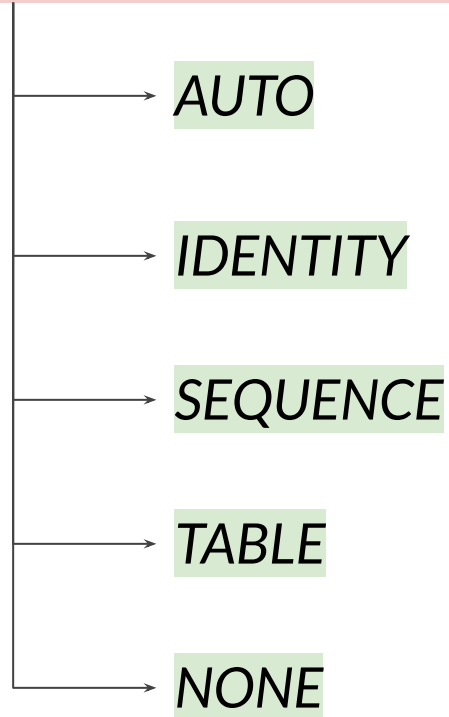


Thank you

Generation Types For Identity

Faisal Memon (EmbarkX)

Different Generation Types



GenerationType.AUTO

```
@Id  
@GeneratedValue(strategy = GenerationType.AUTO)  
private Long id;
```

GenerationType.IDENTITY

```
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
private Long id;
```

GenerationType.SEQUENCE

```
@Id  
@GeneratedValue(strategy = GenerationType.SEQUENCE)  
private Long id;
```

GenerationType.SEQUENCE

@Id

```
@GeneratedValue(strategy = GenerationType.SEQUENCE,  
generator = "order_seq")
```

```
@SequenceGenerator(name = "order_seq", sequenceName =  
"order_sequence", allocationSize = 1)
```

```
private Long id;
```

GenerationType.SEQUENCE

@Id

```
@GeneratedValue(strategy = GenerationType.SEQUENCE,  
generator = "order_seq")
```

```
@SequenceGenerator(name = "order_seq", sequenceName =  
"order_sequence", allocationSize = 1)
```

```
private Long id;
```

GenerationType.SEQUENCE

@Id

```
@GeneratedValue(strategy = GenerationType.SEQUENCE,  
generator = "order_seq")
```

```
@SequenceGenerator(name = "order_seq", sequenceName =  
"order_sequence", allocationSize = 1)
```

```
private Long id;
```

GenerationType.SEQUENCE

@Id

```
@GeneratedValue(strategy = GenerationType.SEQUENCE,  
generator = "order_seq")
```

```
@SequenceGenerator(name = "order_seq", sequenceName =  
"order_sequence", allocationSize = 1)
```

```
private Long id;
```


GenerationType.SEQUENCE

@Id

```
@GeneratedValue(strategy = GenerationType.SEQUENCE,  
generator = "order_seq")
```

```
@SequenceGenerator(name = "order_seq", sequenceName =  
"order_sequence", allocationSize = 1)
```

```
private Long id;
```

GenerationType.SEQUENCE

@Id

```
@GeneratedValue(strategy = GenerationType.SEQUENCE,  
generator = "order_seq")
```

```
@SequenceGenerator(name = "order_seq", sequenceName =  
"order_sequence", allocationSize = 1)
```

```
private Long id;
```

GenerationType.TABLE

```
@Id  
@GeneratedValue(strategy = GenerationType.TABLE)  
private Long id;
```

GenerationType.TABLE

@Id

```
@GeneratedValue(strategy = GenerationType.TABLE,  
generator = "task_gen")
```

```
@TableGenerator(name = "task_gen", table = "id_gen",  
pkColumnName = "gen_key", valueColumnName = "gen_value",  
pkColumnValue = "task_id", allocationSize = 1)
```

```
private Long id;
```

GenerationType.TABLE

@Id

```
@GeneratedValue(strategy = GenerationType.TABLE,  
generator = "task_gen")
```

```
@TableGenerator(name = "task_gen", table = "id_gen",  
pkColumnName = "gen_key", valueColumnName = "gen_value",  
pkColumnValue = "task_id", allocationSize = 1)
```

```
private Long id;
```

GenerationType.TABLE

@Id

```
@GeneratedValue(strategy = GenerationType.TABLE,  
generator = "task_gen")
```

```
@TableGenerator(name = "task_gen", table = "id_gen",  
pkColumnName = "gen_key", valueColumnName = "gen_value",  
pkColumnValue = "task_id", allocationSize = 1)
```

```
private Long id;
```

GenerationType.TABLE

@Id

```
@GeneratedValue(strategy = GenerationType.TABLE,  
generator = "task_gen")
```

```
@TableGenerator(name = "task_gen", table = "id_gen",  
pkColumnName = "gen_key", valueColumnName = "gen_value",  
pkColumnValue = "task_id", allocationSize = 1)
```

```
private Long id;
```

GenerationType.TABLE

@Id

```
@GeneratedValue(strategy = GenerationType.TABLE,  
generator = "task_gen")
```

```
@TableGenerator(name = "task_gen", table = "id_gen",  
pkColumnName = "gen_key", valueColumnName = "gen_value",  
pkColumnValue = "task_id", allocationSize = 1)
```

```
private Long id;
```


Thank you

Validations in Spring Boot

Faisal Memon (EmbarkX)

Validations in Spring Boot are all about ensuring the data your application receives meets certain criteria before it's processed

Validation in Spring Boot

→ *@NotNull*

→ *@NotEmpty*

→ *@Size(min = x, max = y)*

→ *@Email*

→ *@Min(value) and @Max(value)*

Example

```
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotEmpty;
import jakarta.validation.constraints.Size;

public class User {
    @NotEmpty(message = "Email cannot be empty")
    @Email(message = "Email should be valid")
    private String email;

    @NotEmpty(message = "Name cannot be empty")
    @Size(min = 2, message = "Name should have at least 2 characters")
    private String name;

    // getters and setters
}
```

Thank you

Custom Exceptions in Spring Boot

Faisal Memon (EmbarkX)

Use of **ResponseStatusException**

Why Consider Custom Exceptions Anyway?

- *Separation of Concerns*
- *Consistency and Reusability*
- *Detailed Error Information*
- *Complex Error Handling Logic*

Using Custom Exceptions with ResponseStatusException

- *ResponseStatusException* for direct feedback
- *Define Custom Exceptions for Business Logic*
- *Handle Custom Exceptions in Controller Advice*
- *Custom Exceptions for consistency*

