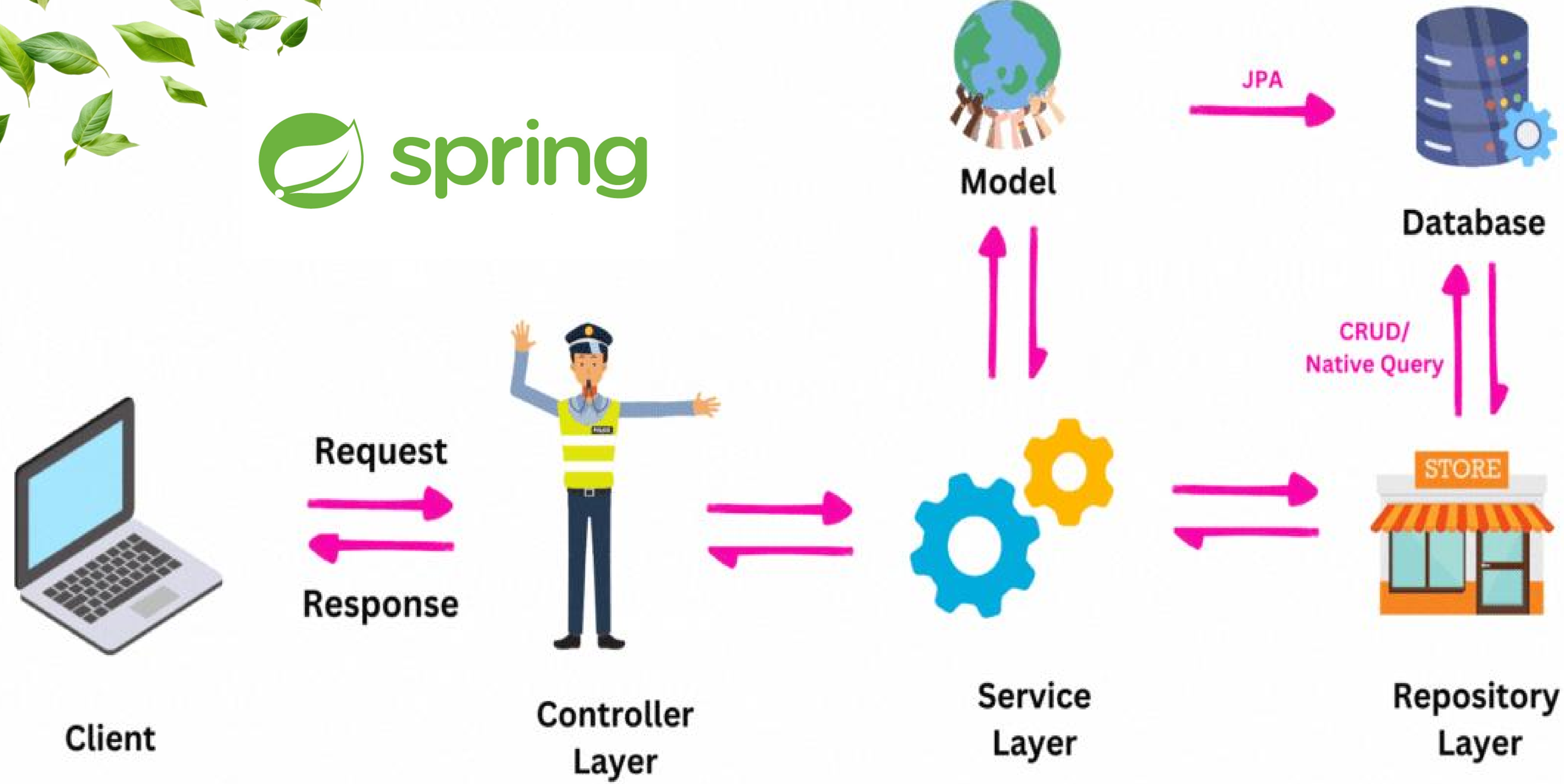
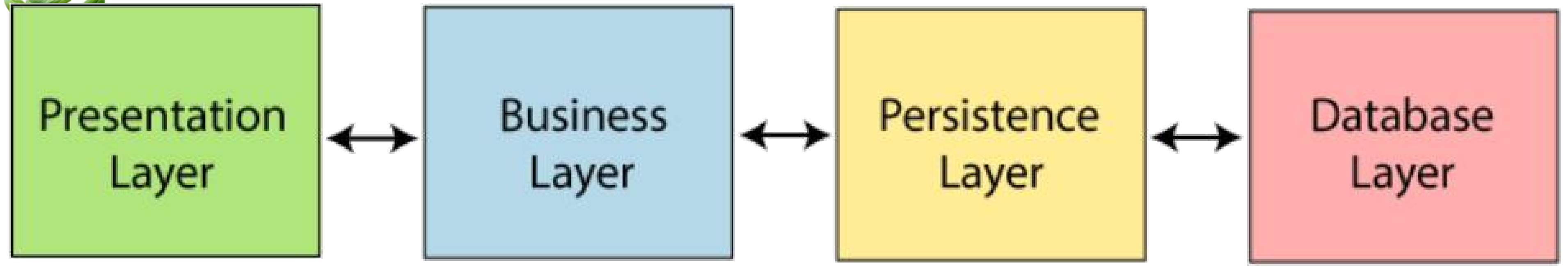
PAW - Programowanie Aplikacji Webowych 08
Spring: REST API

Inż. Juliusz Łosiński

```
Hello, World!
```


spring
Model
JPA
Database
CRUD/
Native Query
Request
Response
STORE
Client
Controller
Layer
Service
Layer
Repository
Layer

