

Repository

Basic of JDBC

```
// 1. Load jdbc driver
Class.forName("postgresql");

// 2. Create connection
Connection connection = DriverManager.getConnection("", "", "");

// 3. Prepared Statement
String sql = "SELECT * FROM TABLE WHERE name=?";
PreparedStatement pStmt = connection.prepareStatement(sql);

// 4. Query
ResultSet resultSet = pStmt.executeQuery();
while(resultSet.next()) {

}

// 5. Release resource
if(resultSet != null) {
    resultSet.close();
    resultSet = null;
}
```

Framework !!

```
// 1. Load jdbc driver  
Class.forName("postgresql");  
// 2. Create connection  
Connection connection = DriverManager.getConnection("","","");
```

Manage by Framework

```
// 3. Prepared Statement  
String sql = "SELECT * FROM TABLE WHERE name=?";  
PreparedStatement pStmt = connection.prepareStatement(sql);  
  
// 4. Query  
ResultSet resultSet = pStmt.executeQuery();  
while(resultSet.next()) {  
  
}
```

```
// 5. Release resource  
if(resultSet != null) {  
    resultSet.close();  
    resultSet = null;  
}
```

Manage by Framework

Working with Database ?

Production

Application



PostgreSQL

Testing

Application

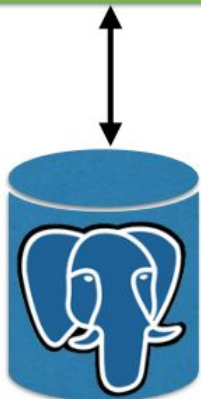


H2

Working with Database ?

Production

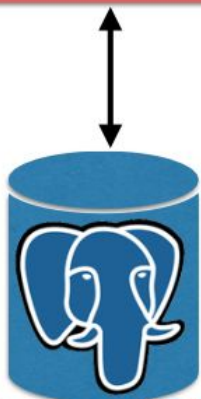
Application



PostgreSQL

Testing

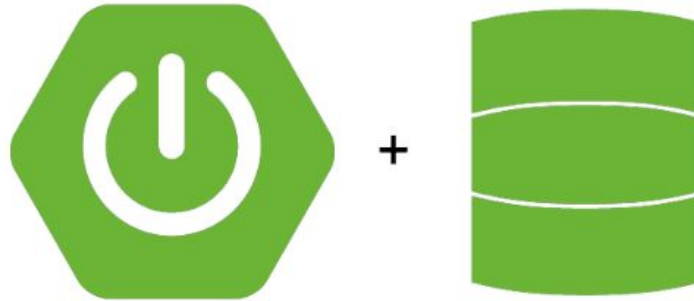
Application



PostgreSQL

Working with repository

We're using Spring Data



<https://spring.io/projects/spring-data>

Spring Data

JDBC

JPA

MongoDB

Redis

more ...

Working with Spring Data JPA

Working with Database

Production

Application



PostgreSQL

Testing

Application



H2

Modify pom.xml

Add library of Spring Data JPA, PostgreSQL, H2



The screenshot shows the 'Dependencies' section of the Spring Start web application. It features a search bar with the placeholder text 'Search dependencies to add' and a dropdown menu showing 'Web, Security, JPA, Actuator, Devtools...'. To the right, under the heading 'Dependencies selected', three database-related dependencies are listed: JPA [SQL], PostgreSQL [SQL], and H2 [SQL]. Each entry includes a brief description of its function.

Dependencies	Search dependencies to add	Dependencies selected
	<i>Web, Security, JPA, Actuator, Devtools...</i>	JPA [SQL] Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate
		PostgreSQL [SQL] PostgreSQL JDBC driver
		H2 [SQL] H2 database (with embedded support)

<https://start.spring.io/>

Modify pom.xml

H2 for testing

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>
```

```
<dependency>  
  <groupId>com.h2database</groupId>  
  <artifactId>h2</artifactId>  
  <scope>test</scope>  
</dependency>
```

```
<dependency>  
  <groupId>org.postgresql</groupId>  
  <artifactId>postgresql</artifactId>  
  <scope>runtime</scope>  
</dependency>
```

Modify pom.xml

PostgreSQL for production

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>
```

```
<dependency>  
  <groupId>com.h2database</groupId>  
  <artifactId>h2</artifactId>  
  <scope>test</scope>  
</dependency>
```

```
<dependency>  
  <groupId>org.postgresql</groupId>  
  <artifactId>postgresql</artifactId>  
  <scope>runtime</scope>  
</dependency>
```