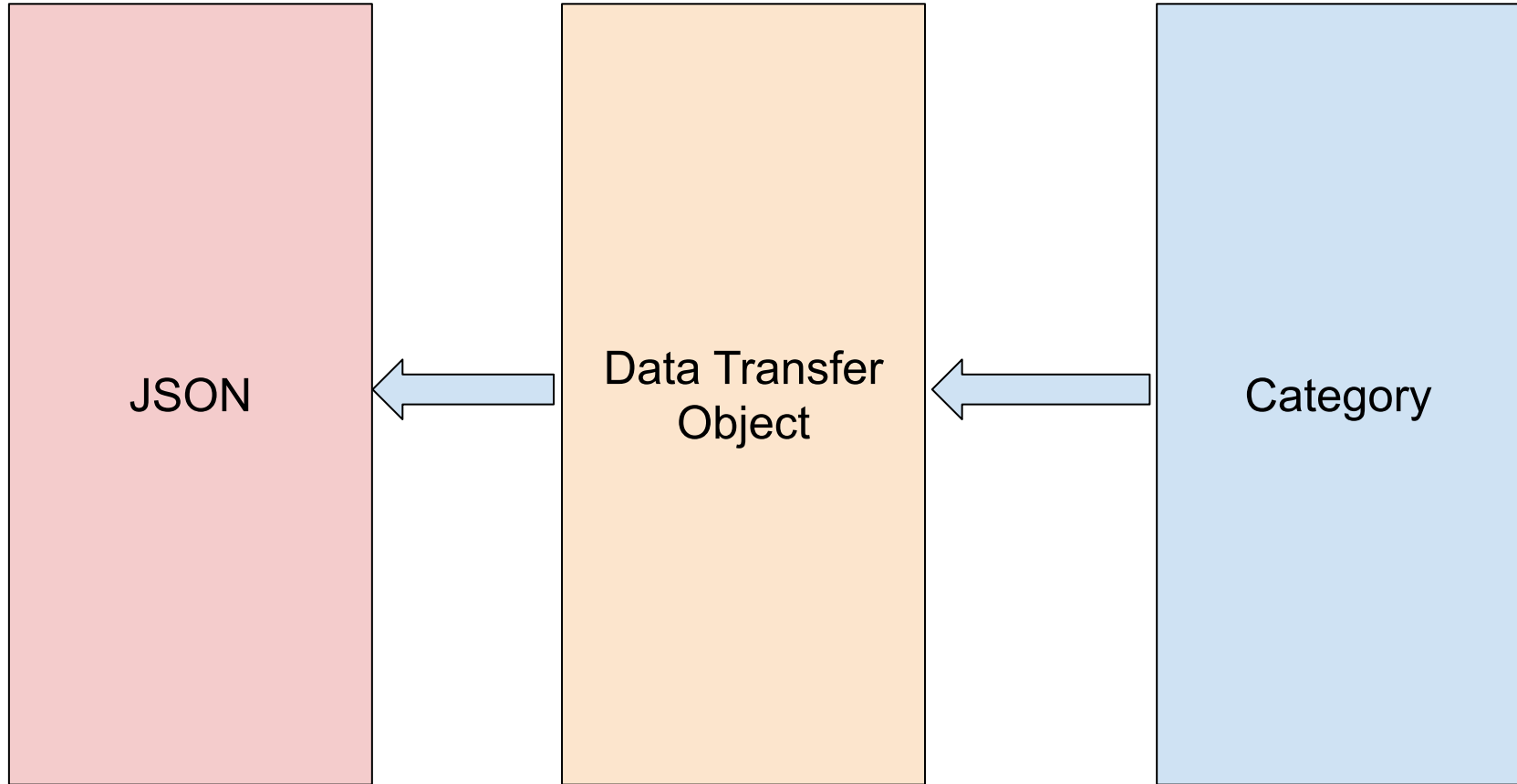
Thank you

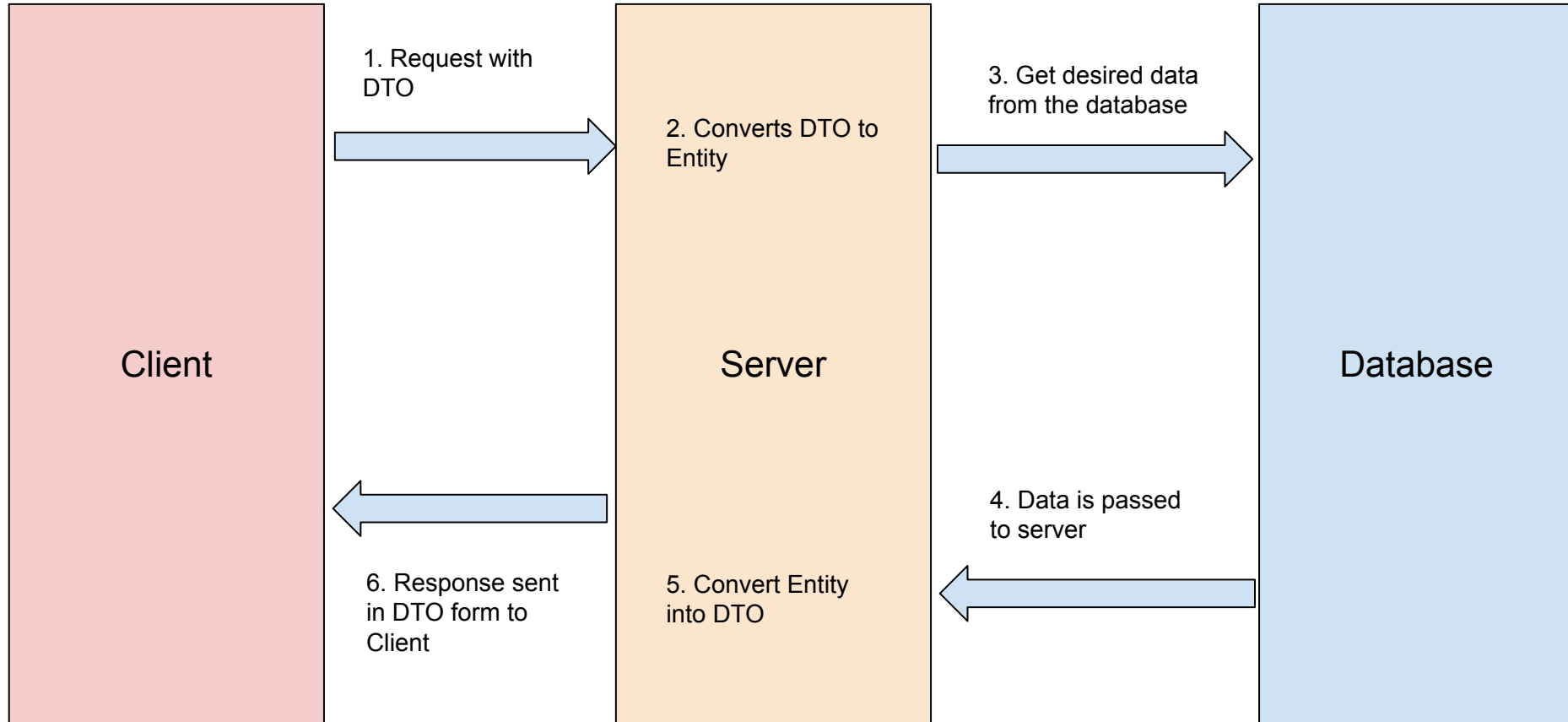
DTO Pattern

Faisal Memon (EmbarkX)

What is it?

Design pattern used to transfer data between software application subsystems





Thank you

Working with Multiple Entities / Relationships

Faisal Memon (EmbarkX)

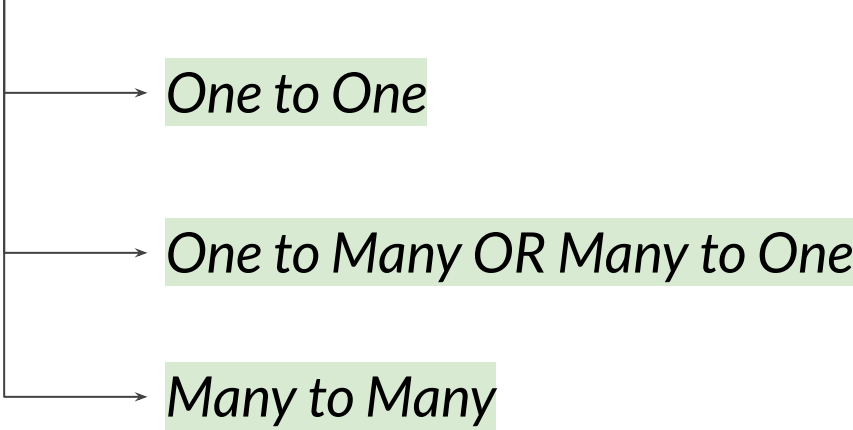
Real world projects will
have multiple models

Entity relationships are
important

JPA and Relationships

- *An entity represents a table in your database*
- *Each instance of an entity corresponds to a row in that table*
- *If you have a table for storing information about books, each book object would be a row*
- *Relationships in JPA define how entities are related to each other*
- *JPA allows you to map these relationships using annotations in your Java code*

Relationships



One to One

One to Many OR Many to One

Many to Many

One to One Relationship

One to one relationship is a type of relationship where in one record in a table is related to exactly one more record in another table and vice versa

Example

Employee and salary account



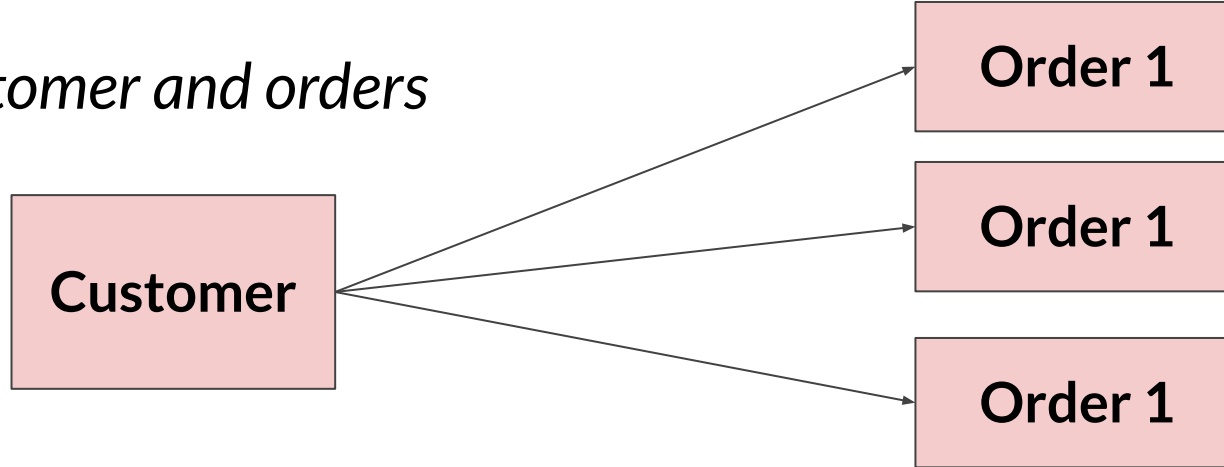
Example 2 → Person and passport

Many to One relationship

A many to one relationship is a type of relationship when one record in one table has one or many related record in another table

Example

Customer and orders



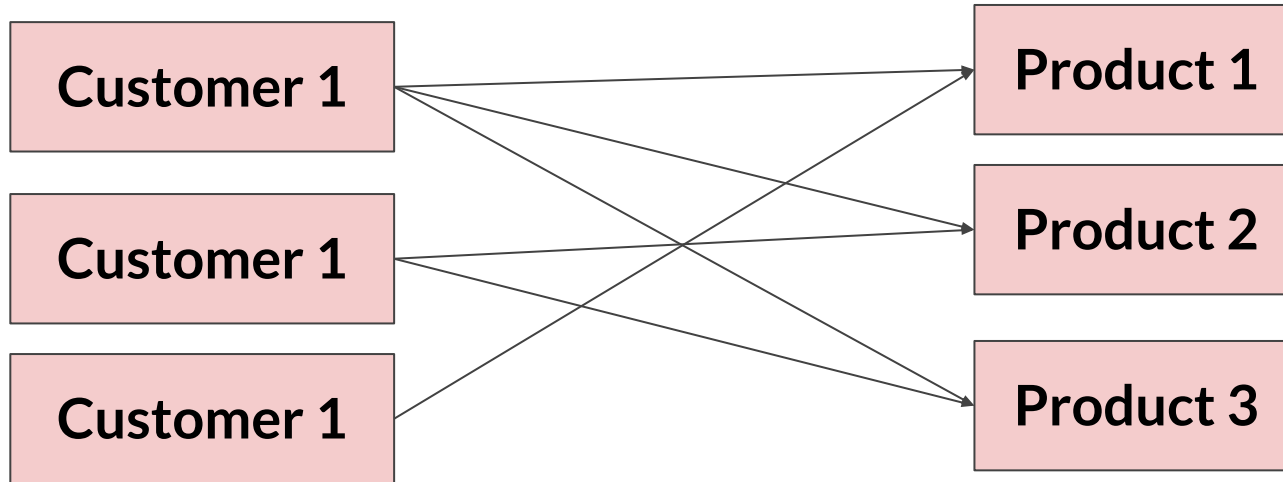
Users and Bank accounts

Many to Many relationship

A many to many relationship is a type of relationship when one record in one table has many related record in another table

Example

Customer and products



Courses and Students

Unidirectional Relationship

When only one entity knows about the relationship

Bidirectional Relationship

When both entities are aware of each other

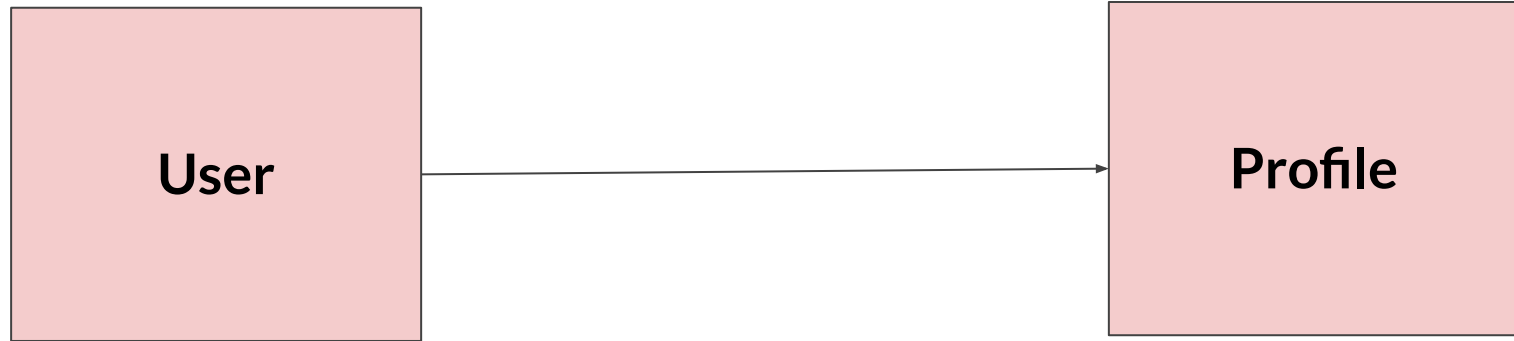
Thank you

One to One Relationship


Faisal Memon (EmbarkX)

