POLITECNICO
MILANO 1863

Software Engineering 2

The Spring Framework

L Lestingi, R Poiani,
M Camilli, E Di Nitto, M Rossi

POLITECNICO
MILANO 1863

The Spring Framework

https://spring.io/

The Spring Framework - Intro

POLITECNICO MILANO 1863

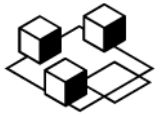
A Software Framework

An integrated collection of components, set of applications, conventions, principles and common practices for design and development of software

Spring

- One of the most popular **Java-based** framework for the development of distributed software systems
- It handles the infrastructure, so that you can focus on the application logic
- Well-supported by the community

The Spring Framework - Intro (cont'd)



Microservices

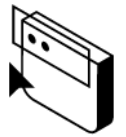
Quickly deliver production-grade features with independently evolvable microservices.

Reactive

Spring's asynchronous, nonblocking architecture means you can get more from your computing resources.

Cloud

Your code, any cloud—we've got you covered. Connect and scale your services, whatever your platform.



Web apps

Frameworks for fast, secure, and responsive web applications connected to any data store.

Serverless

The ultimate flexibility. Scale up on demand and scale to zero when there's no demand.

Event Driven

Integrate with your enterprise. React to business events. Act on your streaming data in realtime.

Batch

Automated tasks. Offline processing of data at a time to suit you.

The Spring Framework - Intro

POLITECNICO MILANO 1863

Spring offers **explicit** support for several key functionalities.

Some examples are:

- Building RESTful API
- Handling communication between microservices
- User Authentication
- Building an API Gateway
- ...

POLITECNICO
MILANO 1863

Building RESTful Web Services

https://spring.io/guides/gs/rest-service/

https://spring.io/guides/tutorials/rest/

A (very simple) example

POLITECNICO MILANO 1863

- You are tasked with developing a service that will handle HTTP GET requests on http://localhost:8080/greeting. The service will provide a JSON representation of a greeting in response:
  {"id":1,"content":"Hello, World!"}
- You have the option to personalize the greeting by including an optional name parameter in the query string:
  http://localhost:8080/greeting?name=User
  In this case, the JSON will contain the following content:
  {"id":1,"content":"Hello, User!"}
- "id" is an integer number that keeps track of the number of times that we greeted a user

A (very simple) example - OpenAPI

POLITECNICO MILANO 1863

```yaml
openapi: "3.0.2"
info:
  title: A (very simple) RESTful web service
  description: RESTful web service. You can find the tutorial at https://spring.io/guides/gs/rest-service/
  version: "1.0"
servers:
  - url: http://localhost:8080
components:
  schemas:
    Greeting:
      properties:
        id:
          type: integer
        content:
          type: string
          pattern: "Hello, ^{.*?}!$"
```

A (very simple) example - OpenAPI

```yaml
/greeting/:
  summary: Greeting and increment counter
  parameters:
    - name: name
      in: query
      description: Name of who should be greeted
      required: false
      schema:
        type: string
        default: World
  get:
    Try it | Audit
    operationId: greetNewUser
    responses:
      "200":
        description: Successful greeting
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/Greeting"
```

A (very simple) example - Code

POLITECNICO MILANO 1863

- First, let us create a very simple model that represents a new "Greeting"
- We use the Java "record" keyword

  Records are designed for scenarios in which a class is generated solely to function as a straightforward data transporter.

```
package com.example.restservice;


public record Greeting(long id, String content) { }
```

```java
package com.example.restservice;

import java.util.concurrent.atomic.AtomicLong;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class GreetingController {

        private static final String template = "Hello, %s!";
        private final AtomicLong counter = new AtomicLong();

        @GetMapping("/greeting")
        public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
                return new Greeting(counter.incrementAndGet(), String.format(template, name));
        }
}
```

A (very simple) example

POLITECNICO MILANO 1863

- **@RestController**: handles HTTP requests
- **@GetMapping(/greeting)**: handles HTTP GET requests for /greeting
- The method greeting() returns a new instance of the Greeting class
- **@RequestParam(value = "name", defaultValue="World")**
  String name: specifies the input to the greeting method
  - Name is a String
  - The value of this parameter is taken from GET request parameter "name"
  - We specify a default value "World"
- The RESTful service populates a Greeting object, that will be directly written to the HTTP response as JSON

A more intricated example

POLITECNICO MILANO 1863

- We are going to build a RESTful service to manage the employees of a company
- A detailed discussion of this example can be found: https://spring.io/guides/tutorials/rest/

A more intricated example

Assume that we want to manage the Employees of a company. We are tasked to develop a microservice that handles CRUD operations on the employees that are part of the company.