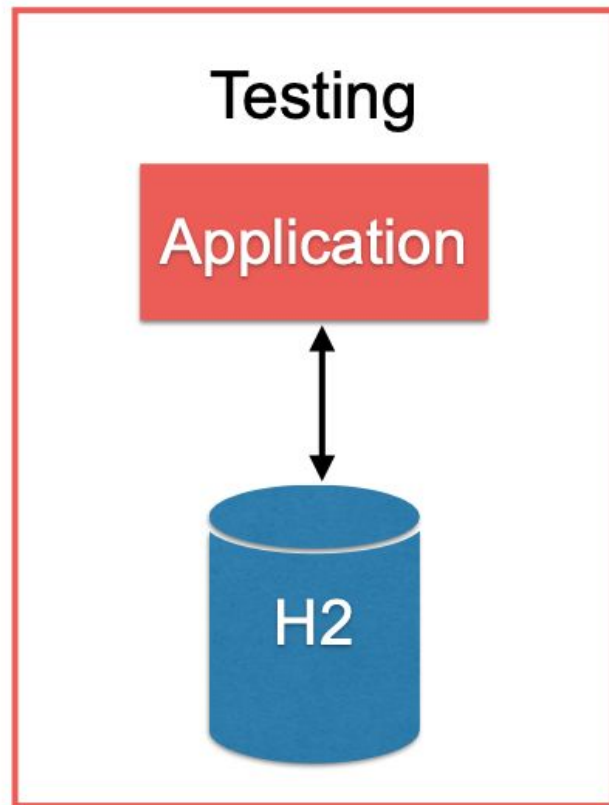
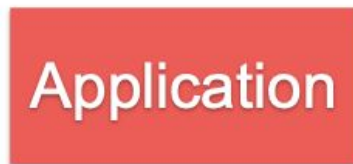
Start in testing scope