Occurs when one record in a table is associated with one and only one record in another table

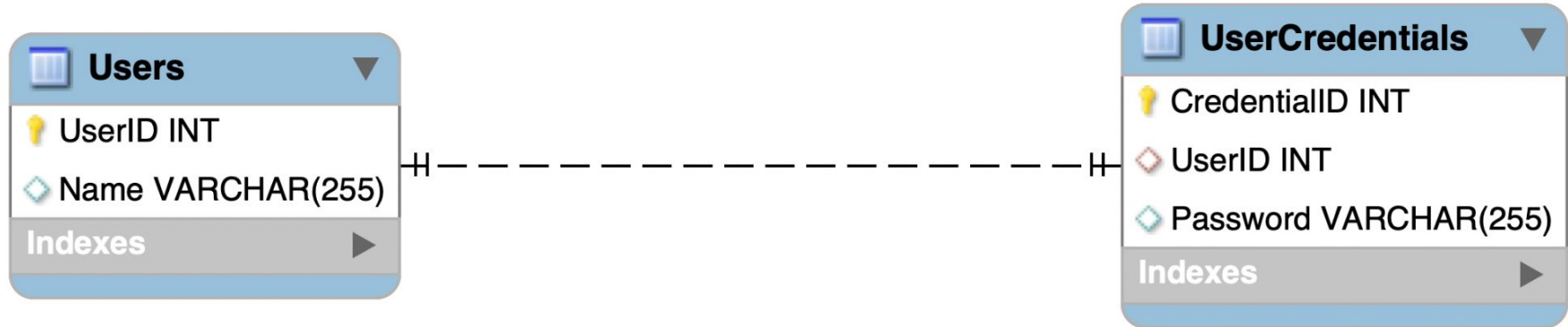
Example



Why Do We Need 1:1 Relationship

- 
- *Separation of sensitive data*
 - *Optional data*
 - *Splitting for performance*

Example



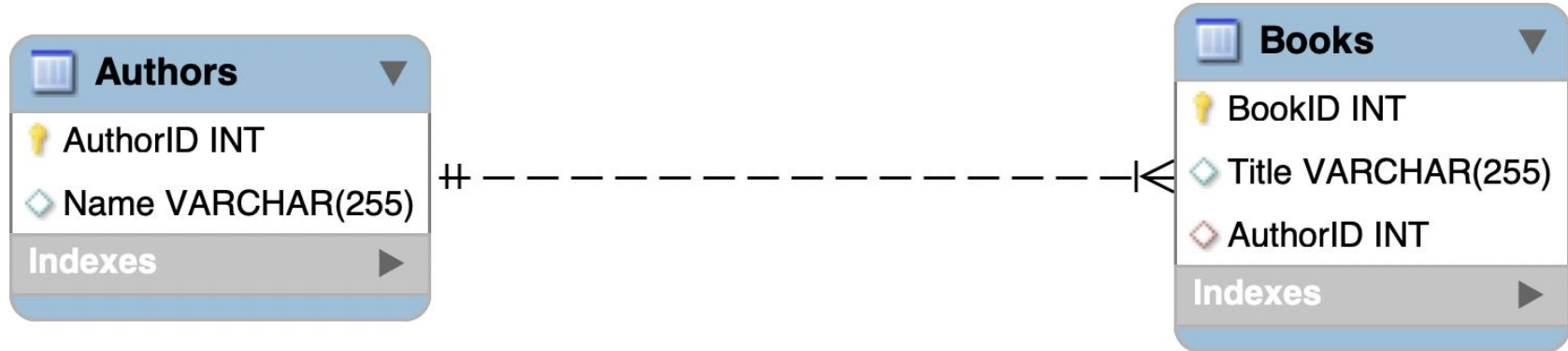
Thank you

One to Many AND Many to One

Faisal Memon (EmbarkX)

Occurs when one record in one table can be associated with multiple records in another table

Example



Before

BookID	Title	AuthorName
1	Quantum Realm	Alice Smith
2	Particle Play	Bob Johnson
3	Atomic Actions	Alice Smith

After

AuthorID	Name
1	Alice Smith
2	Bob Johnson

BookID	Title	AuthorID
1	Quantum Realm	1
2	Particle Play	2
3	Atomic Actions	1

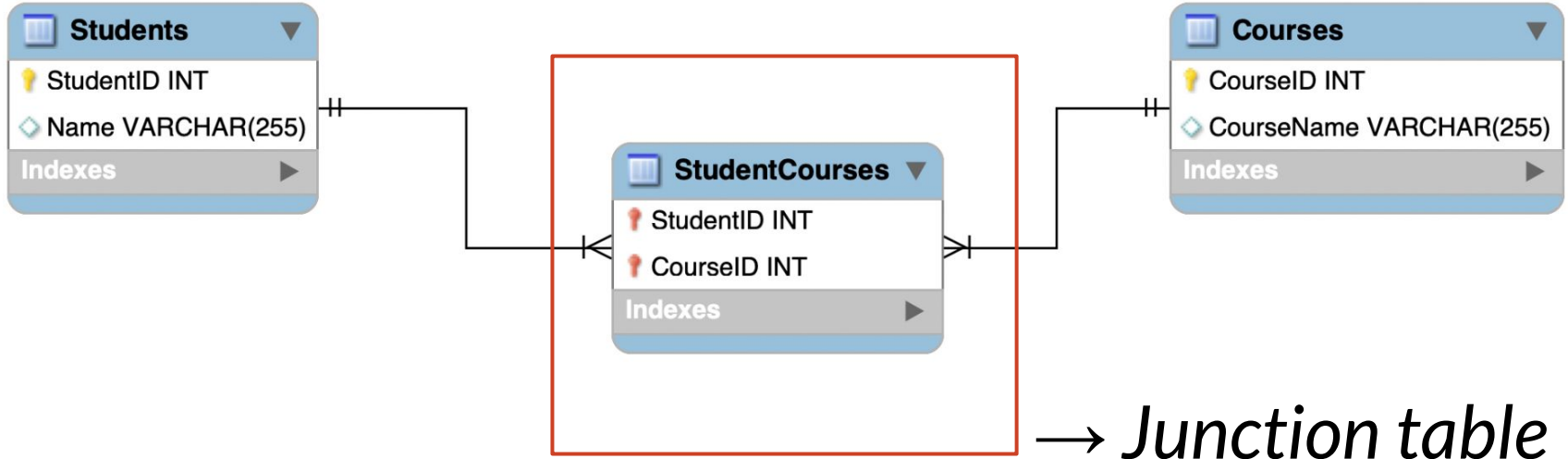
Thank you

Many to Many Relationship

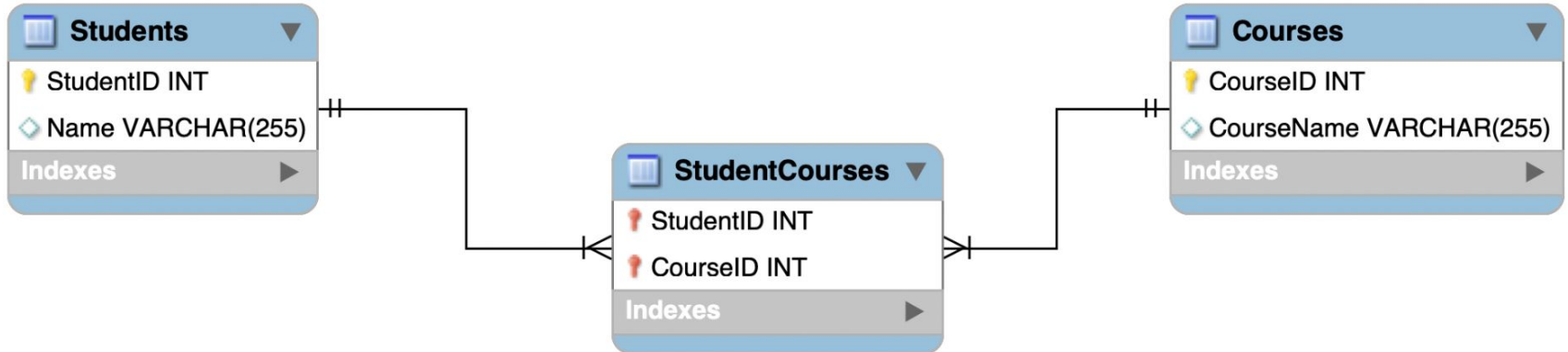
Faisal Memon (EmbarkX)

Occurs when multiple records
in one table can be associated
with multiple records in
another table

Example



Example



Things to remember

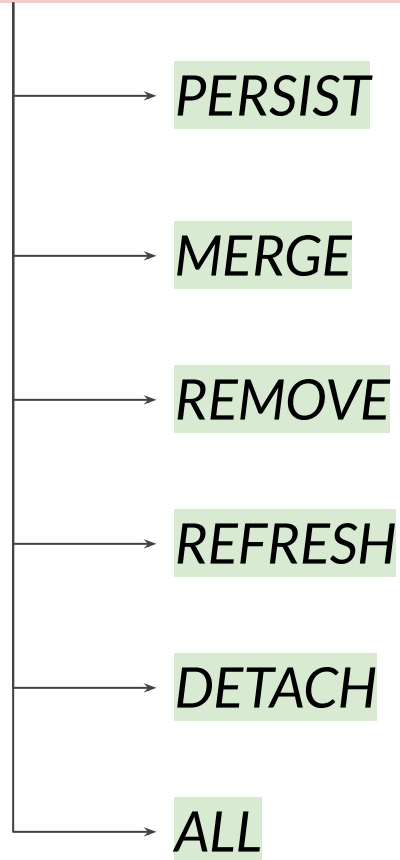
- *Junction table may contain additional attributes*
- *Junction table helps avoid redundancy*
- *To retrieve data you have to write JOIN statements that include the junction table*

Thank you

Cascading

Faisal Memon (EmbarkX)