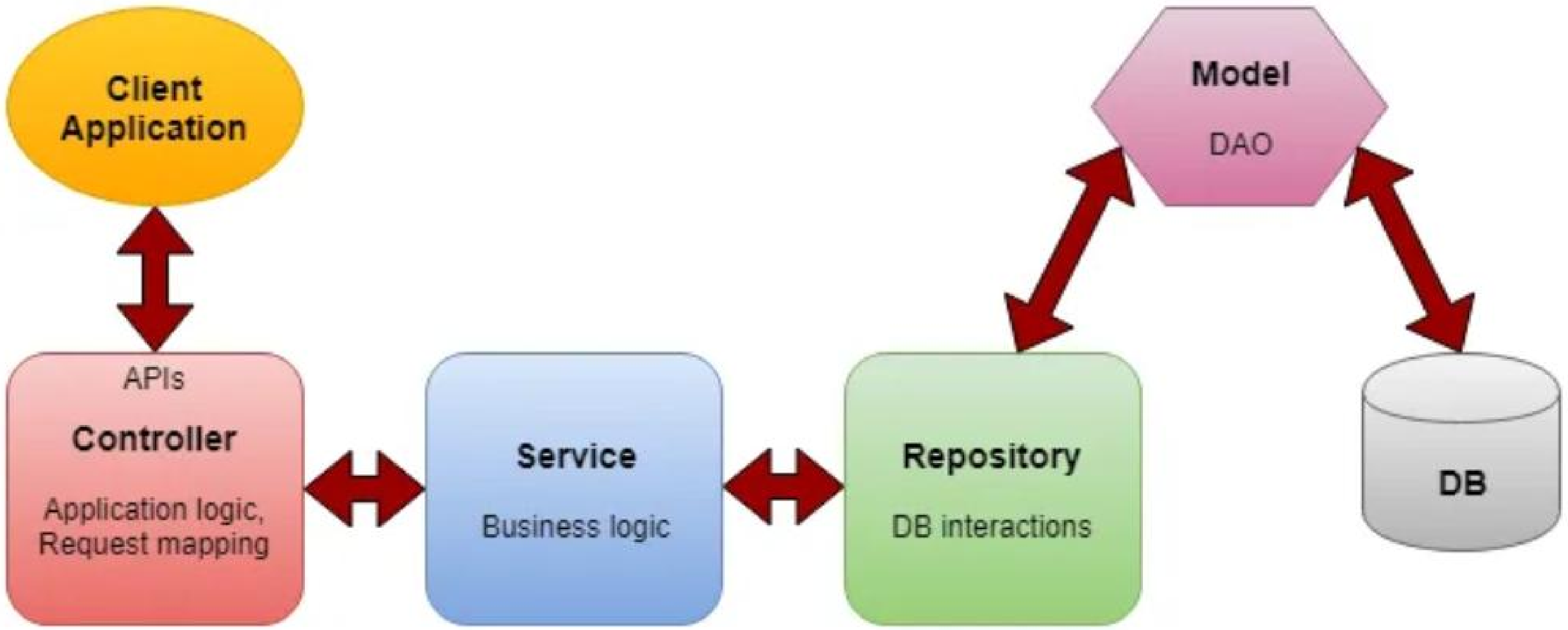


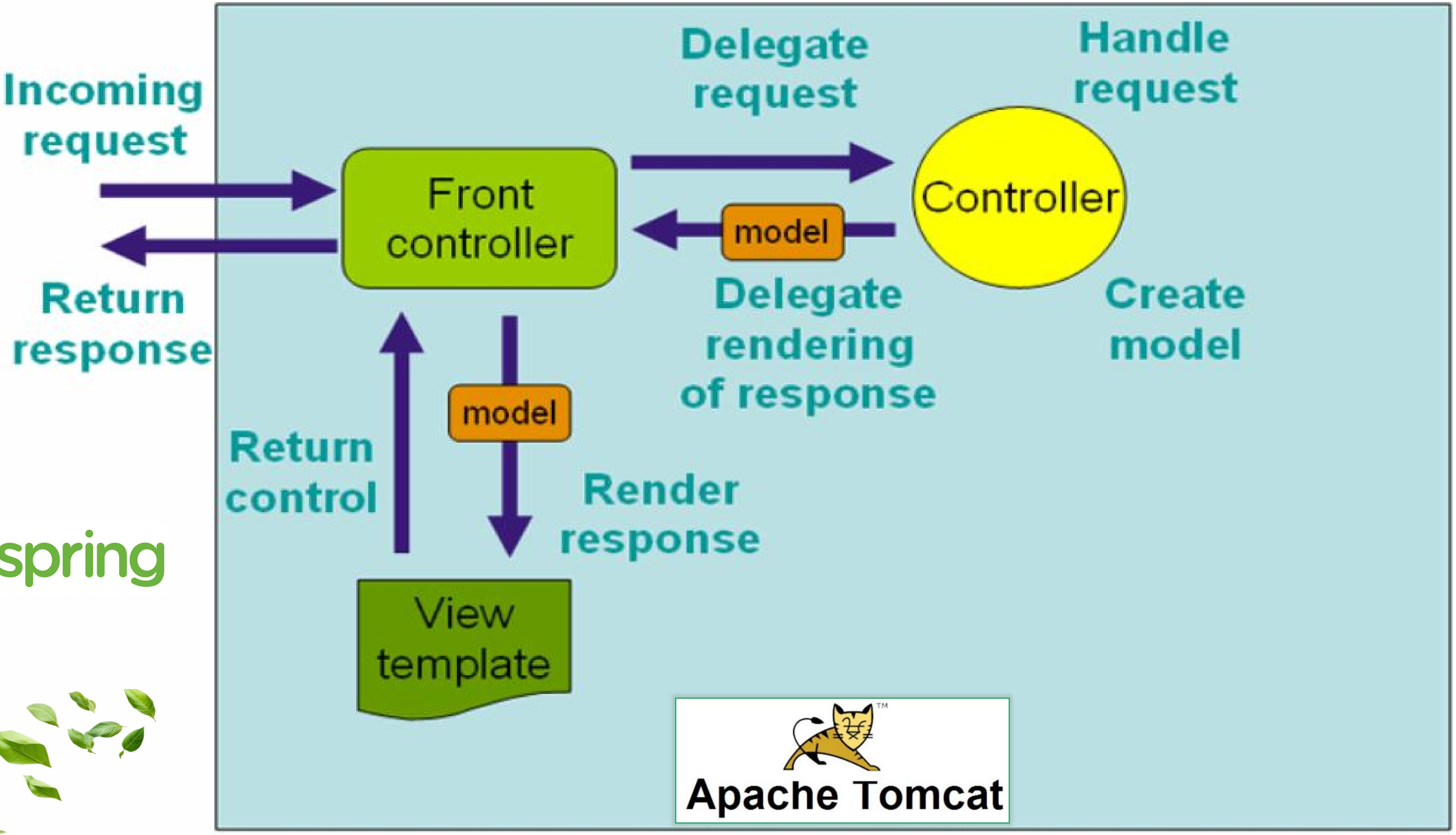
- Authentication
- JSON Translation

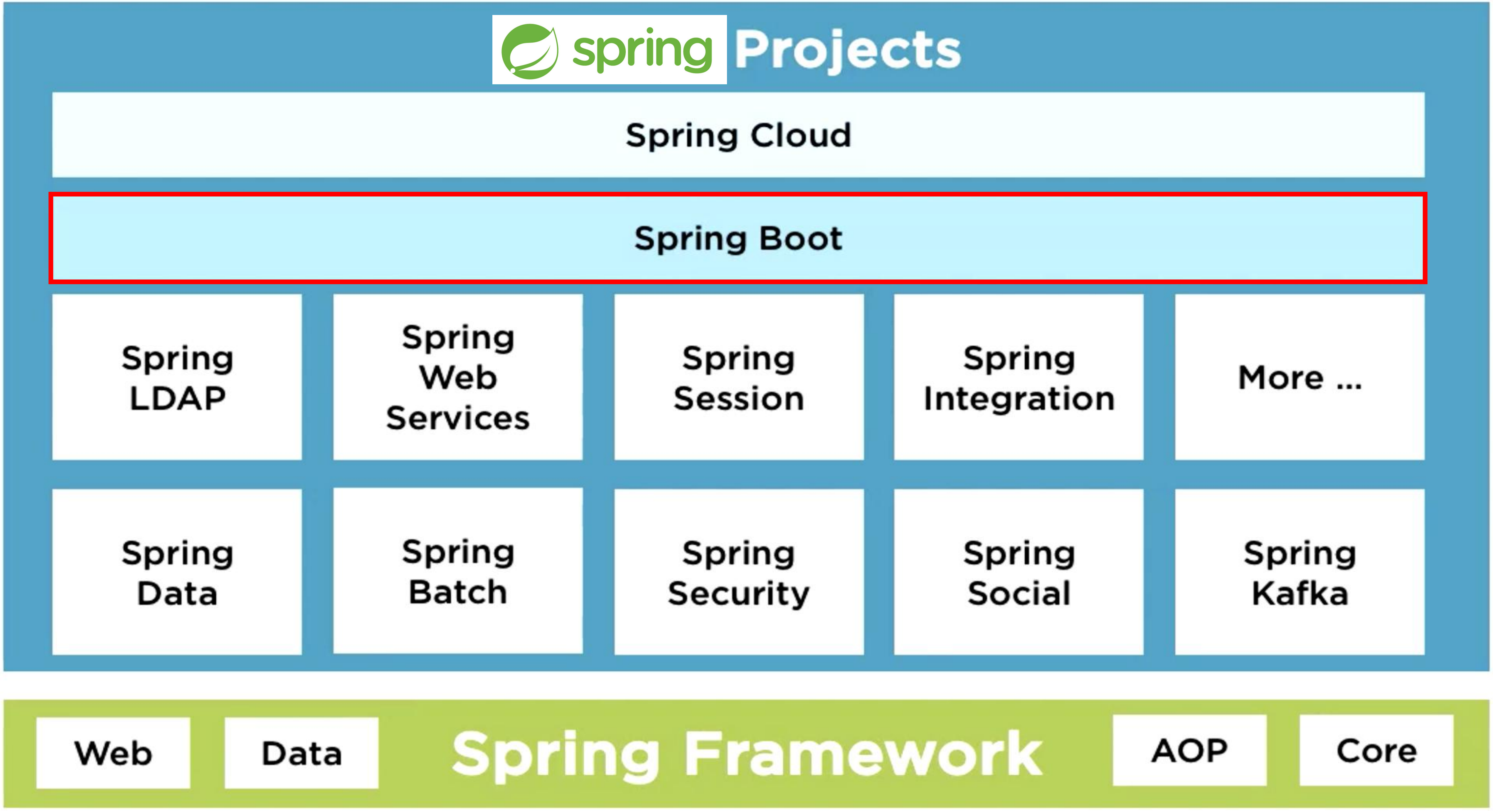
- Business Logic
- Validation
- Authorisation