Production

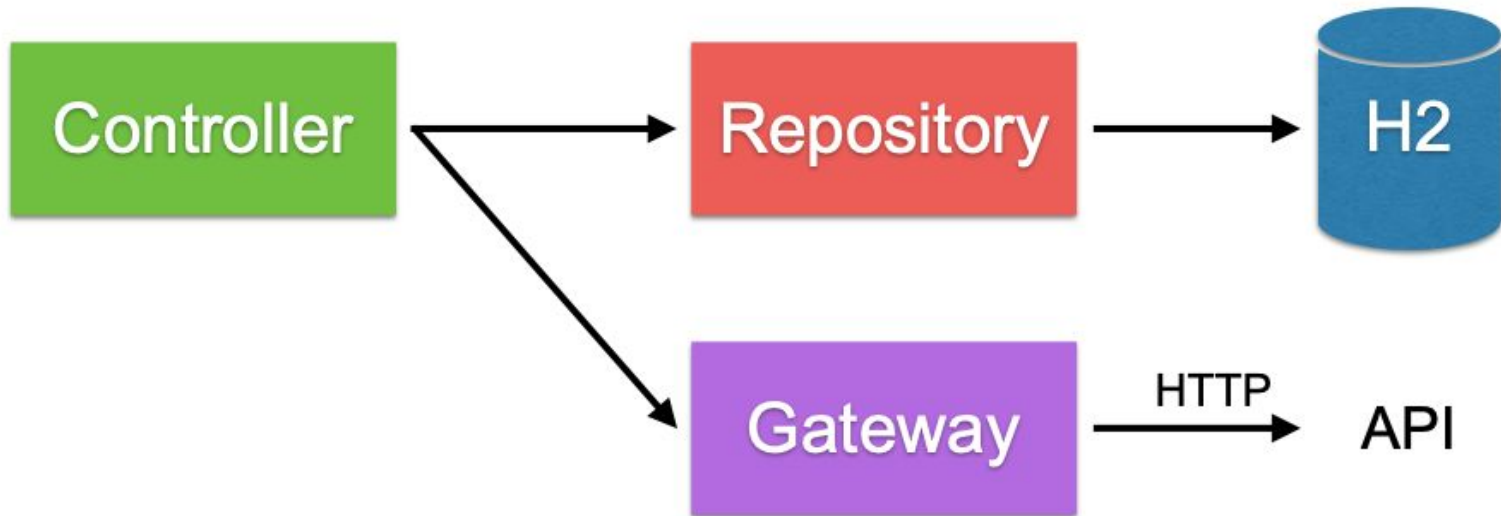


PostgreSQL

Testing

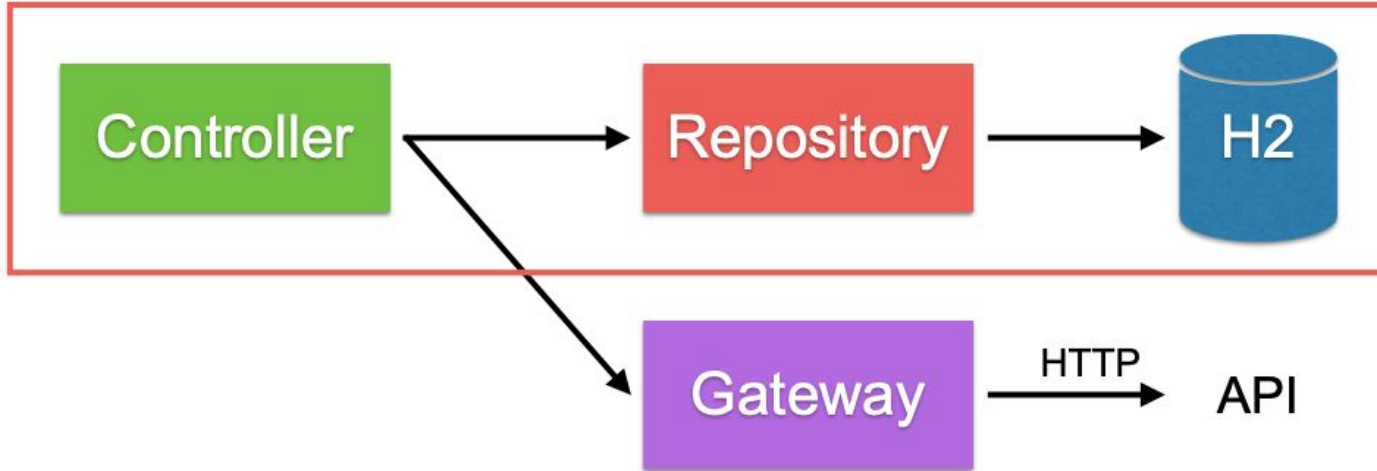


Use cases



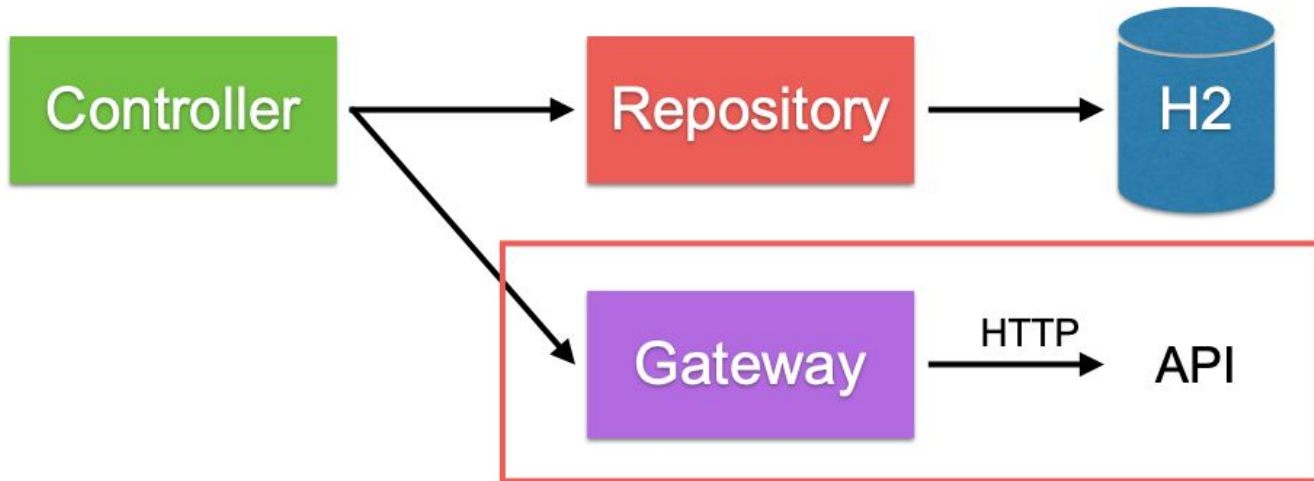
Use case 1

Working with repository



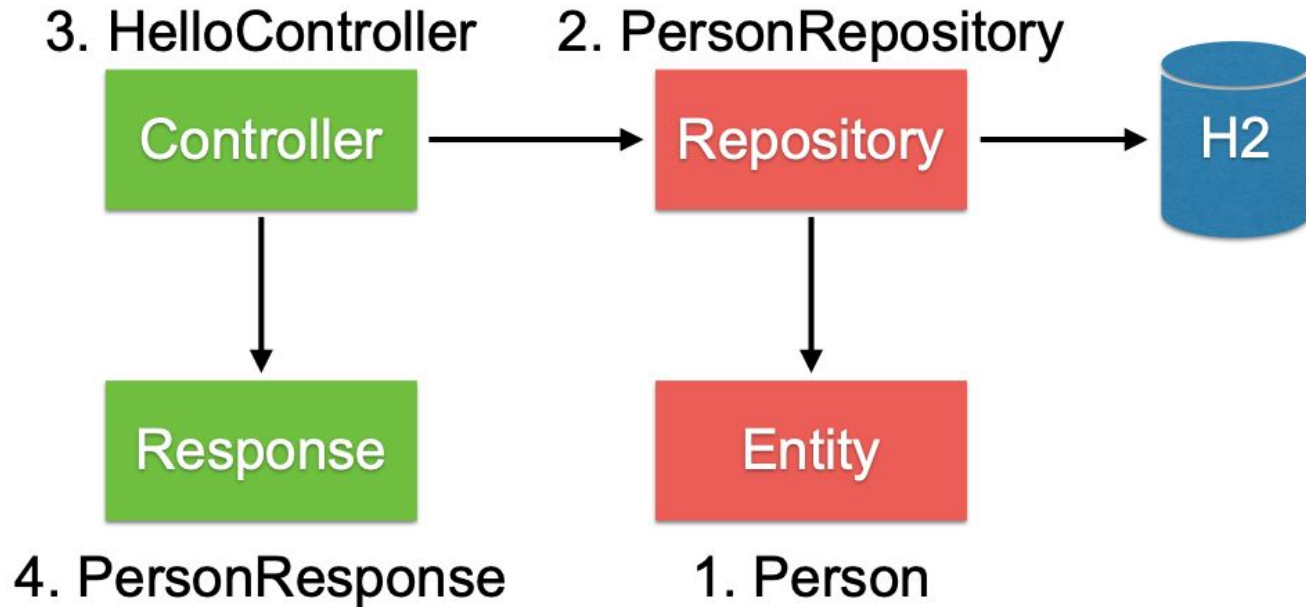
Use case 2

Working with API



Use case 1

Working with repository



1. Create Entity class

In package person

```
@Entity
public class Person {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    private String firstName;
    private String lastName;

    public Person() {
    }
```