Cascading Types

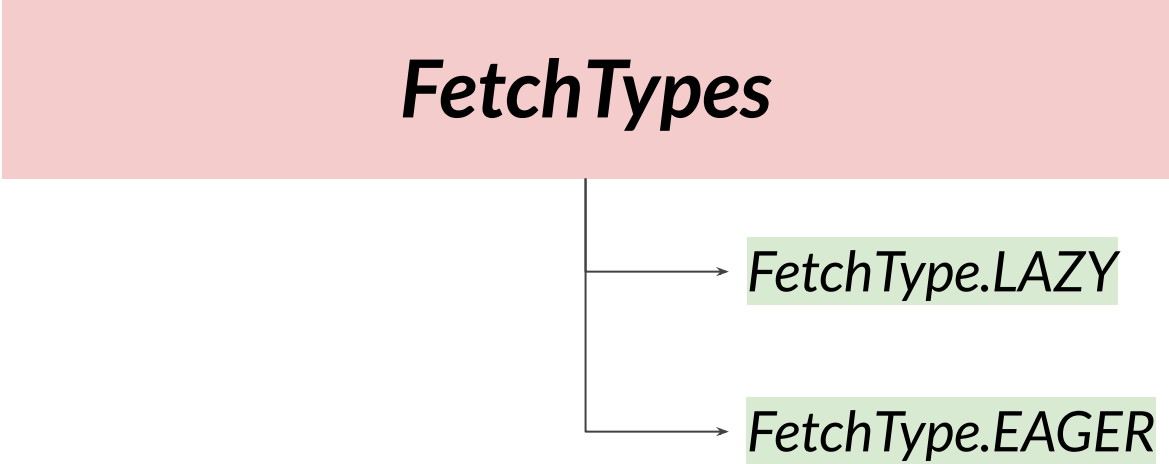


FetchTypes

Faisal Memon (EmbarkX)

FetchType plays a crucial role in defining how and when related entities are loaded from the database in relation to the parent entity

FetchTypes



```
graph TD; A[FetchTypes] --> B[FetchType.LAZY]; A --> C[FetchType.EAGER];
```

FetchType.LAZY

FetchType.EAGER

Default FetchType

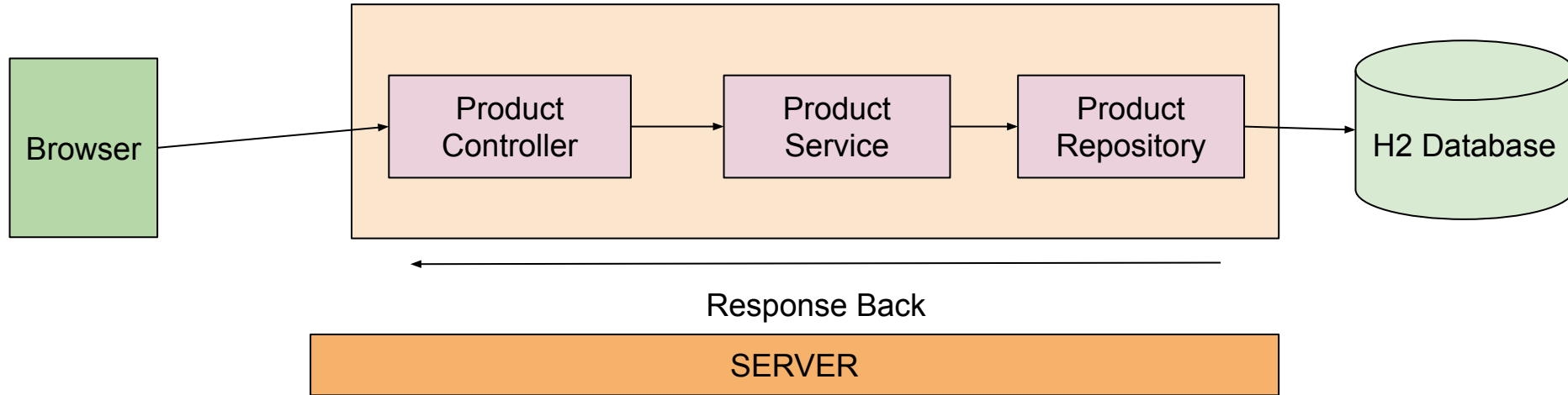
- *OneToMany: Lazy*
- *ManyToOne: Eager*
- *ManyToMany: Lazy*
- *OneToOne: Eager*

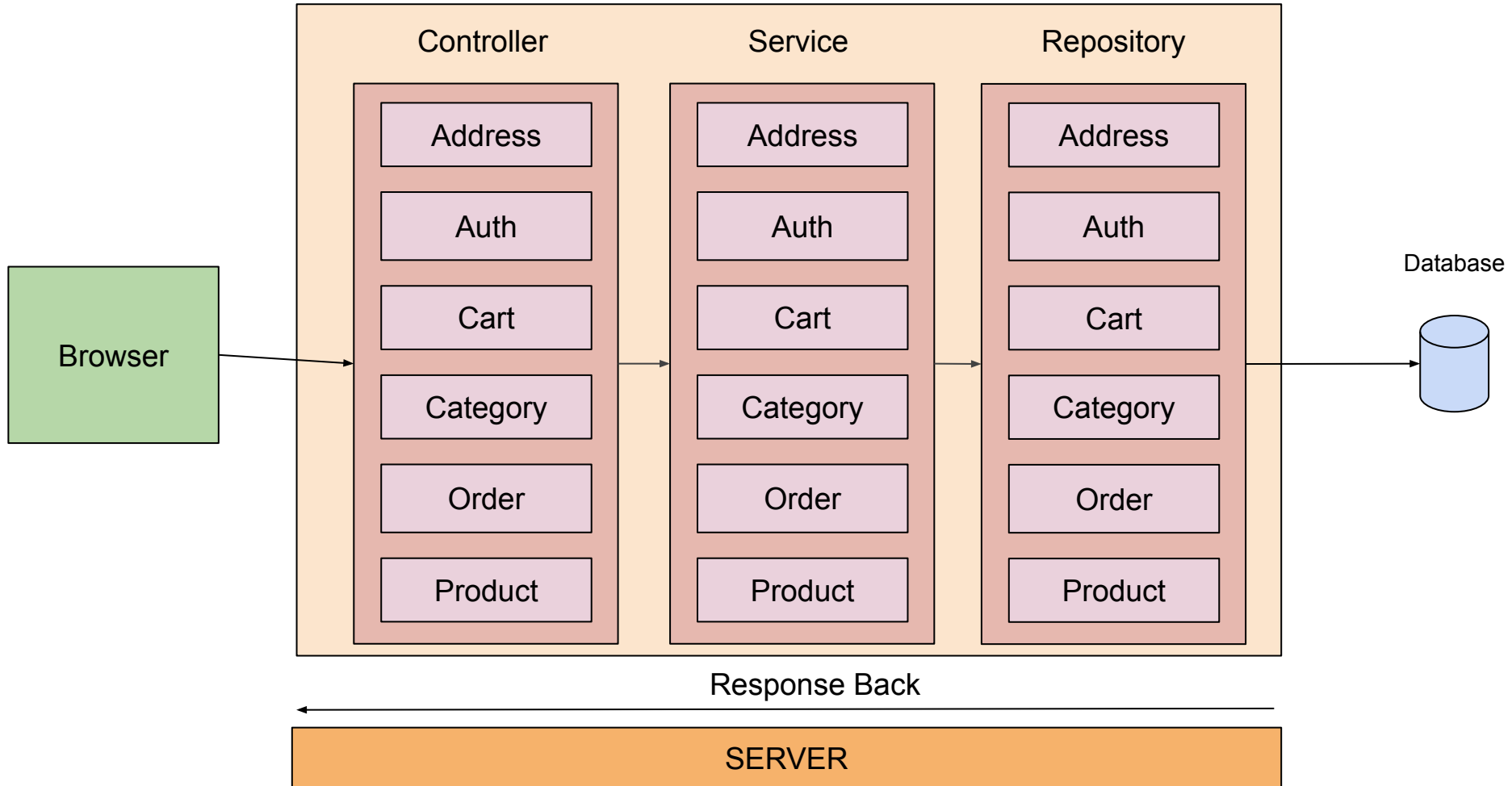
Thank you

Understanding the Product Module

Faisal Memon (EmbarkX)

OUR APPLICATION





API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Add Product	/api/admin/categories/{categoryId}/product	POST	Adds a new product to a category	Product (JSON)	categoryId (PathVariable)	ProductDTO (JSON), HttpStatus 201
Get All Products	/api/public/products	GET	Retrieves all products	-	pageNumber, pageSize, sortBy, sortOrder (RequestParams)	ProductResponse (JSON), HttpStatus 200
Get Products by Category	/api/public/categories/{categoryId}/products	GET	Retrieves products by category	-	categoryId (PathVariable), pageNumber, pageSize, sortBy, sortOrder (RequestParams)	ProductResponse (JSON), HttpStatus 200
Get Products by Keyword	/api/public/products/keyword/{keyword}	GET	Searches products by keyword	-	keyword (PathVariable), pageNumber, pageSize, sortBy, sortOrder (RequestParams)	ProductResponse (JSON), HttpStatus 302
Update Product	/api/products/{productId}	PUT	Updates an existing product	Product (JSON)	productId (PathVariable)	ProductDTO (JSON), HttpStatus 200

API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Update Product Image	/api/products/{productId}/image	PUT	Updates the image of a product	Multipart File (Form Data)	productId (PathVariable)	ProductDTO (JSON), HttpStatus 200
Delete Product	/api/admin/products/{productId}	DELETE	Deletes a product	-	productId (PathVariable)	String (Status Message), HttpStatus 200
Get Products by Seller	/api/seller/products	GET	Retrieves products by seller	-	pageNumber, pageSize, sortBy, sortOrder (RequestParams)	ProductResponse (JSON), HttpStatus 200
Get Product Count	/api/admin/products/count	GET	Retrieves product count	-	-	Long, HttpStatus 200

Thank you

Introduction to Spring Security

Faisal Memon (EmbarkX)

Security is important

Importance of Security

→ *Privacy Protection*

→ *Trust*

→ *Integrity*

→ *Compliance*

Role of Spring Security within the Spring Ecosystem

→ *Spring Framework*

→ *Spring Boot*

→ *Spring Data*

→ *Spring Security*

- *Authentication*

- *Authorization*

Authentication and Authorization

Authentication

Authentication is proving who you are.

Authorization

Authorization is about what you're allowed to do after you've proven who you are.

Authentication and Authorization

Authentication

Scanning your ID badge to confirm your identity as an employee.

Authorization

After confirming your identity, determining if you're permitted to enter certain restricted areas based on your job role or clearance level.