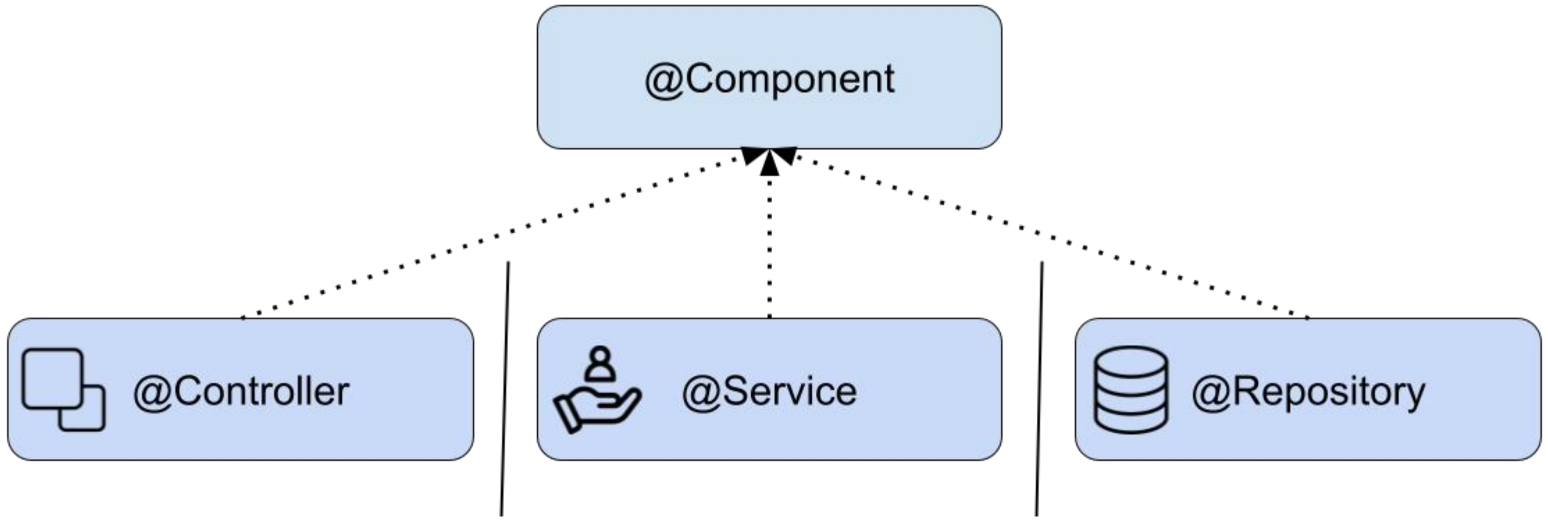
- Storage Logic

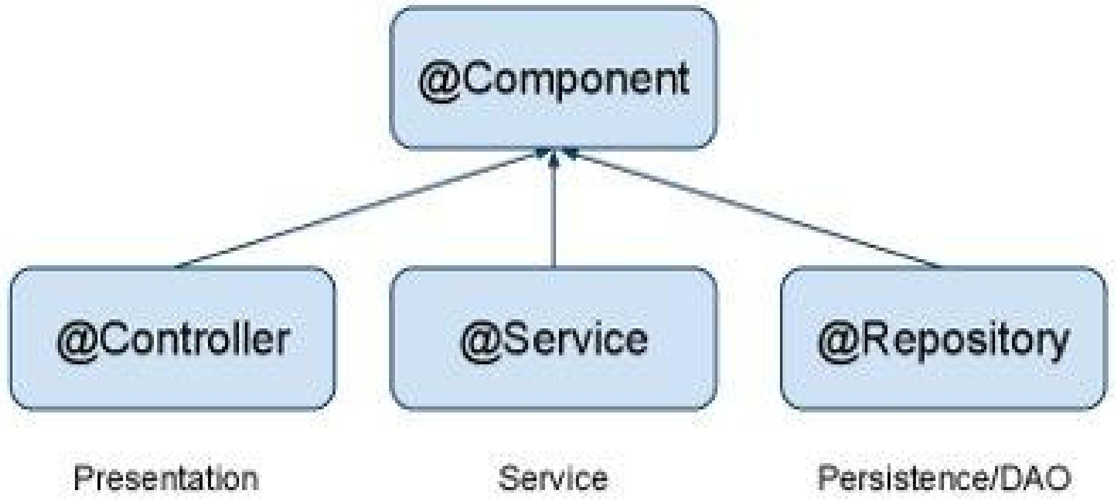
- Actual Database




Incoming request
Return response
Front controller
Delegate request
Handle request
Controller
model
Delegate rendering of response
Create model
model
Return control
Render response
View template
Apache Tomcat


spring Projects
Spring Cloud
Spring Boot
Spring LDAP
Spring Web Services
Spring Session
Spring Integration
More ...
Spring Data
Spring Batch
Spring Security
Spring Social
Spring Kafka
Web
Data
Spring Framework
AOP
Core


@Component
@Controller
@Service
@Repository


@Component
@Controller
@Service
@Repository
Presentation
Service
Persistence/DAO

Main Class	@SpringBootApplication	Spring Boot auto configuration
REST Endpoint	@RestController	Class with REST endpoints
	@RequestMapping	REST endpoint method
	@PathVariable	URI path parameter
	@RequestBody	HTTP request body
Periodic Tasks	@Scheduled	Method to run periodically
	@EnableScheduling	Enable Spring's task scheduling
Beans	@Configuration	A class containing Spring beans
	@Bean	Objects to be used by Spring IoC for dependency injection
Spring Managed Components	@Component	A candidate for dependency injection
	@Service	Like @Component
	@Repository	Like @Component, for data base access
Persistence	@Entity	A class which can be stored in the data base via ORM
	@Id	Primary key
	@GeneratedValue	Generation strategy of primary key
	@EnableJpaRepositories	Triggers the search for classes with @Repository annotation
	@EnableTransactionManagement	Enable Spring's DB transaction management through @Beans objects
Miscellaneous	@Autowired	Force dependency injection
	@ConfigurationProperties	Import settings from properties file
Testing	@SpringBootTest	Spring integration test
	@AutoConfigureMockMvc	Configure MockMvc object to test HTTP queries

Spring Boot and Web annotations

Use annotations to configure your web application.

T **@SpringBootApplication** - uses @Configuration, @EnableAutoConfiguration and @ComponentScan.

T **@EnableAutoConfiguration** - make Spring guess the configuration based on the classpath.

T **@Controller** - marks the class as web controller, capable of handling the requests. T **@RestController** - a convenience annotation of a @Controller and @ResponseBody.

M T **@ResponseBody** - makes Spring bind method's return value to the web response body.

M **@RequestMapping** - specify on the method in the controller, to map a HTTP request to the URL to this method.

P **@RequestParam** - bind HTTP parameters into method arguments.

P **@PathVariable** - binds placeholder from the URI to the method parameter.

Spring Cloud annotations

Make you application work well in the cloud.

T **@EnableConfigServer** - turns your application into a server other apps can get their configuration from.

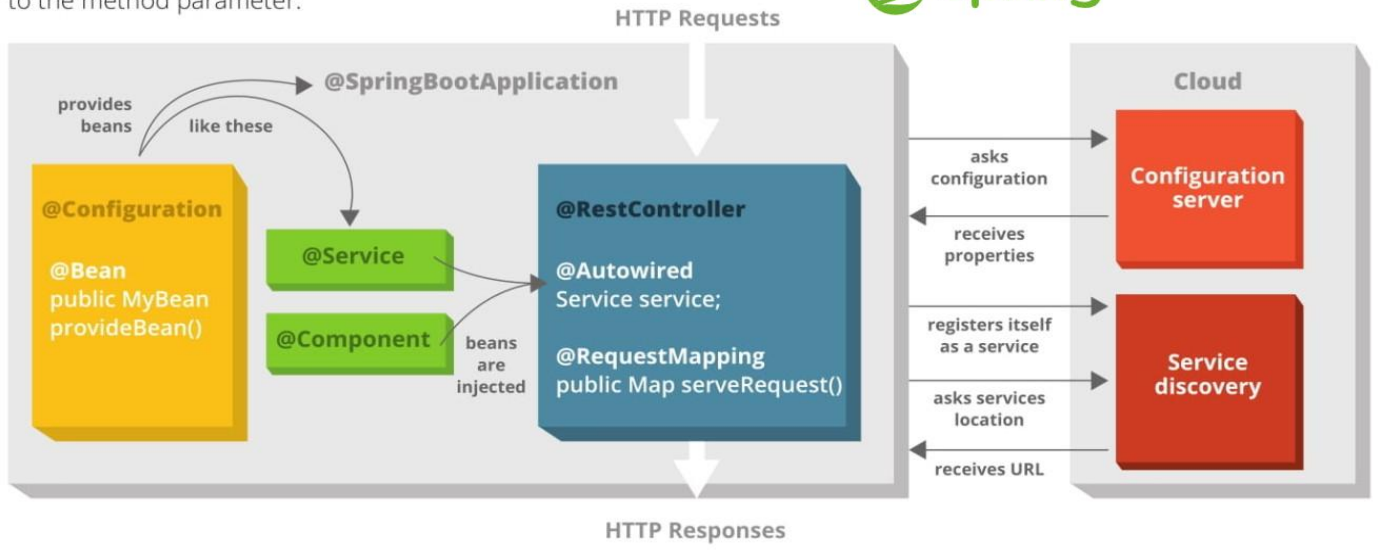
Use `spring.application.cloud.config.uri` in the client @SpringBootApplication to point to the config server.

T **@EnableEurekaServer** - makes your app an Eureka discovery service, other apps can locate services through it.

T **@EnableDiscoveryClient** - makes your app register in the service discovery server and discover other services through it.