2. Create repository with JPA

PersonRepository.java

```
import java.util.Optional;

import org.springframework.data.repository.CrudRepository;

public interface PersonRepository
    extends CrudRepository<Person, Long> {

    Optional<Person> findByLastName(String lastName);

}
```

2. Create repository with JPA

PersonRepository.java

```
import java.util.Optional;

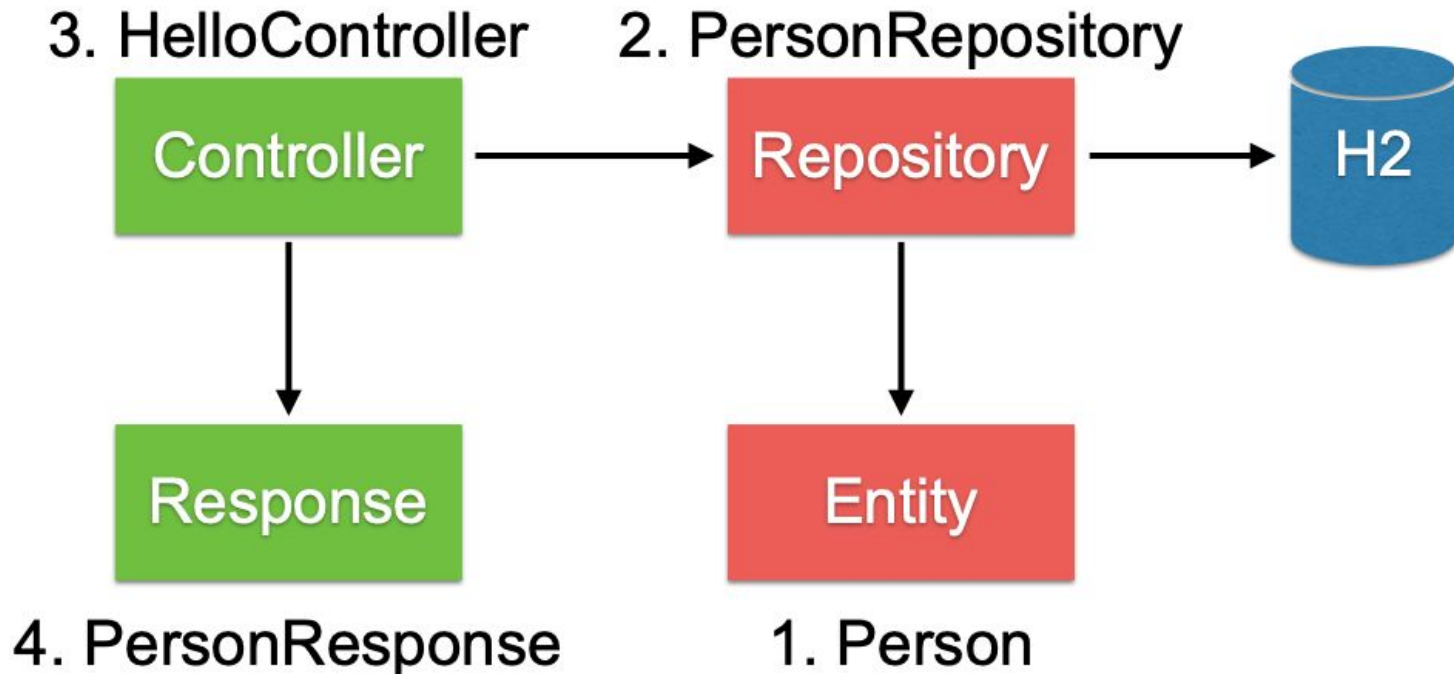
import org.springframework.data.repository.CrudRepository

public interface PersonRepository
    extends CrudRepository<Person, Long> {
    Optional<Person> findByLastName(String lastName);
}
```