Key Security Principles

- *Least Privilege*
- *Secure by Design*
- *Fail-Safe Defaults*
- *Secure Communication*

Key Security Principles

→ *Input Validation*

→ *Auditing and Logging*

→ *Regular Updates and Patch Management*

Hashing

Faisal Memon (EmbarkX)

What is **Hashing**?

programming



Hashing

\$2a\$12\$JBtXfRbJBXD/lnskS70/3eaT3hTAp/lSzdm0xaFTv7dS3SQ8tNyLW

Hashing involves using
Algorithms

bcrypt involves using
salting

Salting helps increase
security

programming



XwZ78

SALT

Hashing

\$2a\$12\$JBtXfRbJBXD/lnskS70/3eaT3hTAp/lSzdm0xaFTv7dS3SQ8tNyLW

programming



7w139768b

SALT

Hashing

\$2a\$12\$xwHrcZF9BsDDoqF1JirbMu9h911nvqFUldZFcSvXE91MYTGwMLpYa

JWT Authentication

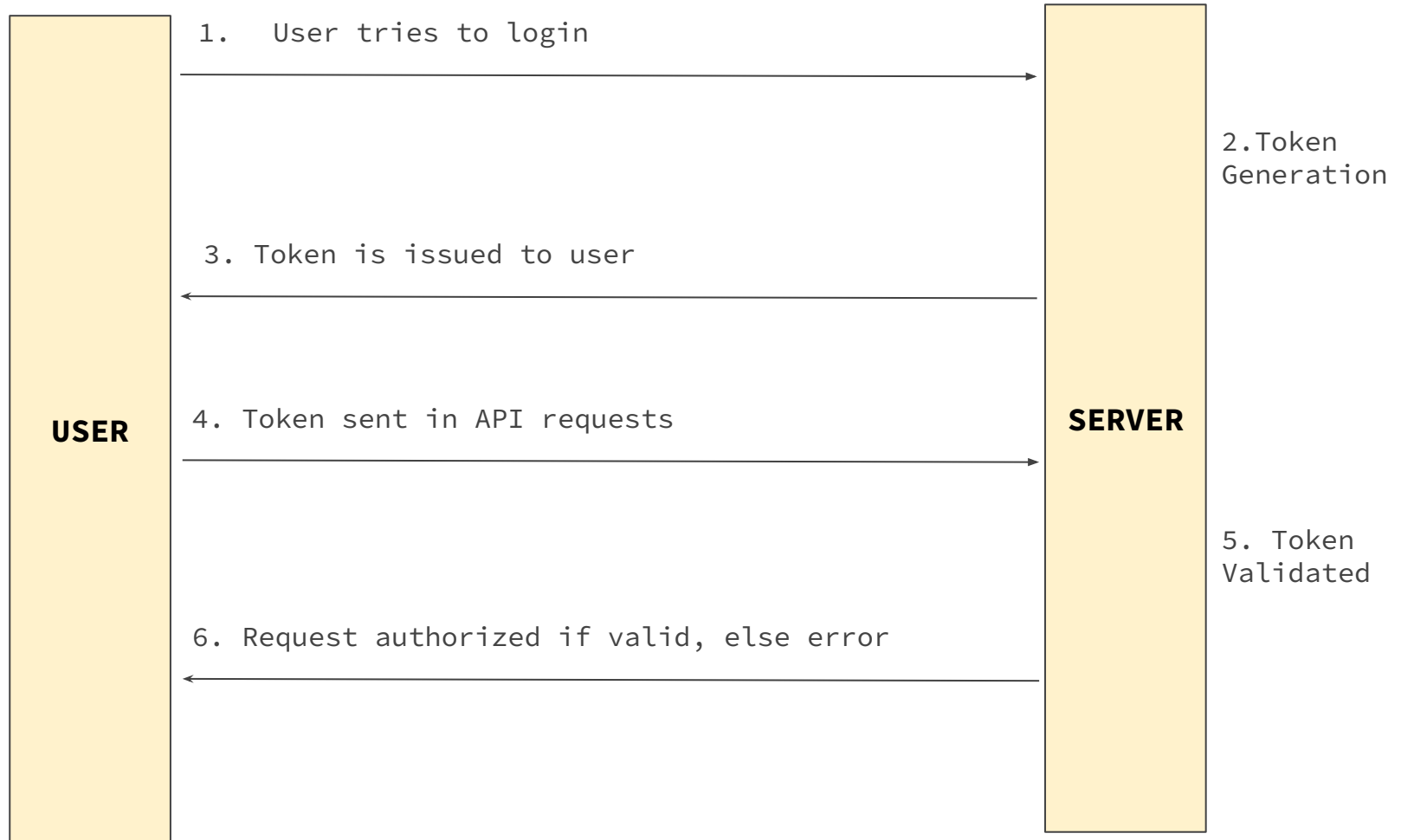
Faisal Memon (EmbarkX)

Without JWT

- *No advanced features like expiration time*
- *Can be decoded easily*
- *Should we go for “Custom token system”*

JWT = **JSON Web
Token**

JSON Web Tokens are
an **open, industry
standard**



How is Token sent

Tokens are sent using HTTP Authorization header

Format

Authorization: Bearer <token>

Header

PAYLOAD

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c

VERIFY SIGNATURE

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsIm1hdCI6MTcxNDYzMTIzMCwiZXhwIjoxNzE0TMxMjMwfQ.aPzkoasvY0Ryq2rtuCnVlZQ_pQBSo33oVc_yNi1ko-s
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256"  
}
```

PAYLOAD: DATA