T **@EnableCircuitBreaker** - configures Hystrix circuit breaker protocols.

M **@HystrixCommand(fallbackMethod = "fallbackMethodName")** - marks methods to fall back to another method if they cannot succeed normally.



Spring Framework annotat

Spring uses dependency injection to configure an your application together.

T **@ComponentScan** - make Spring scan the package for the @Configuration classes.

T **@Configuration** - mark a class as a source of bean definitions.

M **@Bean** - indicates that a method produces a bean to be managed by the Spring container.

T **@Component** - turns the class into a Spring bean at the auto-scan time. T **@Service** - specialization of the @Component, has no encapsulated state.

C F M **@Autowired** - Spring's dependency injection wires an appropriate bean into the marked class member.

T M **@Lazy** - makes @Bean or @Component be initialized on demand rather than eagerly.

C F M **@Qualifier** - filters what beans should be used to @Autowire a field or parameter.

C F M **@Value** - indicates a default value expression for the field or parameter, typically something like `"#{systemProperties.myProp}"`

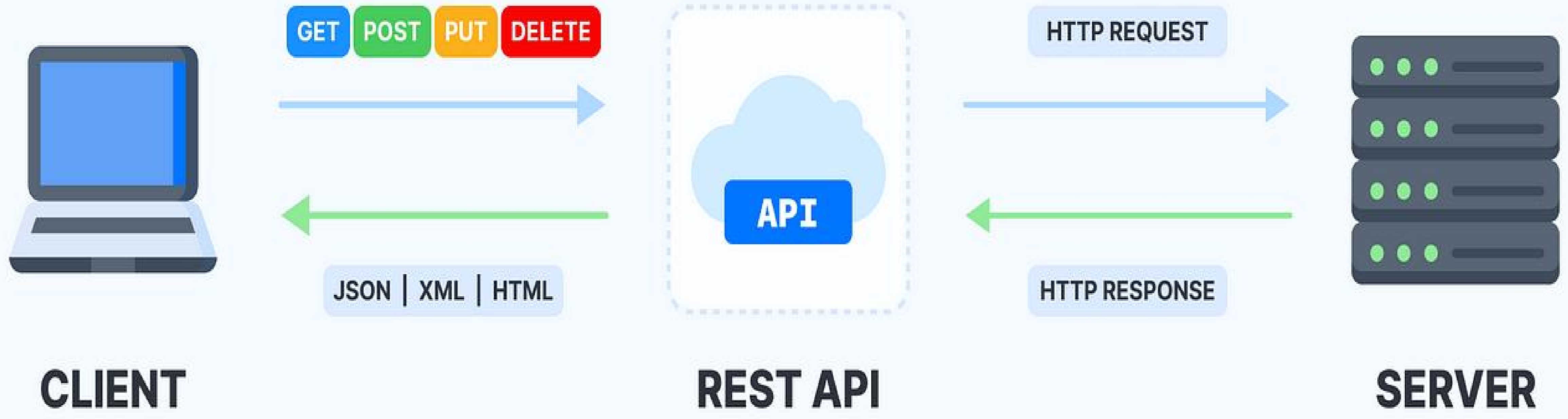
C F M **@Required** - fail the configuration, if the dependency cannot be injected.

Legend

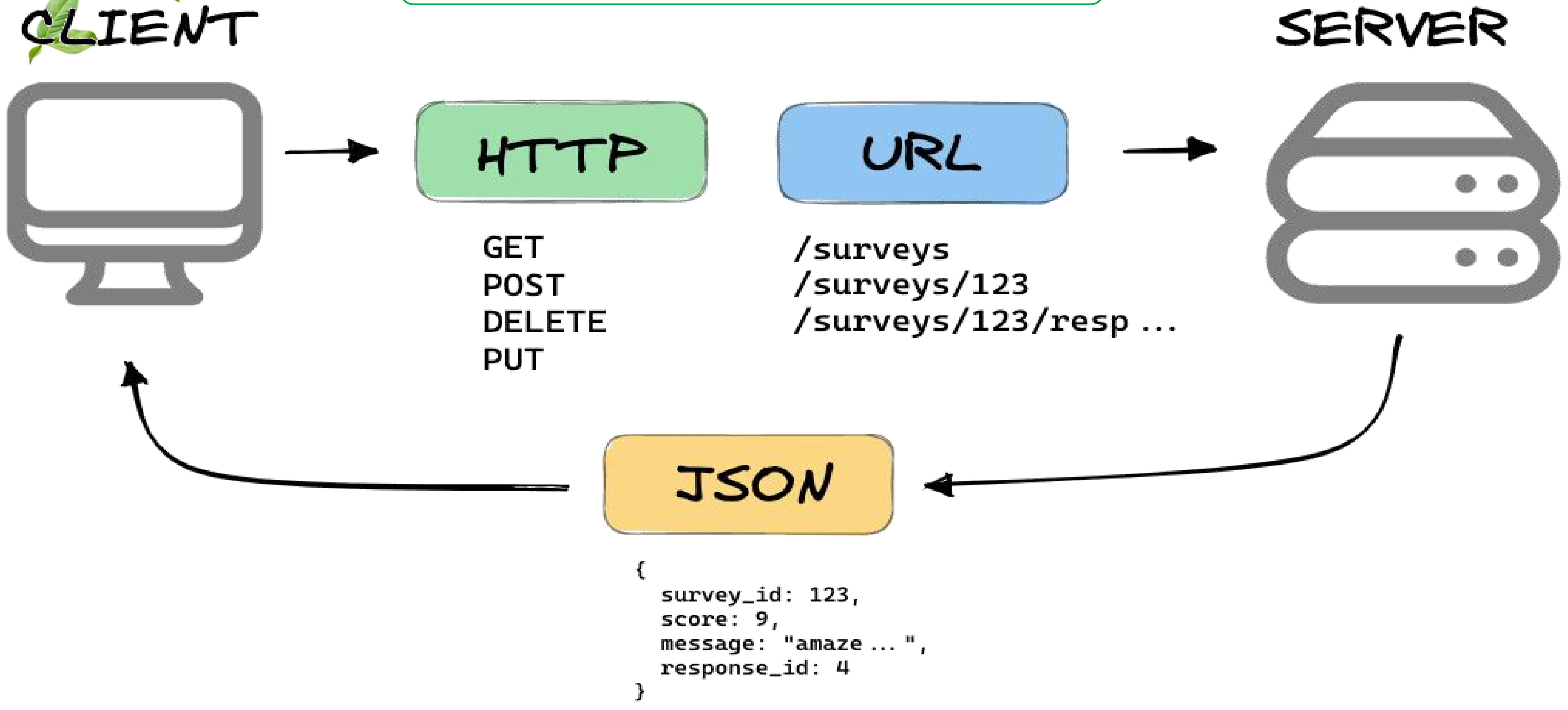
T - class
F - field annotation
C - constructor annotation
M - method
P - parameter