*SELECT * FROM Person WHERE LastName=?*

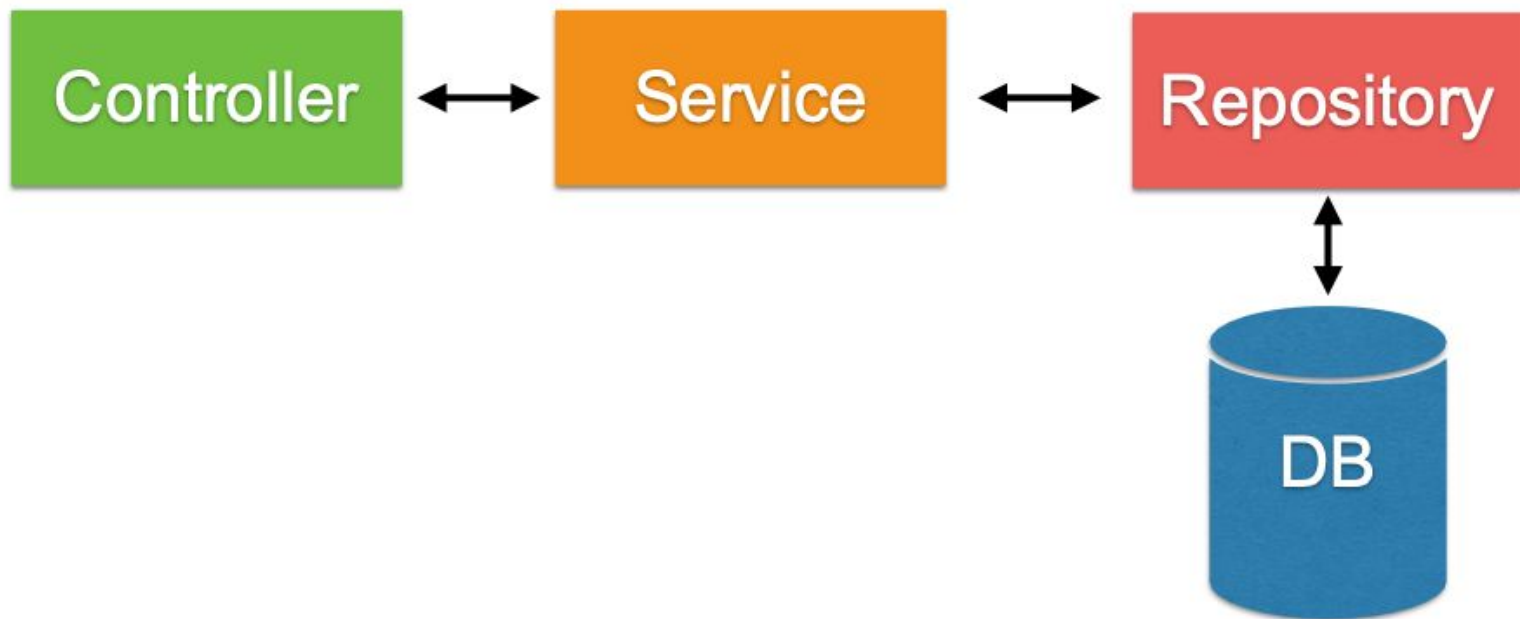
Use case 1

Integrate repository with controller

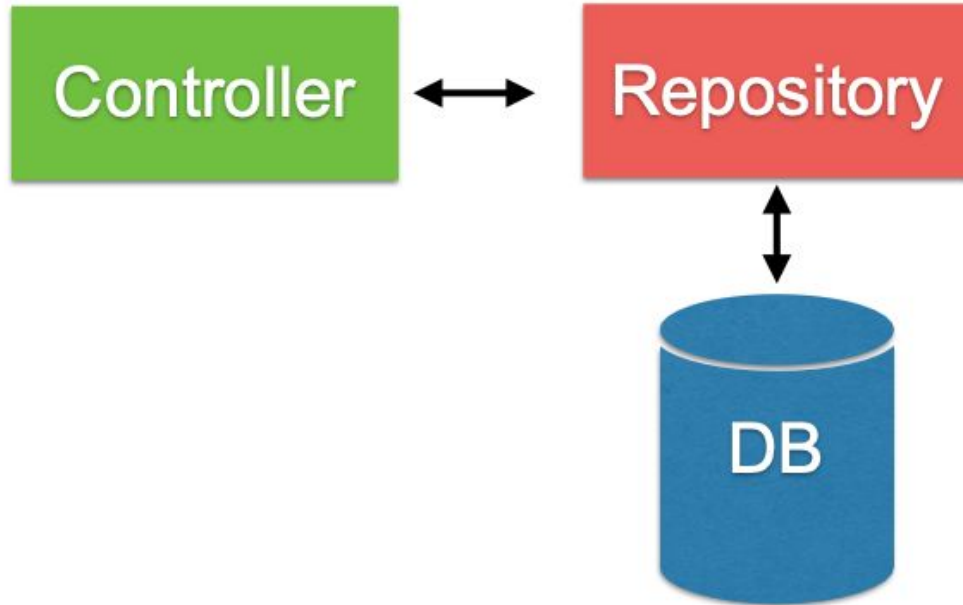


**Integrate repository with
service/controller**

Service use repository ?



Controller use repository ?



Controller call repository

Create HelloController.java

```
@RestController
public class HelloController {

    private final PersonRepository personRepository;

    @Autowired
    public HelloController(final PersonRepository personRepository) {
        this.personRepository = personRepository;
    }
}
```

Controller call repository

Create HelloController.java

```
@GetMapping("/hello/{lastName}")
public HelloResponse hello(@PathVariable final String lastName) {

    Optional<Person> foundPerson
        = personRepository.findByLastName(lastName);

    return foundPerson
        .map(person ->
            new HelloResponse(person.getFirstName(),
                              person.getLastName()))
        .orElseThrow(() -> new RuntimeException());
}
```

Run spring boot

```
$mvnw spring-boot:run
```

Fix !!!