```
{  
  "sub": "admin",  
  "iat": 1714631230,  
  "exp": 1714931230  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
) ☐ secret base64 encoded
```

Understanding Implementation of JWT

Faisal Memon (EmbarkX)

JwtUtils

AuthTokenFilter

AuthEntryPointJwt

SecurityConfig

Files we are going to need

JwtUtils

AuthTokenFilter

AuthEntryPointJwt

SecurityConfig

JwtUtils

→ Contains utility methods for generating, parsing, and validating JWTs.

→ Include generating a token from a username, validating a JWT, and extracting the username from a token.

AuthTokenFilter

→ Filters incoming requests to check for a valid JWT in the header, setting the authentication context if the token is valid.

→ Extracts JWT from request header, validates it, and configures the Spring Security context with user details if the token is valid.

JwtUtils

AuthTokenFilter

AuthEntryPointJwt

SecurityConfig



AuthEntryPointJwt

→ Provides custom handling for unauthorized requests, typically when authentication is required but not supplied or valid.

→ When an unauthorized request is detected, it logs the error and returns a JSON response with an error message, status code, and the path attempted.



SecurityConfig

→ Configures Spring Security filters and rules for the application

→ Sets up the security filter chain, permitting or denying access based on paths and roles. It also configures session management to stateless, which is crucial for JWT usage.

Authentication Controller

Faisal Memon (EmbarkX)

API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Sign In	<code>/signin</code>	POST	Authenticate a user	<code>LoginRequest</code> (JSON)	-	<code>UserInfoResponse</code> (JSON), <code>HttpStatus.OK</code>
Sign Up	<code>/signup</code>	POST	Register a new user	<code>SignupRequest</code> (JSON)	-	<code>MessageResponse</code> (JSON), <code>HttpStatus.OK</code>
Sign Out	<code>/signout</code>	POST	Sign out the user	-	-	<code>MessageResponse</code> (JSON), <code>HttpStatus.OK</code>
Current Username	<code>/username</code>	GET	Retrieve the username of the authenticated user	-	-	<code>String</code> (username), <code>HttpStatus.OK</code>
User Info	<code>/user</code>	GET	Retrieve user information	-	-	<code>UserInfoResponse</code> (JSON), <code>HttpStatus.OK</code>
All Sellers	<code>/sellers</code>	GET	Retrieve a paginated list of sellers	-	<code>pageNumber</code> (Query Parameter)	<code>UserResponse</code> (JSON), <code>HttpStatus.OK</code>

Jwt Cookie Based Auth

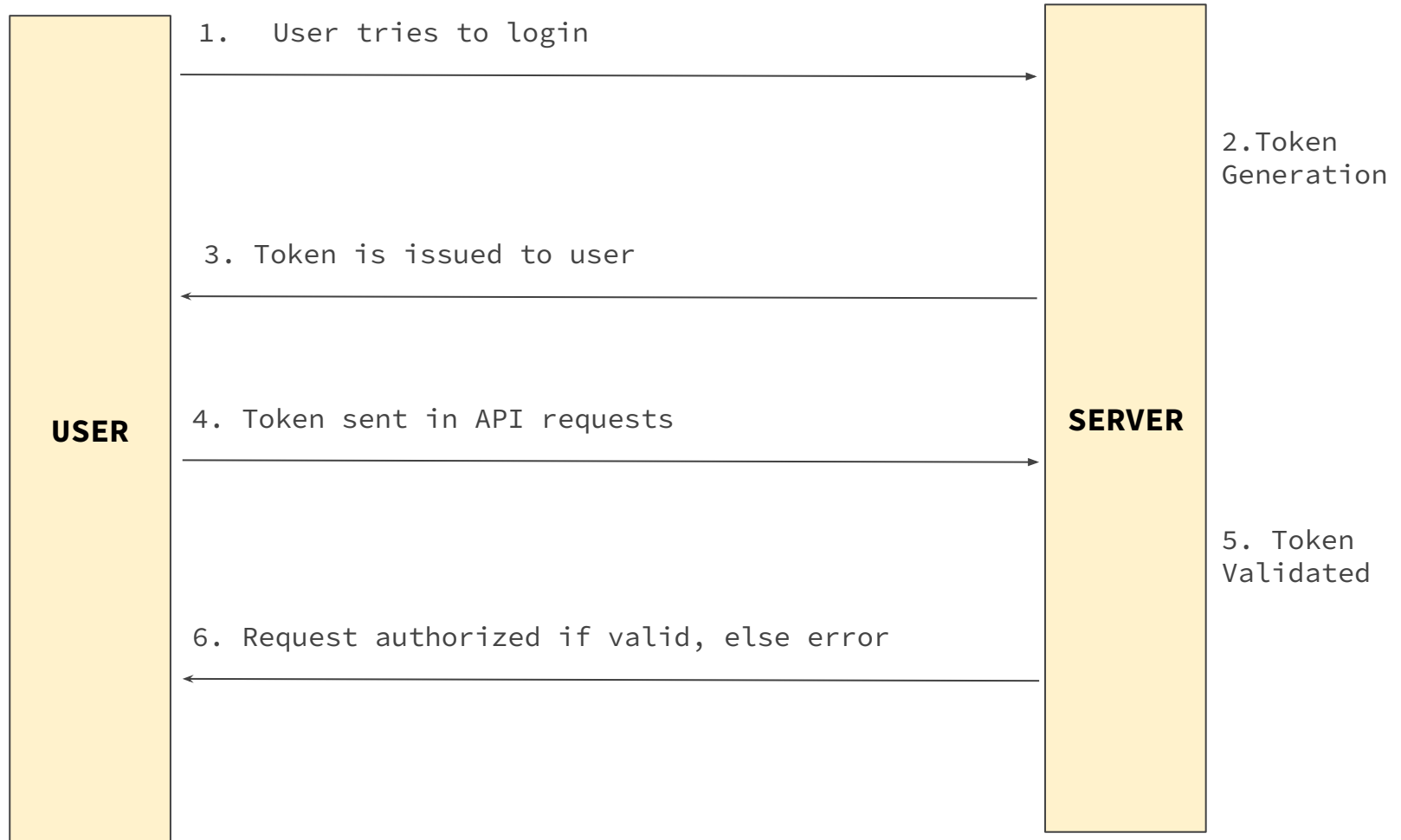
Faisal Memon (EmbarkX)

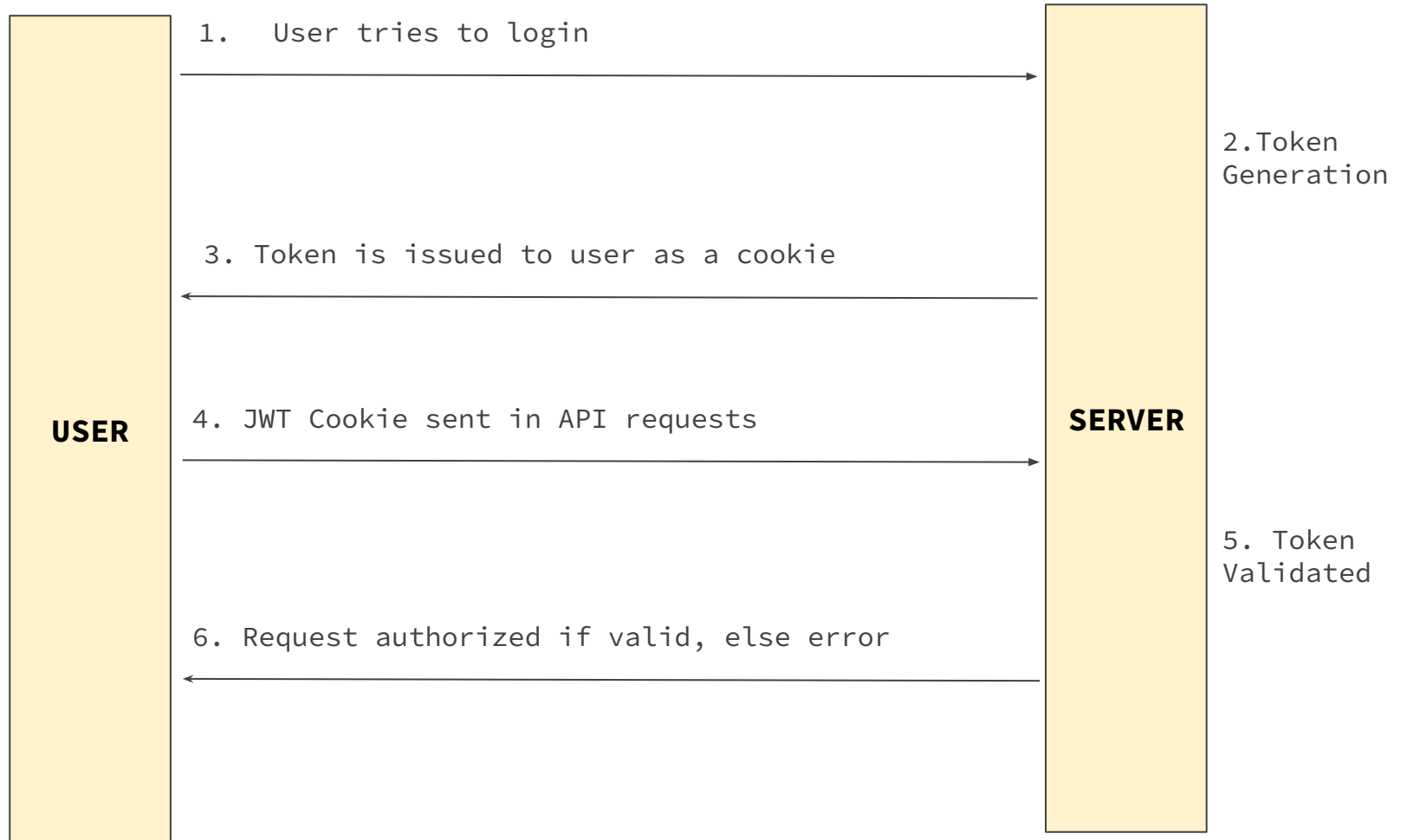
Bearer tokens need to be added
explicitly to the HTTP request

Format

Authorization: Bearer <token>

Browser will
automatically send
cookies





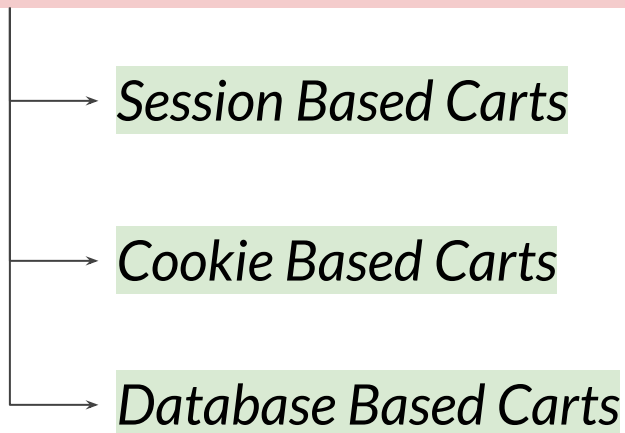


Thinking About Shopping Cart

Faisal Memon (EmbarkX)

Shopping Cart allows users to select and store items they wish to purchase

Ways to Implement Shopping Carts



Shopping Carts

Session-Based Carts

Cart's contents are stored in the user's session. If session expires, data is lost.

Cookie-Based Carts

Cart data is stored in cookies on the user's browser.

Database-Based Carts

Cart data is stored on the server side, within a database. This approach is scalable, secure, and allows for advanced features like cart recovery, detailed analytics, and cross-device accessibility.

We Will Use **Database Based** Carts

Advantages of Database-Based Carts

→ *Persistence and Reliability*

→ *Scalability*

→ *Enhanced Features*

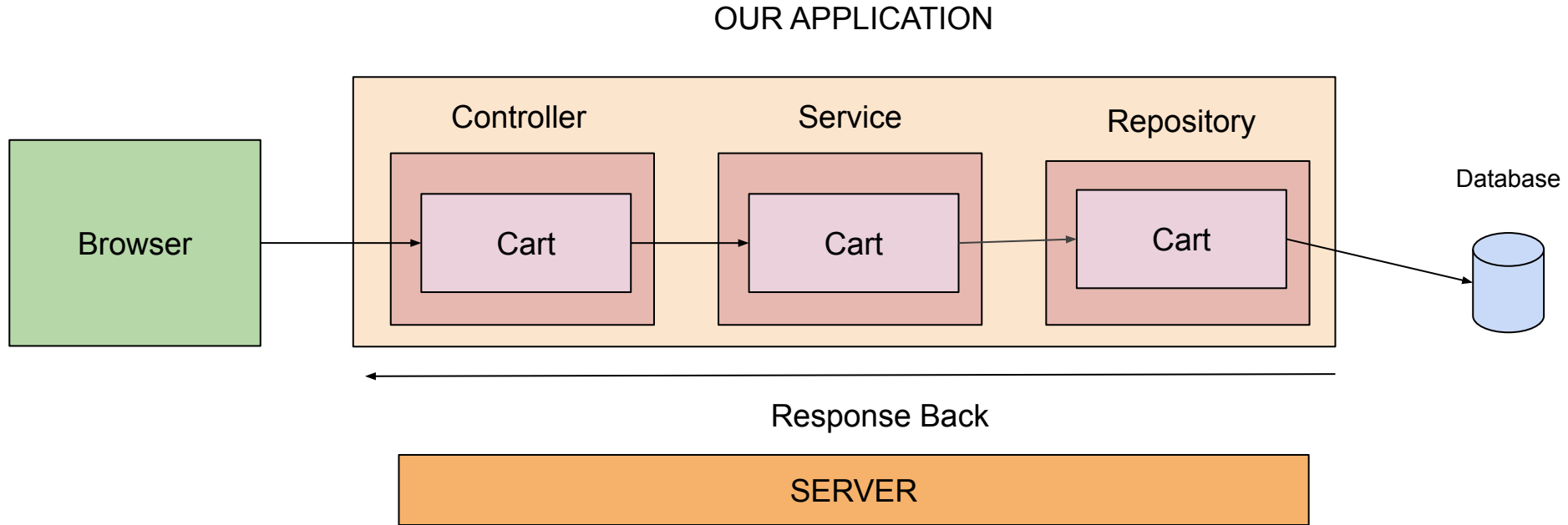
→ *Security*

→ *User Experience*

Thank you

Designing Cart Module

Faisal Memon (EmbarkX)



API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Add Product to Cart	/api/carts/products/{productId}/quantity/{quantity}	POST	Adds a specified product and quantity to the user's cart.	None	productId: Long, quantity: Integer	CartDTO (JSON)
Get All Carts	/api/carts	GET	Retrieves a list of all carts.	None	None	List of CartDTO (JSON)
Get User's Cart	/api/carts/users/cart	GET	Retrieves the cart of the logged-in user.	None	None	CartDTO (JSON)
Update Product Quantity	/api/cart/products/{productId}/quantity/{operation}	PUT	Updates the quantity of a specific product in the cart.	None	productId: Long, operation: String	CartDTO (JSON)
Delete Product from Cart	/api/carts/{cartId}/product/{productId}	DELETE	Removes a specific product from the user's cart.	None	cartId: Long, productId: Long	String (Status message)

API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Add Product to Cart	/api/carts/products/{productId}/quantity/{quantity}	POST	Adds a specified product and quantity to the user's cart.	None	productId: Long, quantity: Integer	CartDTO (JSON)
Get All Carts	/api/carts	GET	Retrieves a list of all carts.	None	None	List of CartDTO (JSON)
Get User's Cart	/api/carts/users/cart	GET	Retrieves the cart of the logged-in user.	None	None	CartDTO (JSON)
Update Product Quantity	/api/cart/products/{productId}/quantity/{operation}	PUT	Updates the quantity of a specific product in the cart.	None	productId: Long, operation: String	CartDTO (JSON)
Delete Product from Cart	/api/carts/{cartId}/product/{productId}	DELETE	Removes a specific product from the user's cart.	None	cartId: Long, productId: Long	String (Status message)

API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Add Product to Cart	/api/carts/products/{productId}/quantity/{quantity}	POST	Adds a specified product and quantity to the user's cart.	None	productId: Long, quantity: Integer	CartDTO (JSON)
Get All Carts	/api/carts	GET	Retrieves a list of all carts.	None	None	List of CartDTO (JSON)
Get User's Cart	/api/carts/users/cart	GET	Retrieves the cart of the logged-in user.	None	None	CartDTO (JSON)
Update Product Quantity	/api/cart/products/{productId}/quantity/{operation}	PUT	Updates the quantity of a specific product in the cart.	None	productId: Long, operation: String	CartDTO (JSON)
Delete Product from Cart	/api/carts/{cartId}/product/{productId}	DELETE	Removes a specific product from the user's cart.	None	cartId: Long, productId: Long	String (Status message)

API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Add Product to Cart	/api/carts/products/{productId}/quantity/{quantity}	POST	Adds a specified product and quantity to the user's cart.	None	productId: Long, quantity: Integer	CartDTO (JSON)
Get All Carts	/api/carts	GET	Retrieves a list of all carts.	None	None	List of CartDTO (JSON)
Get User's Cart	/api/carts/users/cart	GET	Retrieves the cart of the logged-in user.	None	None	CartDTO (JSON)
Update Product Quantity	/api/cart/products/{productId}/quantity/{operation}	PUT	Updates the quantity of a specific product in the cart.	None	productId: Long, operation: String	CartDTO (JSON)
Delete Product from Cart	/api/carts/{cartId}/product/{productId}	DELETE	Removes a specific product from the user's cart.	None	cartId: Long, productId: Long	String (Status message)

API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Add Product to Cart	/api/carts/products/{productId}/quantity/{quantity}	POST	Adds a specified product and quantity to the user's cart.	None	productId: Long, quantity: Integer	CartDTO (JSON)
Get All Carts	/api/carts	GET	Retrieves a list of all carts.	None	None	List of CartDTO (JSON)
Get User's Cart	/api/carts/users/cart	GET	Retrieves the cart of the logged-in user.	None	None	CartDTO (JSON)
Update Product Quantity	/api/cart/products/{productId}/quantity/{operation}	PUT	Updates the quantity of a specific product in the cart.	None	productId: Long, operation: String	CartDTO (JSON)
Delete Product from Cart	/api/carts/{cartId}/product/{productId}	DELETE	Removes a specific product from the user's cart.	None	cartId: Long, productId: Long	String (Status message)

API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Add Product to Cart	/api/carts/products/{productId}/quantity/{quantity}	POST	Adds a specified product and quantity to the user's cart.	None	productId: Long, quantity: Integer	CartDTO (JSON)
Get All Carts	/api/carts	GET	Retrieves a list of all carts.	None	None	List of CartDTO (JSON)
Get User's Cart	/api/carts/users/cart	GET	Retrieves the cart of the logged-in user.	None	None	CartDTO (JSON)
Update Product Quantity	/api/cart/products/{productId}/quantity/{operation}	PUT	Updates the quantity of a specific product in the cart.	None	productId: Long, operation: String	CartDTO (JSON)
Delete Product from Cart	/api/carts/{cartId}/product/{productId}	DELETE	Removes a specific product from the user's cart.	None	cartId: Long, productId: Long	String (Status message)

API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Add Product to Cart	/api/carts/products/{productId}/quantity/{quantity}	POST	Adds a specified product and quantity to the user's cart.	None	productId: Long, quantity: Integer	CartDTO (JSON)
Get All Carts	/api/carts	GET	Retrieves a list of all carts.	None	None	List of CartDTO (JSON)
Get User's Cart	/api/carts/users/cart	GET	Retrieves the cart of the logged-in user.	None	None	CartDTO (JSON)
Update Product Quantity	/api/cart/products/{productId}/quantity/{operation}	PUT	Updates the quantity of a specific product in the cart.	None	productId: Long, operation: String	CartDTO (JSON)
Delete Product from Cart	/api/carts/{cartId}/product/{productId}	DELETE	Removes a specific product from the user's cart.	None	cartId: Long, productId: Long	String (Status message)

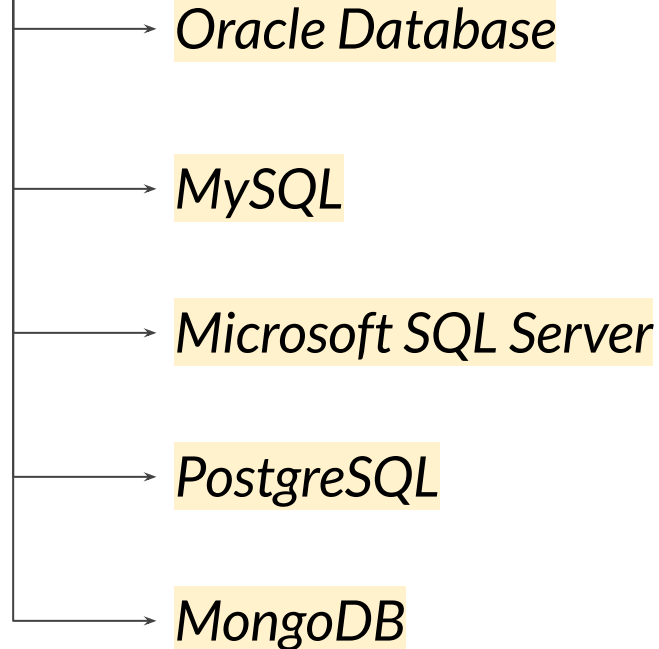
Different Databases and Magic of JPA

Faisal Memon (EmbarkX)

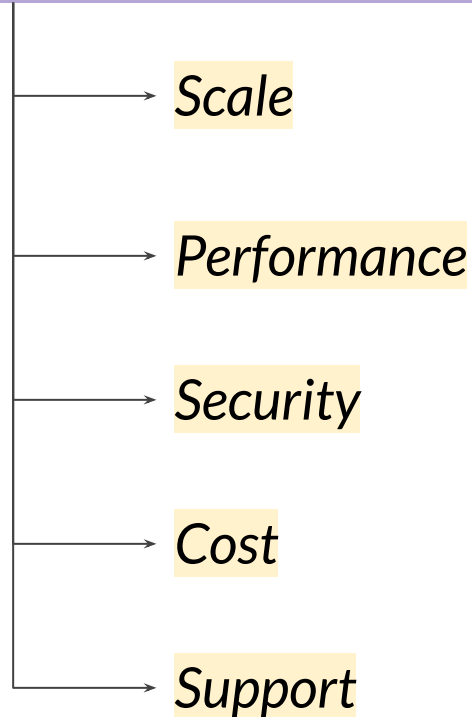
Database Vendor

A database vendor is a company or organization that develops and maintains a database management system

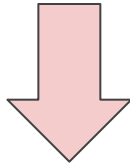
Types of Vendors



Choosing the right vendor



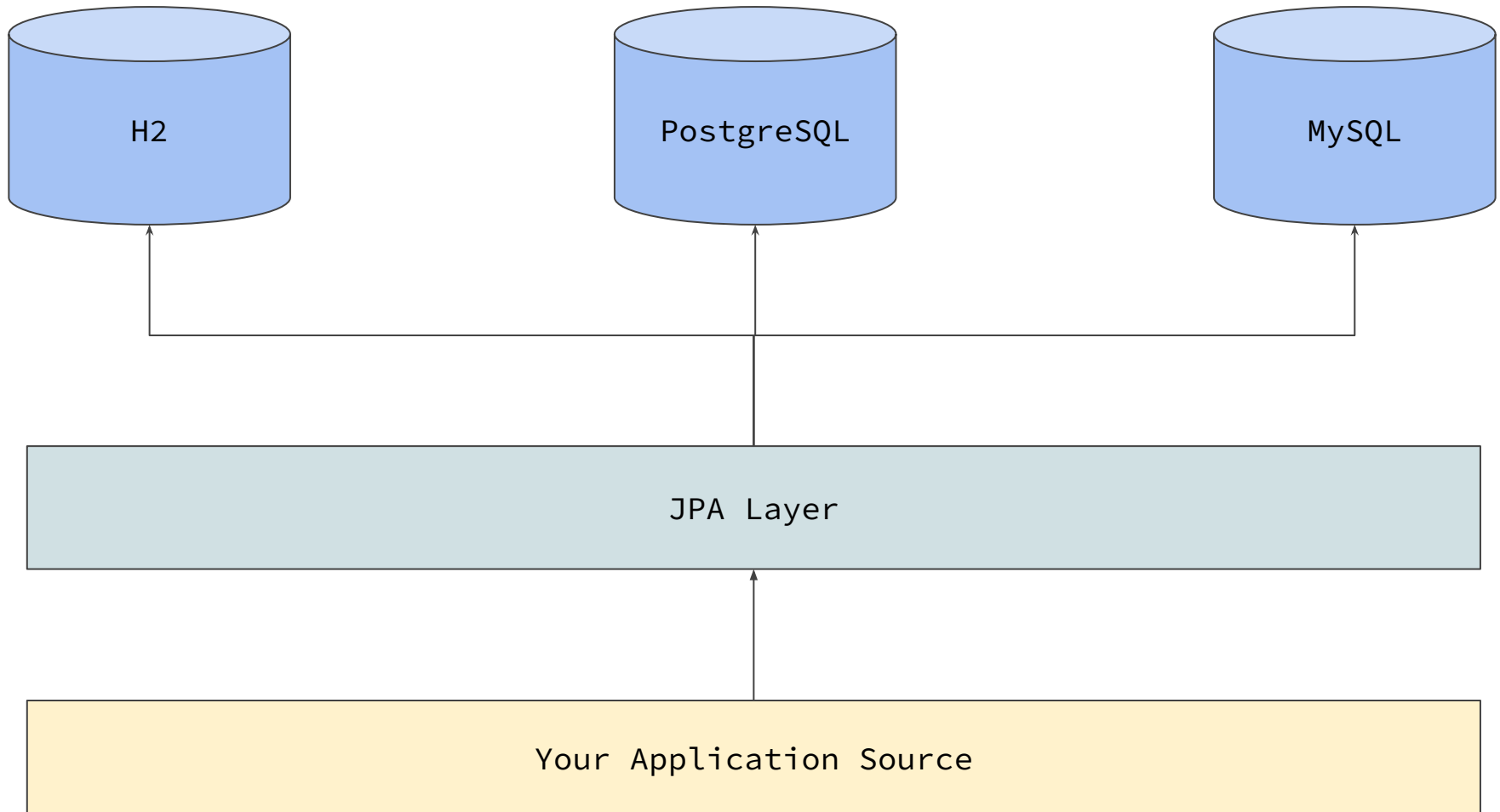
```
class Category {  
    Long categoryId;  
    String categoryName;  
}
```



categoryId	categoryName
1	Senior Software Engineer

Advantages of using JPA

- *Easy and Simple*
- *Makes querying easier*
- *Allows to save and update objects*
- *Easy integration with Spring Boot*



Configuration for MySQL