REST API Model

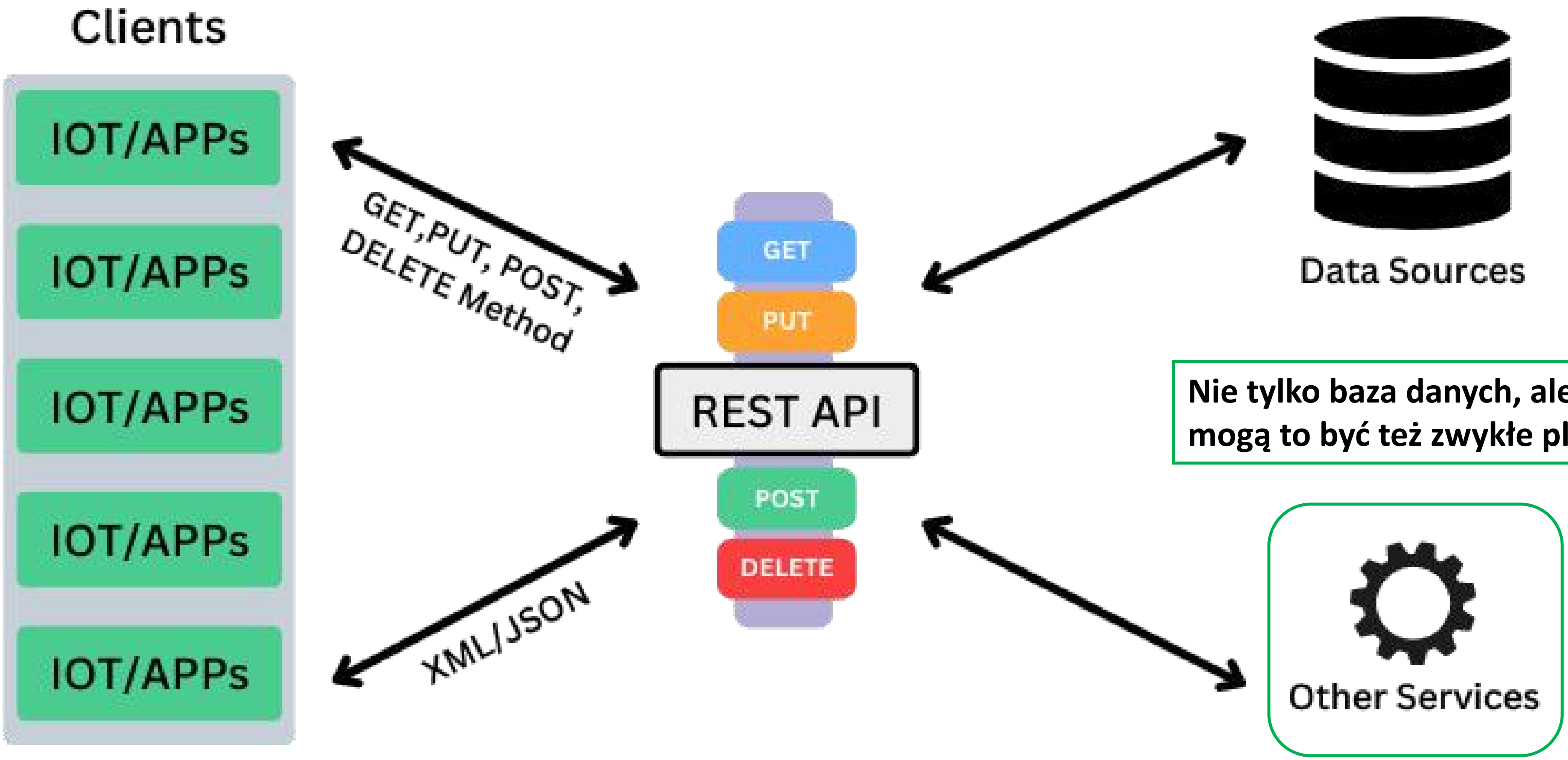
REST API Model



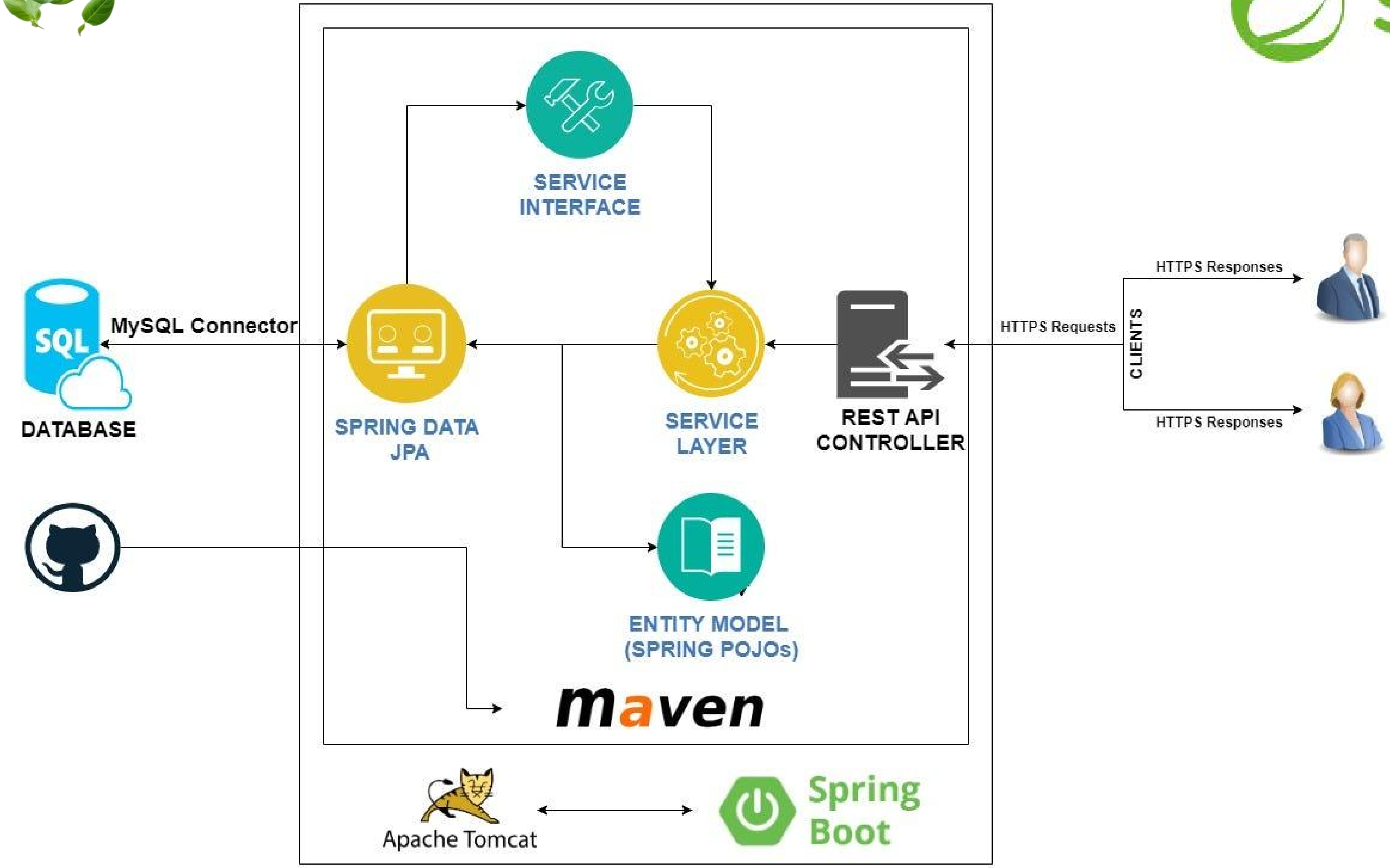
REST API Model

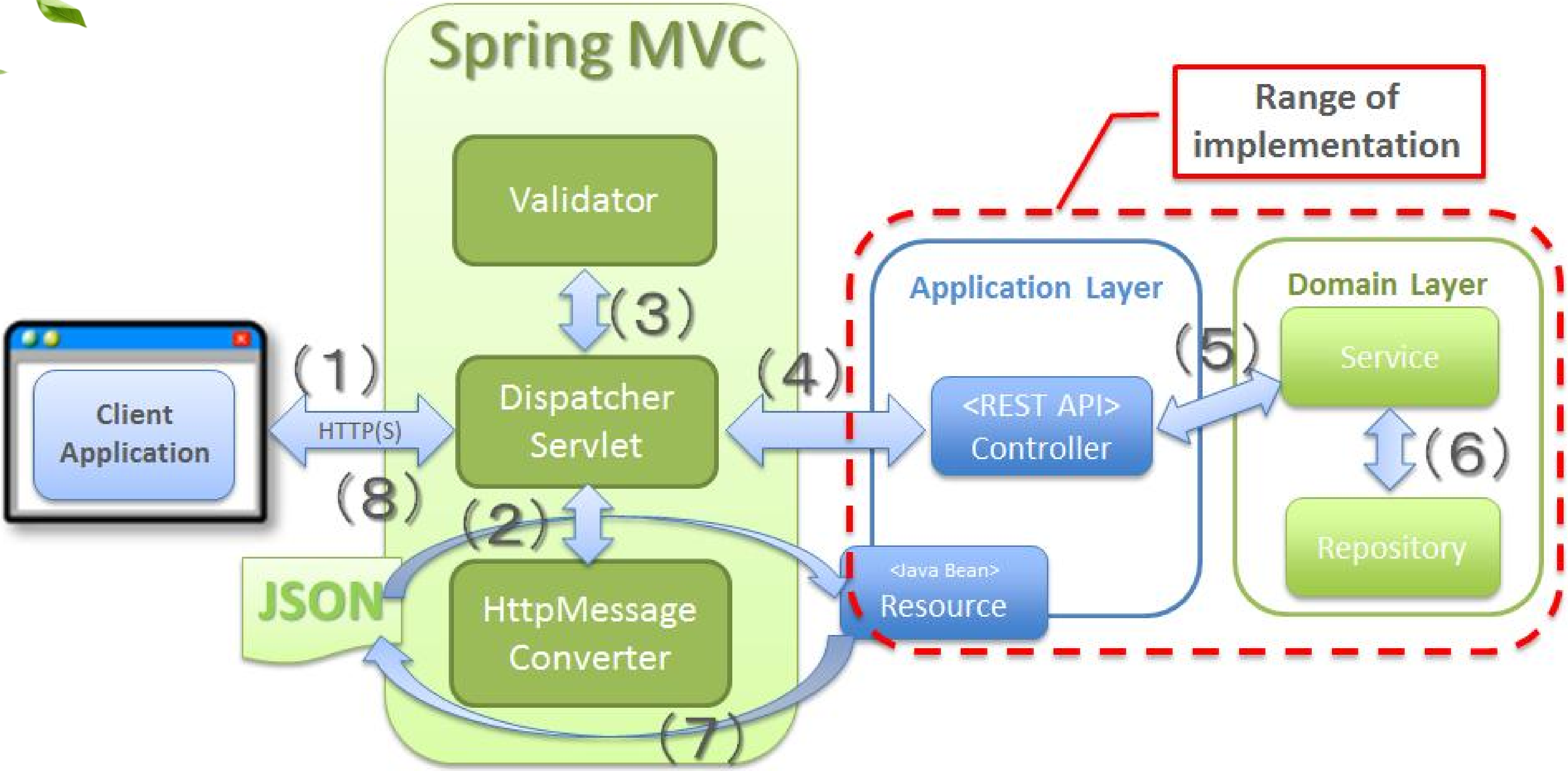


```
{
  survey_id: 123,
  score: 9,
  message: "amaze ... ",
  response_id: 4
}
```



Nie tylko baza danych, ale mogą to być też zwykłe pliki.