Modify src/main/resources/application.properties

server.port=8088

spring.datasource.url=jdbc:postgresql://127.0.0.1:15432/postgres

spring.datasource.username=testuser

spring.datasource.password=password

spring.datasource.platform=POSTGRESQL

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

Start database server !!

Fix !!!

Modify pom.xml

Delete or comment postgresql dependency

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>

<!-- <dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency> -->
```

Run spring boot

```
$mvnw spring-boot:run
```

← → ↻ 🏠 ⓘ localhost:8080/lotto

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Sep 05 12:08:49 ICT 2023

There was an unexpected error (type=Method Not Allowed, status=405).

Initial data in database

Initial database

Using @PostConstruct

```
@PostConstruct
public void initData() {
    Account account1 = new Account();
    account1.setAccountId("01");
    accountRepository.save(account1);
    Account account2 = new Account();
    account2.setAccountId("02");
    accountRepository.save(account2);
}
```


Initial database

Schema (resources/schema.sql)

Data (resources/data.sql)

Schema.sql

```
CREATE TABLE account(  
  id BIGINT AUTO_INCREMENT PRIMARY KEY,  
  account_Id VARCHAR(16) NOT NULL UNIQUE,  
  mobile_No VARCHAR(10),  
  name VARCHAR(50),  
  account_Type CHAR(2)  
);
```

Data.sql

```
INSERT INTO account (account_Id) VALUES ('01');  
INSERT INTO account (account_Id) VALUES ('02');
```

Initial database

Disable auto generate DDL from JPA in file
application.yml

```
spring:  
  jpa:  
    show-sql: true  
    hibernate:  
      ddl-auto: none
```

Initial database

Problem with naming strategy !!

```
spring:
  jpa:
    show-sql: true
    hibernate:
      ddl-auto: none
      naming:
        physical-strategy:
org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy
        implicit-strategy:
org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy
```

Run and see from logging

Execute file `schema.sql` and `data.sql`

Run spring boot

```
$mvnw spring-boot:run
```

10 minutes

Break 14:50

Error handling

Error handling

```
@Service
public class UserService {

    private AccountRepository accountRepository;

    @Autowired
    public UserService(AccountRepository accountRepository) {
        this.accountRepository = accountRepository;
    }

    public Account getAccount(int id) {
        Optional<Account> account = accountRepository.findById(id);
        if(account.isPresent()) {
            return account.get();
        }
        throw new MyAccountNotFoundException(
            String.format("Account id=[%d] not found", id));
    }
}
```


MyAccountNotFoundException

```
public class MyAccountNotFoundException  
    extends RuntimeException {
```

```
    public MyAccountNotFoundException(String message) {  
        super(message);  
    }
```

```
}
```

Response Status

404 = Not Found

Status	Description
400	Request body doesn't meet API spec
401	Authentication/Authorization fail
403	User can't perform the operation
404	Resource does not exist
405	Unsupported operation
500	Error on server

Handling error in Spring Boot

```
@RestControllerAdvice
```

```
public class AccountControllerHandler {
```

1

```
    @ExceptionHandler(MyAccountNotFoundException.class)
```

```
    public ResponseEntity<ExceptionResponse> accountNotFound(  
        MyAccountNotFoundException exception) {
```

```
        ExceptionResponse response =
```

```
            new ExceptionResponse(exception.getMessage(),  
                                "More detail");
```

```
        return new ResponseEntity<ExceptionResponse>(response,  
                                                    HttpStatus.NOT_FOUND);
```

```
    }
```

```
}
```

Handling error in Spring Boot

```
@RestControllerAdvice
```

```
public class AccountControllerHandler {
```

```
@ExceptionHandler(MyAccountNotFoundException.class)
```

```
public ResponseEntity<ExceptionResponse> accountNotFound(  
    MyAccountNotFoundException exception) {
```

```
    ExceptionResponse response =
```

```
        new ExceptionResponse(exception.getMessage(),  
                               "More detail");
```