```
spring.datasource.url=jdbc:mysql://localhost:3306/ecommerce  
spring.datasource.username=root  
spring.datasource.password=<your-password>
```

```
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

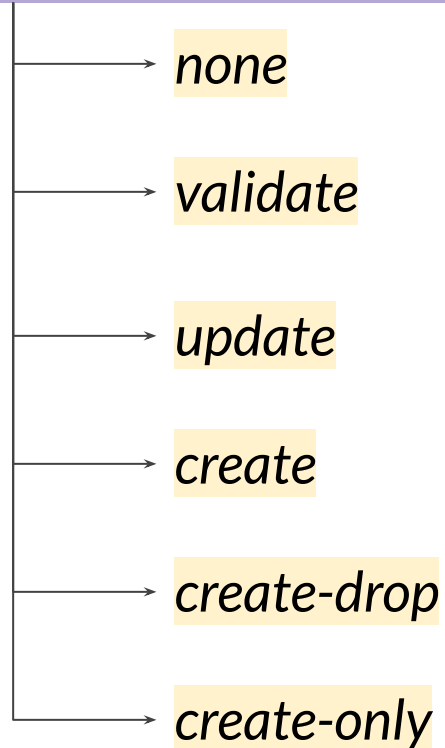
Thank you

Database Schema Management

Faisal Memon (EmbarkX)

```
spring.jpa.hibernate.ddl-auto=update
```

Possible Values



Value	Description	When to Use
none	Hibernate does not perform any schema generation or modification.	Use this in production environments where schema changes are managed manually or through migrations.
validate	Hibernate validates the schema against the entities. It checks if the tables and columns in the database schema match the entities. If there is a mismatch, an exception is thrown, and the application fails to start.	Use this in production environments to ensure the schema matches the entity mappings without making changes.
update	Hibernate updates the database schema to match the entities. It adds new columns and tables as necessary, but it does not remove or modify existing columns and tables.	Use this in development and testing environments where you want the schema to evolve with the entity mappings without losing data.