SERVICE INTERFACE
MySQL Connector
SQL
DATABASE
SPRING DATA JPA
SERVICE LAYER
REST API CONTROLLER
HTTPS Requests
HTTPS Responses
CLIENTS
HTTPS Responses
ENTITY MODEL (SPRING POJOs)
maven
Apache Tomcat
Spring Boot


Spring MVC
Validator
(3)
Client Application
(1)
HTTP(S)
Dispatcher Servlet
(4)
(8)
(2)
JSON
HttpMessage Converter
(7)
Range of implementation
Application Layer
<REST API> Controller
<Java Bean> Resource
(5)
Domain Layer
Service
(6)
Repository

Przykładowa Aplikacja

application.properties:

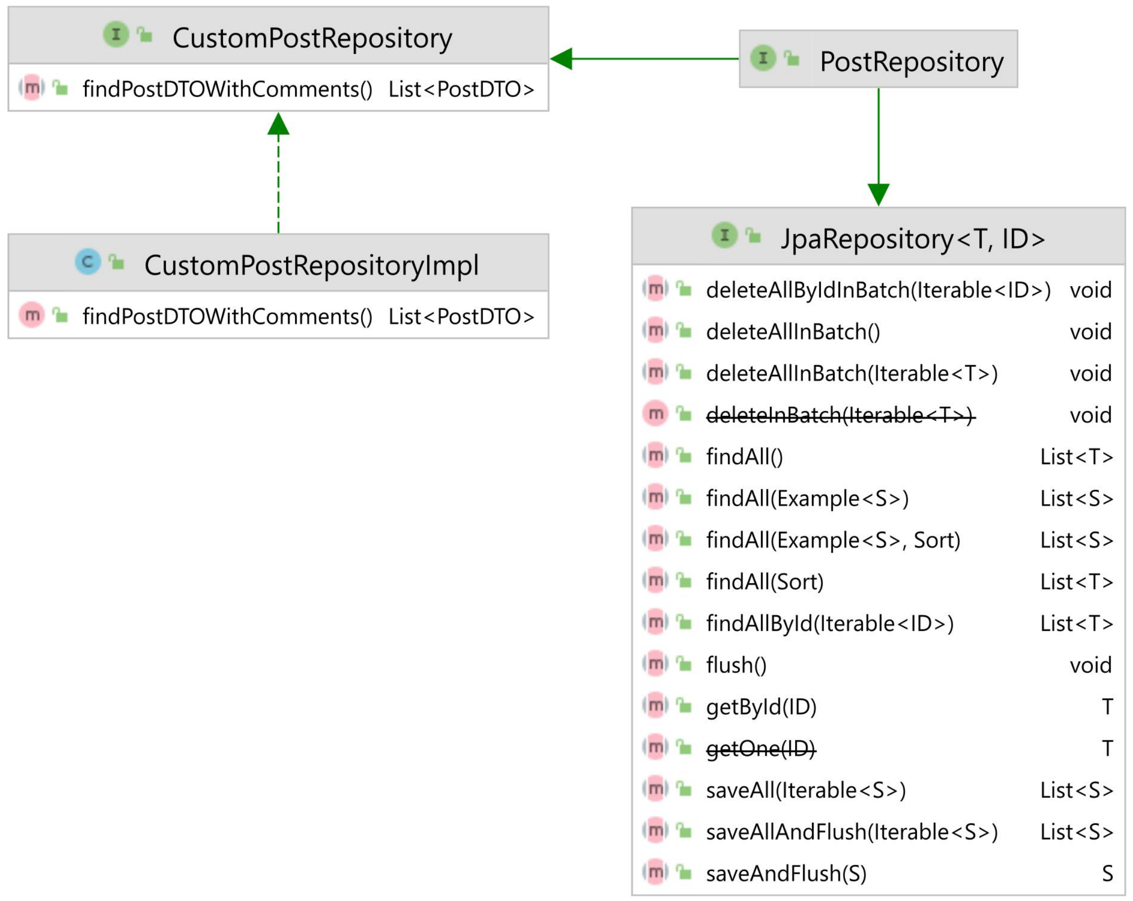
```
spring.datasource.url = jdbc:mysql://localhost:3306/user
spring.datasource.username = user
spring.datasource.password = user
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

User.java:

```
@Entity
@Table(name = "user")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    private String name;
}
```

Repozytorium

```
@Repository
public interface UserRepository extends CrudRepository<User, Long> {}
```


CustomPostRepository
findPostDTOWithComments() List<PostDTO>
PostRepository
CustomPostRepositoryImpl
findPostDTOWithComments() List<PostDTO>
JpaRepository<T, ID>
deleteAllByIdInBatch(Iterable<ID>) void
deleteAllInBatch() void
deleteAllInBatch(Iterable<T>) void
~~deleteInBatch(Iterable<T>)~~ void
findAll() List<T>
findAll(Example<S>) List<S>
findAll(Example<S>, Sort) List<S>
findAll(Sort) List<T>
findAllById(Iterable<ID>) List<T>
flush() void
getById(ID) T
~~getOne(ID)~~ T
saveAll(Iterable<S>) List<S>
saveAllAndFlush(Iterable<S>) List<S>
saveAndFlush(S) S

Kontroler:

```
@RestController
@RequestMapping("/api/user")
public class UserController {

    @Autowired
    private UserRepository userRepository;

    @GetMapping
    public List<User> findAllUsers() {
        return userRepository.findAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> findUserById(@PathVariable(value = "id") long id) {
        Optional<User> user = userRepository.findById(id);

        if(user.isPresent()) {
            return ResponseEntity.ok().body(user.get());
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    @PostMapping
    public User saveUser(@Validated @RequestBody User user) {
        return userRepository.save(user);
    }
}
```

Testowanie:

```
$ curl --location --request POST 'http://localhost:8080/api/user'
--header 'Content-Type: application/json'
--data-raw '{ "id": 4, "name": "Jason" }'
```