```
    return new ResponseEntity<ExceptionResponse>(response,  
                                                  HttpStatus.NOT_FOUND);
```

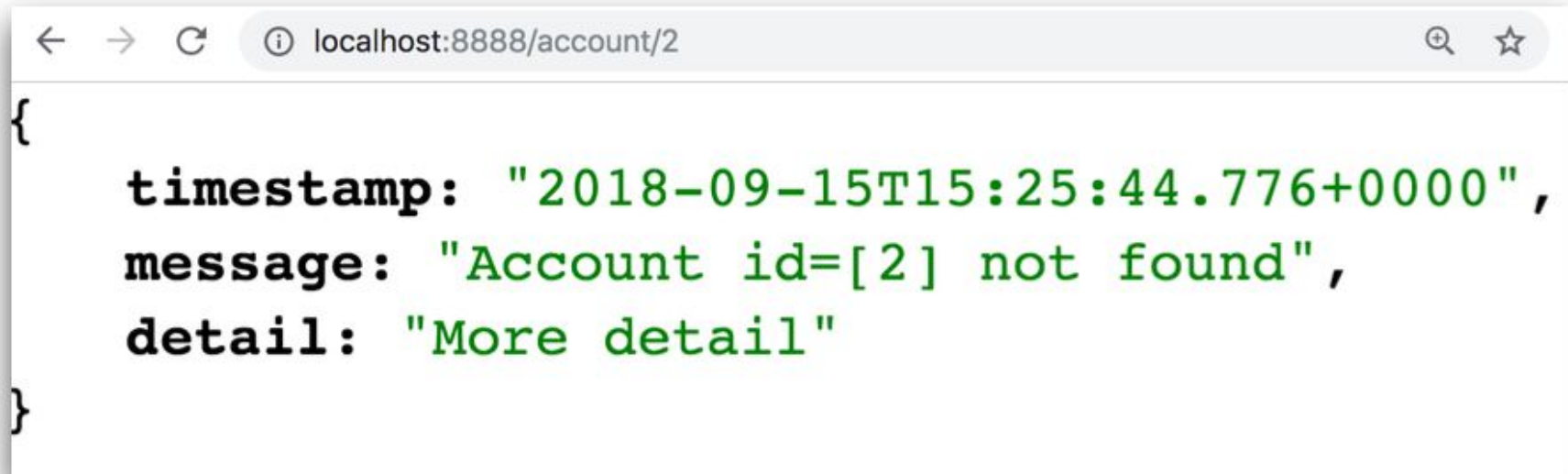
```
}
```

ExceptionResponse

Response format of error

```
public class ExceptionResponse{  
  
    private Date timestamp = new Date();  
    private String message;  
    private String detail;  
  
    public ExceptionResponse(String message, String detail) {  
        this.message = message;  
        this.detail = detail;  
    }  
}
```

Result of API

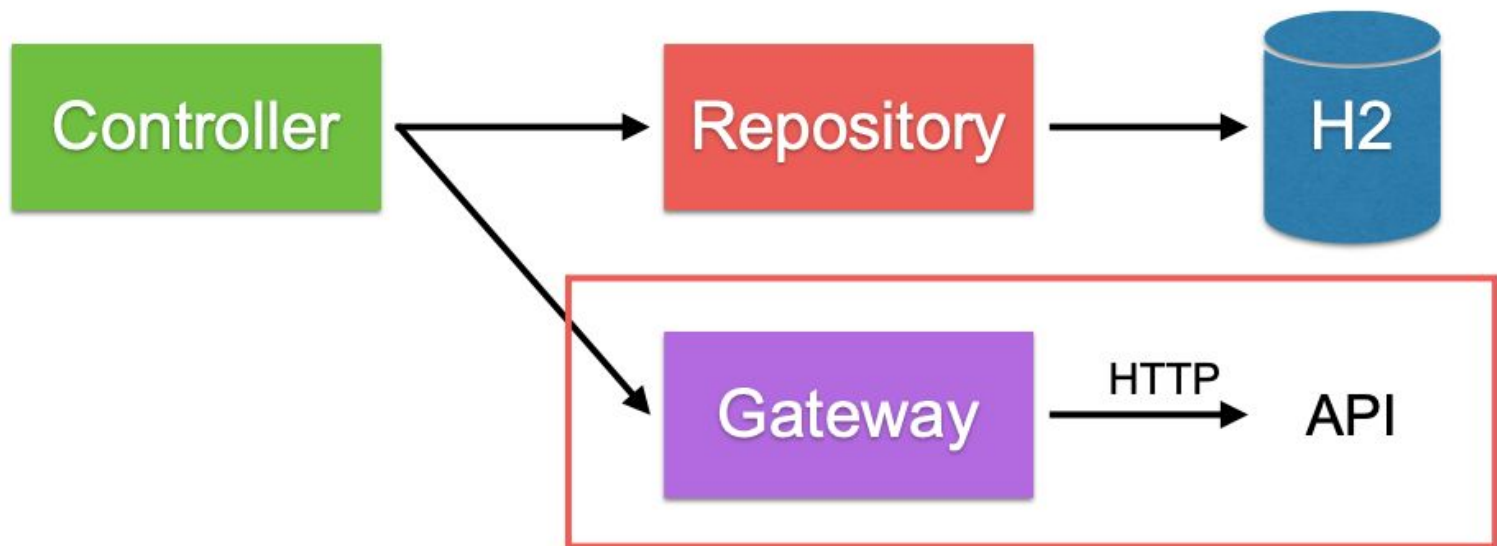
A screenshot of a web browser window. The address bar shows 'localhost:8888/account/2'. The main content area displays a JSON object with three fields: 'timestamp', 'message', and 'detail'. The text is styled with a monospaced font and green color for the values.