Value	Description	When to Use
create	Hibernate drops the existing schema (tables) and creates a new schema based on the entity mappings. This means that all data in the existing tables will be lost.	Use this in development and testing environments where you need to start with a fresh schema on every run.
create-drop	Similar to create, but in addition, Hibernate drops the schema when the SessionFactory is closed, typically when the application shuts down.	Use this in unit tests or short-lived applications where you need a fresh schema on every run and don't need to keep the data after the application ends.
create-only	Hibernate creates the schema, but does not drop it when the session factory is closed.	Use this when you need to create the schema initially but want to handle cleanup or further management manually.

Thinking About Managing Addresses

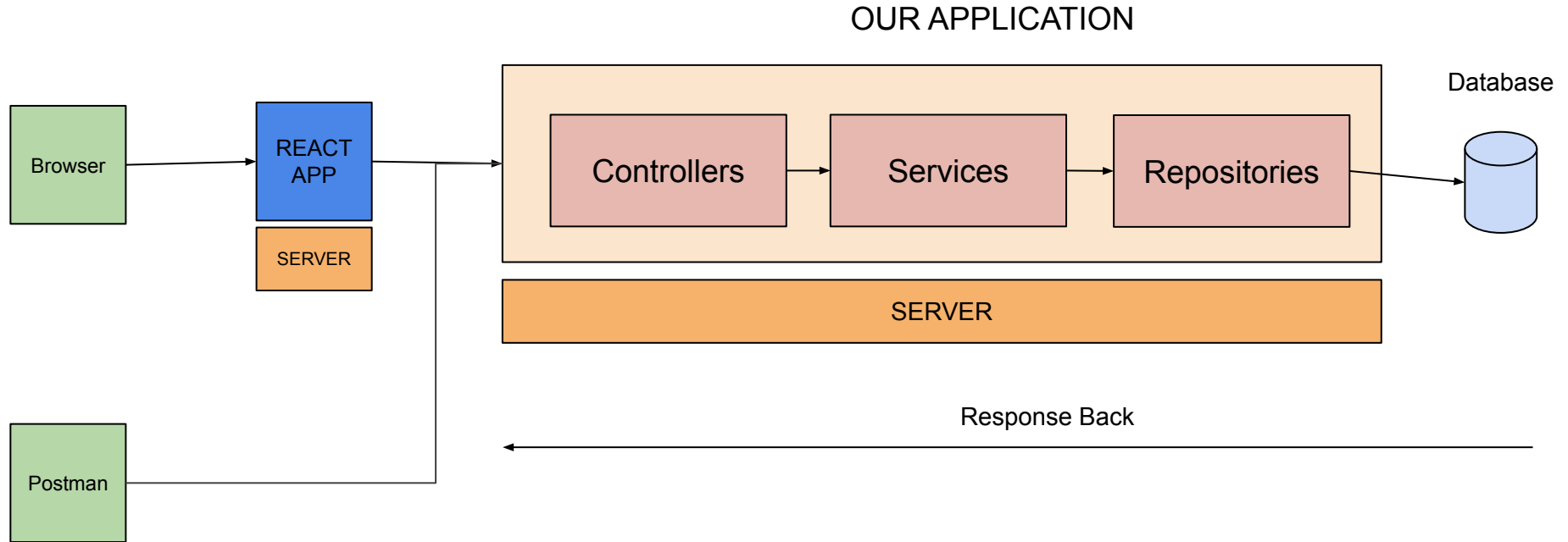
Faisal Memon (EmbarkX)

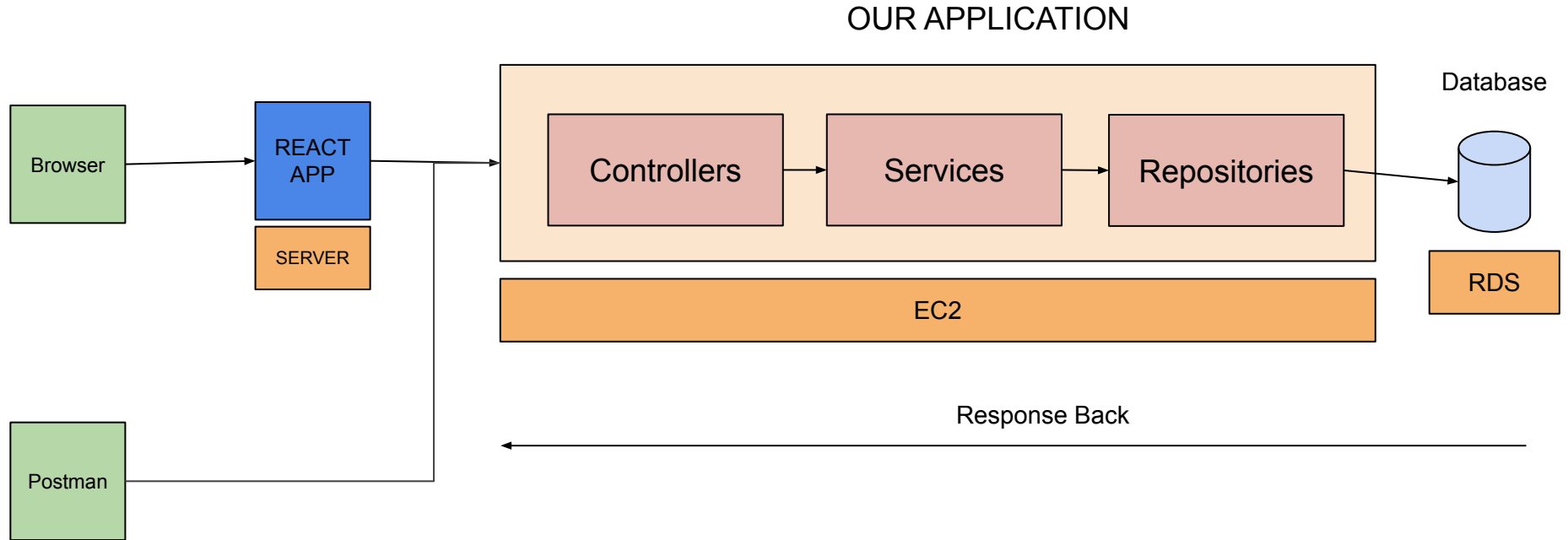
API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Create Address	/addresses	POST	Create a new address	AddressDTO	None	AddressDTO with HttpStatus.CREATED
Get All Addresses	/addresses	GET	Retrieve all addresses	None	None	List of AddressDTO with HttpStatus.OK
Get Address by ID	/addresses/{addressId}	GET	Retrieve an address by its ID	None	Path: addressId (Long)	AddressDTO with HttpStatus.OK
Get Address by User	/users/addresses	GET	Retrieve the logged-in user's address	None	None	AddressDTO with HttpStatus.OK

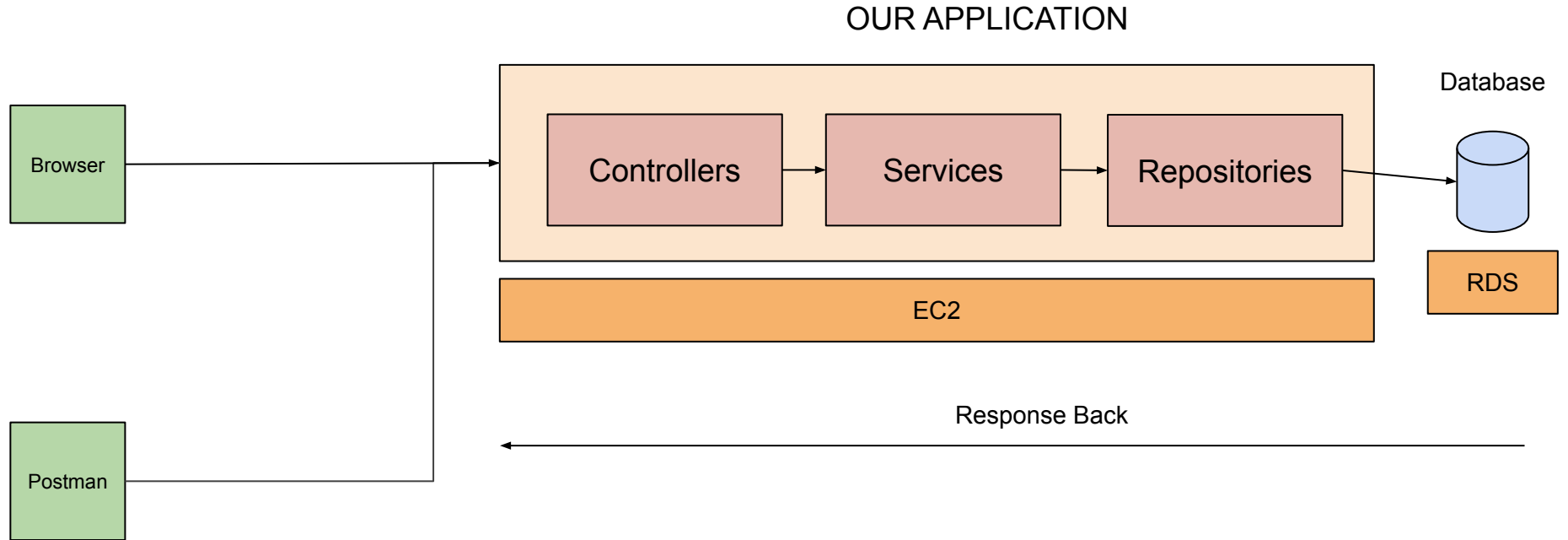
API Name	Endpoint	Method	Purpose	Request Body	Request Parameters	Response
Update Address	/addresses/{addressId}	PUT	Update an existing address by its ID	Address	Path: addressId (Long)	AddressDTO with HttpStatus.OK
Delete Address	/addresses/{addressId}	DELETE	Delete an address by its ID	None	Path: addressId (Long)	Status message with HttpStatus.OK

Understanding Deployments

Faisal Memon (EmbarkX)







If you think this course helped you, please do help provide an **honest rating and review** of the course. Your insights help us **improve and provide better content** for future learners.

We appreciate your **support** and look forward to hearing your **thoughts!**