```
{
     "id": 4,
     "name": "Jason"
}
```

```
[
   {
         "id": 1,
         "name":"John"
   },
   {
         "id": 2,
         "name":"Jane"
   },
   {
         "id": 3,
         "name": "Juan"
   }
]
```

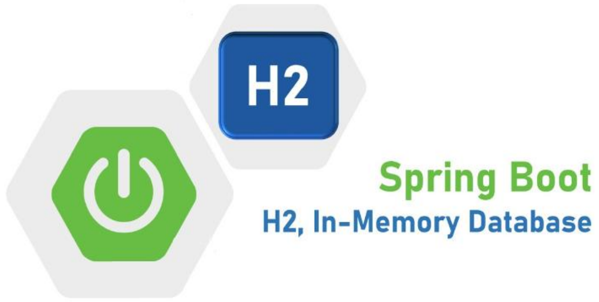
Laboratorium

Tworzenie REST API z wykorzystaniem frameworka Spring.

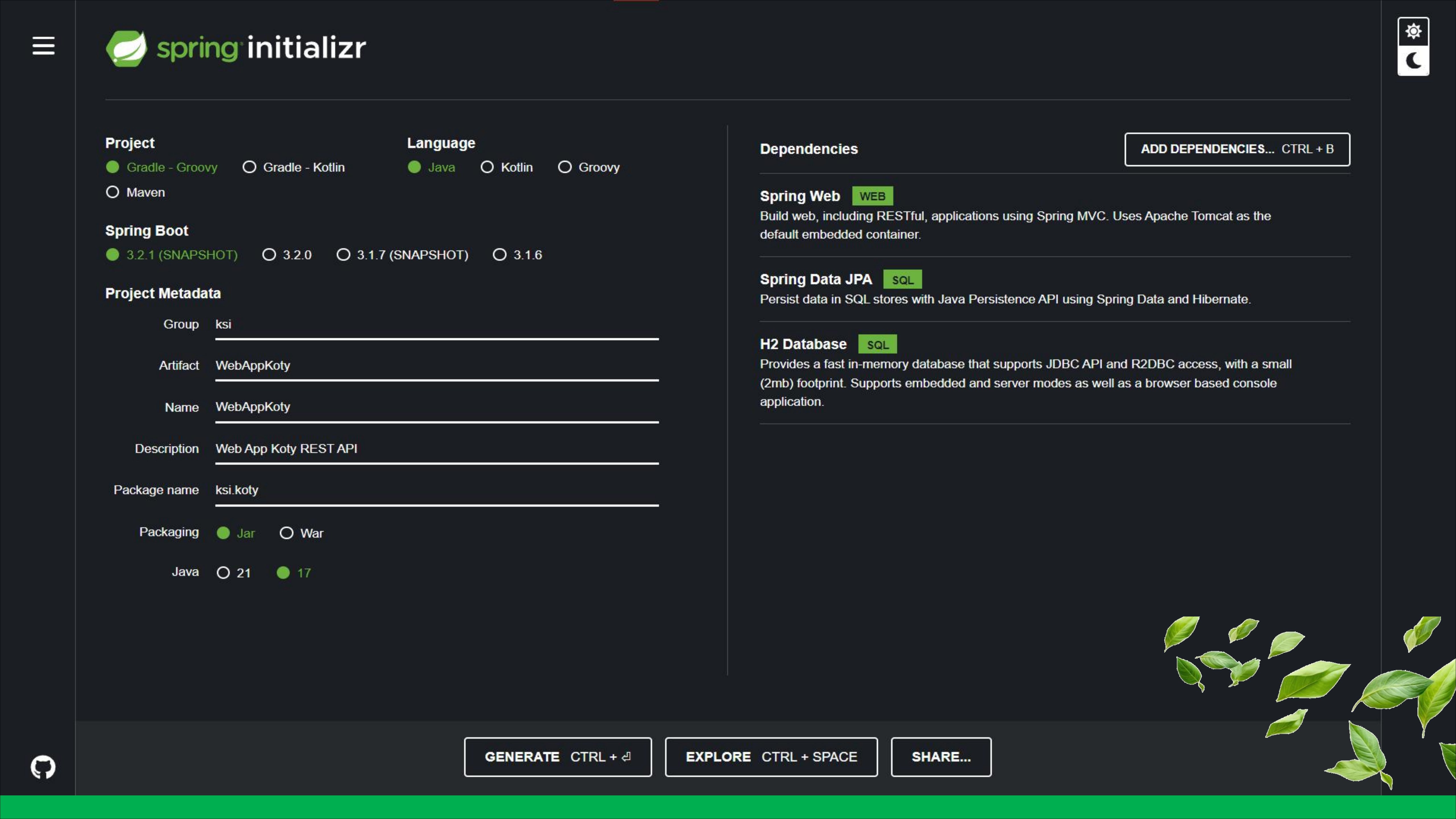
Celem zajęć jest stworzenie aplikacji backendowej umożliwiającej zarządzanie bazą kotów i pobieranie ciekawostek o nich, czerpanych z zewnętrznego serwisu.

W tym projekcie wykorzystamy:

- webową wersję Spring Initializr (zamiast pluginu w Eclipse),
- system baz danych H2,



System bazodanowy **H2** został napisany w Javie i może działać jako oprogramowanie klient serwer lub być osadzony w aplikacji Java. W tym projekcie skorzystamy z osadzonej bazy danych.


spring initializr

Project
[✓ Gradle - Groovy] Gradle - Kotlin Maven

Language
[✓ Java] Kotlin Groovy

Spring Boot
[✓ 3.2.1 (SNAPSHOT)] 3.2.0 3.1.7 (SNAPSHOT) 3.1.6

Project Metadata

Group	ksi
Artifact	WebAppKoty
Name	WebAppKoty
Description	Web App Koty REST API
Package name	ksi.koty
Packaging	[✓ Jar] War
Java	21 [✓ 17]

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

H2 Database SQL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

```java
@Entity
class Cat
{
    private @Id @GeneratedValue Long id;
    private String name;
    private String breed;

    Cat(){}

    Cat(String name, String breed)
    {
        this.name=name;
        this.breed=breed;
    }

    public Long getId()
    {
        return this.id;
    }

    public String getName()
    {
        return this.name;
    }

    public String getBreed()
    {
        return this.breed;
    }

    public void setId(Long id)
    {
        this.id = id;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public void setBreed(String breed)
    {
        this.breed = breed;
    }
```

```java
@Override
public boolean equals(Object object)
{
    if (this == object)
        return true;
    if (!(object instanceof Cat))
        return false;
    Cat cat = (Cat) object;
    return Objects.equals(this.id, cat.id) && Objects.equals(this.name, cat.name)
        && Objects.equals(this.breed, cat.breed);
}

@Override
public int hashCode()
{
   return Objects.hash(this.id, this.name, this.breed);
}

@Override
public String toString()
{
    return "Cat{" + "id=" + this.id + ", name='" + this.name + '\'' + ", breed='" + this.breed + '\'' + '}';
}
```

Klasa Cat.

4. Zasiewanie bazy danych

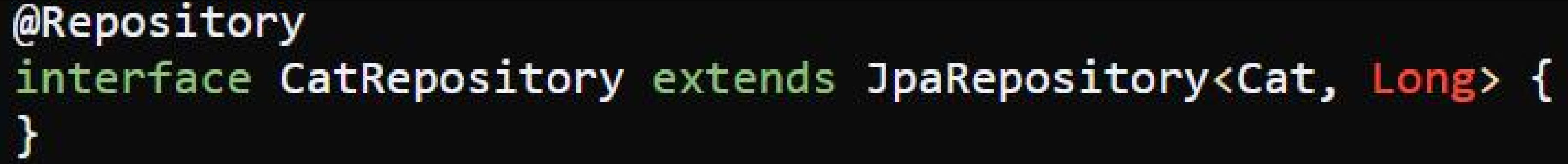
Dzięki zastosowaniu osadzonej bazy danych H2, można skupić się na programowaniu, bez konieczności manualnego korzystania z zewnętrznego systemu baz danych np. PostgreSQL, a także bez konfiguracji połączenia, co może być przydatne np. w fazie testowania lub budowania prototypu.

Utworzymy plik dodający pierwsze koty do bazy H2, w tym celu utworzymy klasę LoadDatabase z adnotacją `@Configuration`. Adnotacja ta informuje o tym, że klasa zawiera metody do tworzenia obiektów typu bean. Dzięki dodaniu adnotacji `@Bean` przed metodą `initDatabase`, która zwraca w wyniku obiekt klasy `CommandLineRunner`, nie musimy sami tworzyć tego obiektu, zajmie się tym Spring.

W metodzie wykorzystamy log z pakietu `org.slf4j` do prezentowania informacji, a na końcu zapiszemy nowe koty do bazy oraz wyświetlimy o nich informacje w logu.

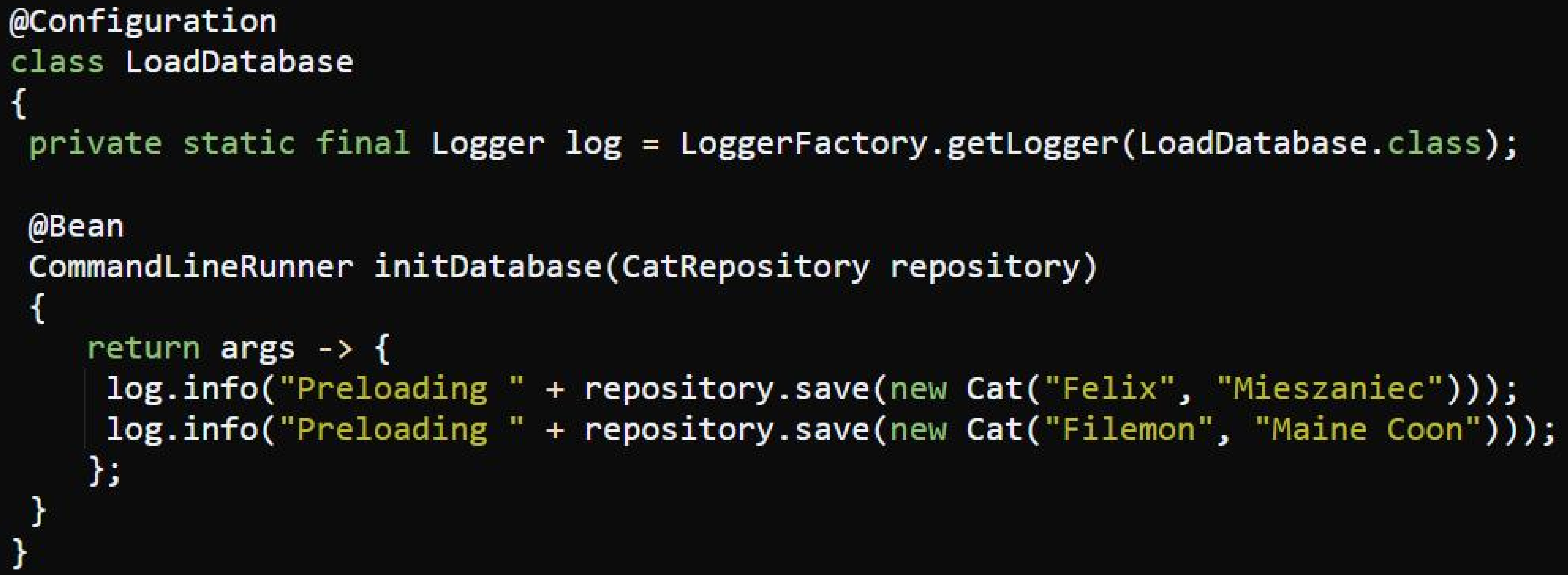
H2

Spring Boot
H2, In-Memory Database