```
{  
  timestamp: "2018-09-15T15:25:44.776+0000",  
  message: "Account id=[2] not found",  
  detail: "More detail"  
}
```

Use case 2

Use case 2

Working with API



JSON Place Holder

<https://jsonplaceholder.cypress.io/posts/1>

JSONPlaceholder

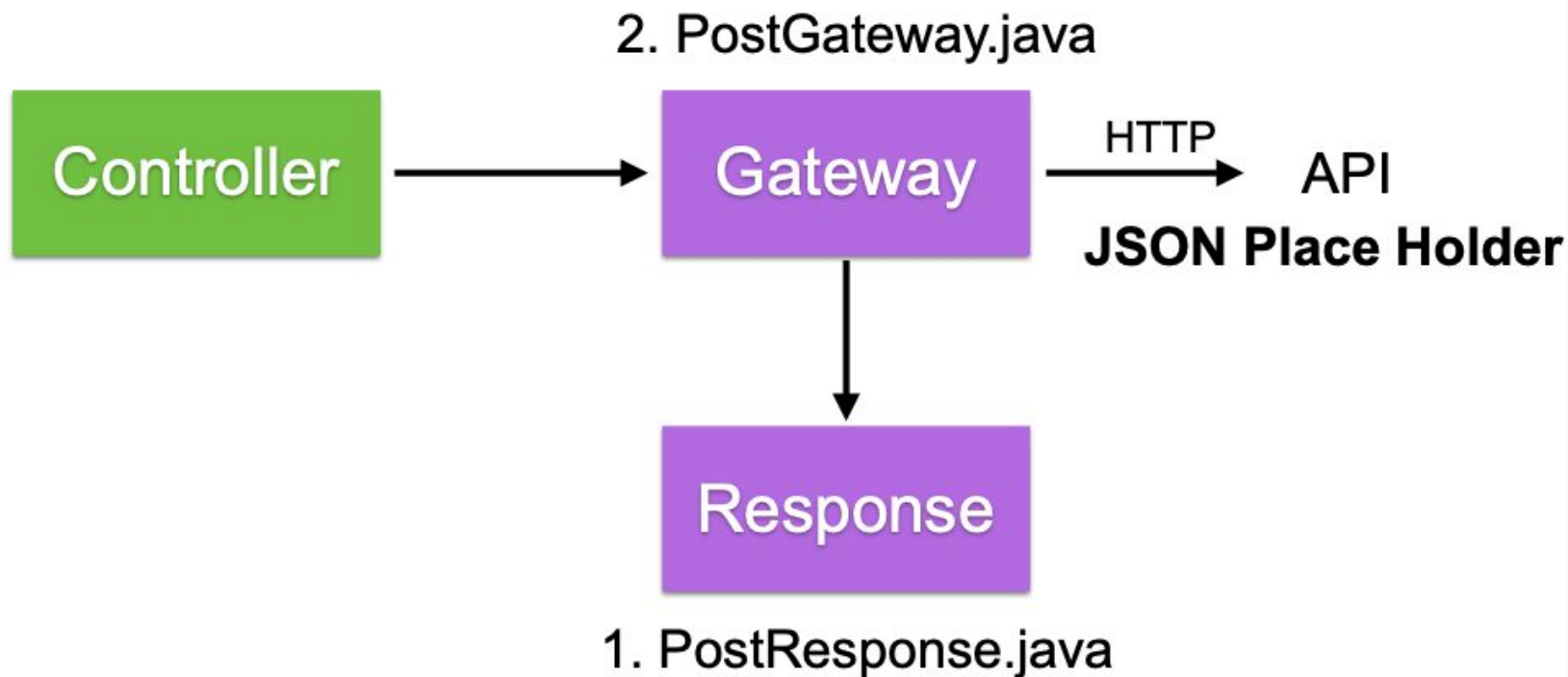
Fake Online REST API for Testing and Prototyping

Powered by [JSON Server](#) + [LowDB](#)

```
fetch('https://jsonplaceholder.cypress.io/todos/1')  
  .then(response => response.json())  
  .then(json => console.log(json))
```

Try it

Working with API



1. Create Response class

In package **post**

```
public class PostResponse {  
    private int id;  
    private int userId;  
    private String title;  
    private String body;
```

2. Create PostGateway class #1

In package post

@Component

```
public class PostGateway {
```

```
    private final RestTemplate restTemplate;
```

```
    private final String postApiUrl;
```

@Autowired

```
public PostGateway(final RestTemplate restTemplate,
```

```
                  @Value("${post.api.url}") final String postApiUrl) {
```

```
    this.restTemplate = restTemplate;
```

```
    this.postApiUrl = postApiUrl;
```

```
}
```

2. Create PostGateway class #1

In package post

@Component

```
public class PostGateway {
```

```
    private final RestTemplate restTemplate;
```

```
    private final String postApiUrl;
```

~~@Autowired~~

```
    public PostGateway(final RestTemplate restTemplate,
```

```
        @Value("${post.api.url}") final String postApiUrl) {
```

```
        this.restTemplate = restTemplate;
```

```
        this.postApiUrl = postApiUrl;
```

```
    }
```

Configuration ?

Configuration

Configuration in file application.properties

```
post.api.url=https://jsonplaceholder.cypress.io
```

2. Create PostGateway class #2

Get data from API

```
public Optional<PostResponse> getPostById(int id) {  
    String url = String.format("%s/posts/%d", postApiUrl, id);  
  
    try {  
        return Optional.ofNullable(  
            restTemplate.getForObject(url, PostResponse.class));  
    } catch (RestClientException e) {  
        return Optional.empty();  
    }  
}
```


Take a Group Shooting Photo