```java
@Repository
interface CatRepository extends JpaRepository<Cat, Long> {
}
```

Zdefinowanie repozytorium oraz konfiguracji.



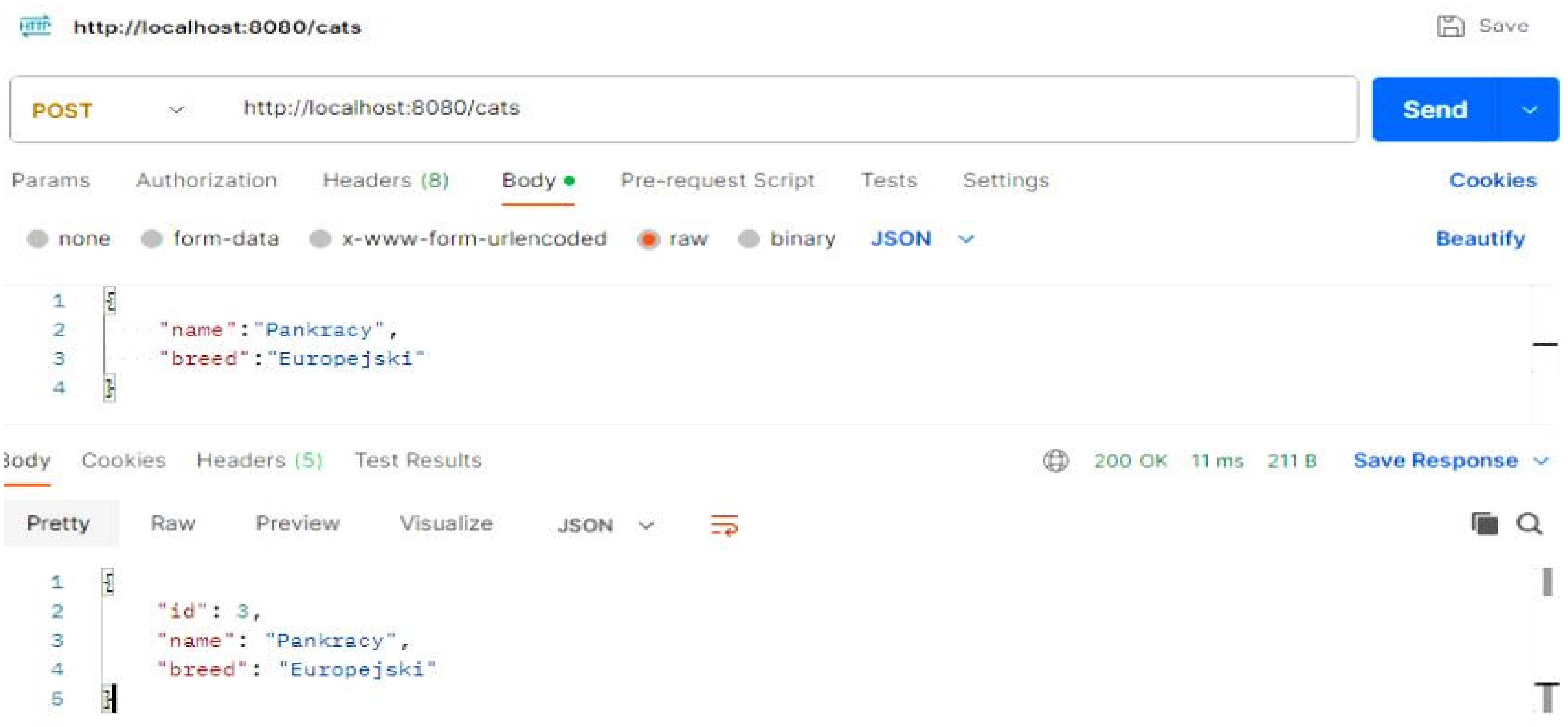
```java
@Configuration
class LoadDatabase
{
 private static final Logger log = LoggerFactory.getLogger(LoadDatabase.class);

 @Bean
 CommandLineRunner initDatabase(CatRepository repository)
 {
    return args -> {
     log.info("Preloading " + repository.save(new Cat("Felix", "Mieszaniec")));
     log.info("Preloading " + repository.save(new Cat("Filemon", "Maine Coon")));
    };
 }
}
```

```java
@RestController
class CatController
{
 private final CatRepository repository;

 CatController(CatRepository repository)
 {
    this.repository = repository;
 }

 @GetMapping("/cats")
 List<Cat> getCats()
 {
    return repository.findAll();
 }

 @GetMapping("/cats/{id}")
 Optional<Cat> getCat(@PathVariable("id") Long id)
 {
    return repository.findById(id);
 }

 @GetMapping("/fun-fact")
 String getFunFact()
 {
    final String uri = "https://catfact.ninja/fact";
    RestTemplate restTemplate = new RestTemplate();
    String result = restTemplate.getForObject(uri, String.class)
    return result;
 }

 @PostMapping("/cats")
 Cat newCat(@RequestBody Cat newCat)
 {
    return repository.save(newCat);
 }
```

```java
@PutMapping("/cats/{id}")
Cat replaceCat(@RequestBody Cat newCat, @PathVariable("id") Long id)
{
    return repository.findById(id)
        .map(cat -> {
        cat.setName(newCat.getName());
        cat.setBreed(newCat.getBreed());
        return repository.save(cat);})
       .orElseGet(() -> {
        newCat.setId(id);
        return repository.save(newCat);});
}

@DeleteMapping("/cats/{id}")
void deleteCat(@PathVariable("id") Long id)
{
   repository.deleteById(id);
}
```

Zdefiniowanie kontrolera.

Testowanie przy użyciu Postman’a



POST http://localhost:8080/cats

```
{
    "name":"Pankracy",
    "breed":"Europejski"
}
```

200 OK 11 ms 211 B

```
{
    "id": 3,
    "name": "Pankracy",
    "breed": "Europejski"
}
```

7. Modyfikacje

Oprogramuj 3 wybrane błędy, które mogą pojawić się w odpowiedziach HTTP aplikacji korzystając z klasy ResponseEntity. Informacje na ten temat można znaleźć m.in. pod adresem:

https://spring.io/guides/tutorials/rest/
https://www.baeldung.com/spring-response-entity

Dodaj do pliku .pdf zrzuty ekranu z Postmana ilustrujące wykonane modyfikacje.

8. Frontend

Utwórz prostą stronę HTML + JS, z przyciskiem i polem tekstowym pobierającą informacje o kocie.

Podpowiedź:
```
fetch('http://localhost:8080/cats/2')
  .then((response) => response.json())
  .then((data) => console.log(data));
```

System sprawozdań laboratoryjnych Sprawer

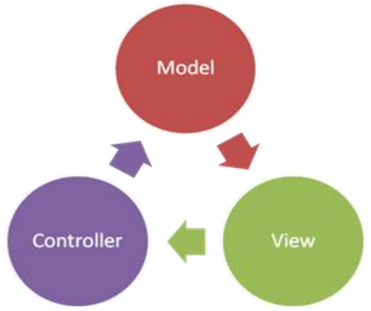
Katedra Systemów Informacyjnych UMG

Żeby uzyskać dostęp do systemu należy uzupełnić poniższe dane:

Login:
Hasło:

Sprawozdanie

W sprawozdaniu w systemie Sprawer wyślij plik .pdf ze zrzutami ekranu z Postmana (i przeglądarki) oraz spakowane pliki aplikacji.



Powodzenia!

Źródła

- https://spring.io/guides/tutorials/rest/,
- https://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server,
- https://spring.io/projects/spring-boot,
- https://www.baeldung.com/spring-core-annotations,
- https://www.baeldung.com/rest-with-spring-series