We model employees with the following data:

- **ID:** identifier of the employee. It should be unique for each employee
- **Name:** the name of the employee
- **Role:** a string describing the role of the employee within the company

A more intricated example

POLITECNICO MILANO 1863

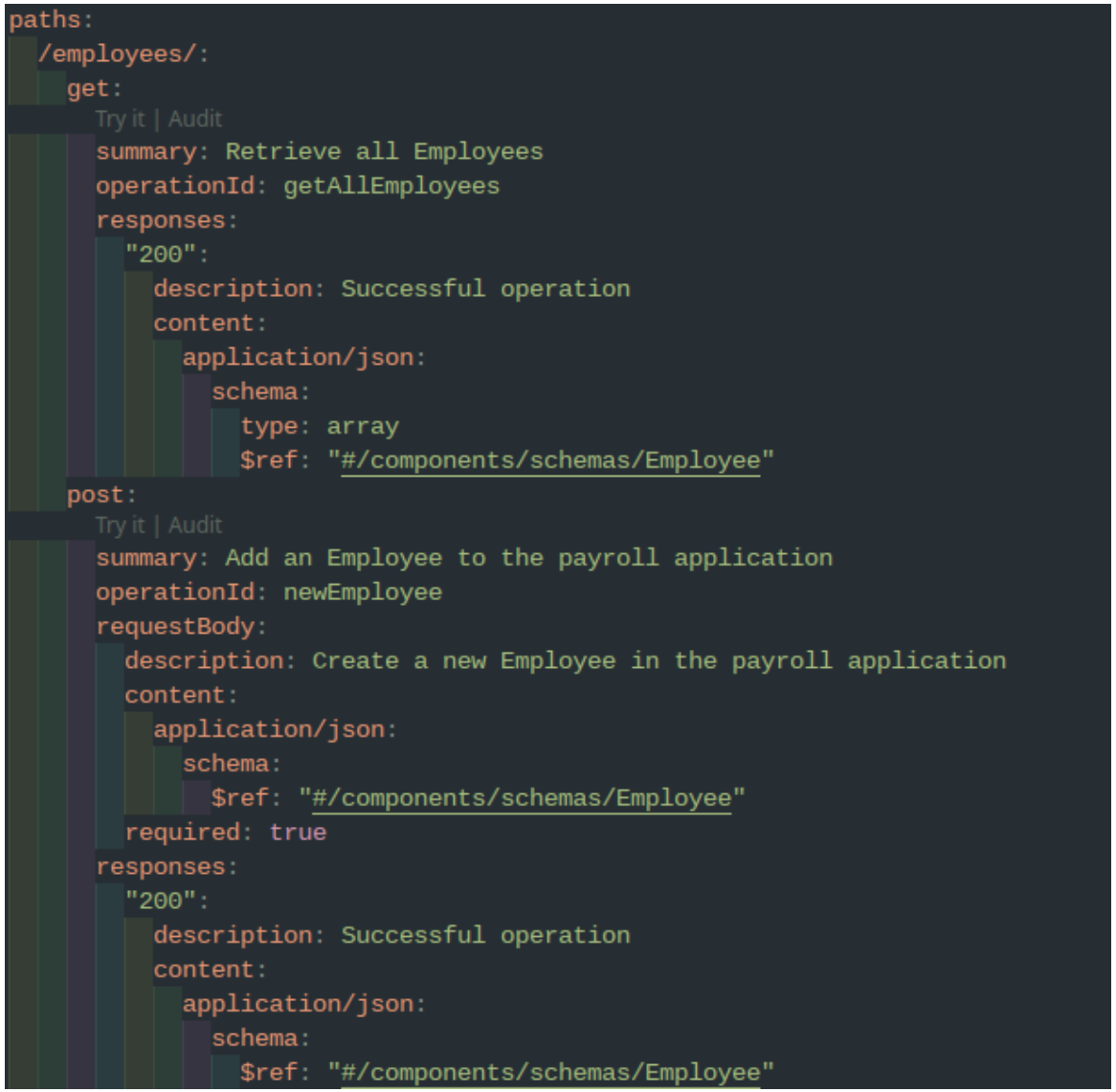
Our APIs should expose the following functionalities:

- **getAllEmployees**: retrieve the list of all the employees of the company
- **newEmployee:** insert a new Employee. The new Employee is returned to the user of the API.
- **findEmployeeByID:** retrieve data of a certain Employee given the ID. In the case in which no Employee is found, a 404 not found error is returned
- **replaceEmployee:** given an Employee ID and a new Employee, replace the (eventual) existing employee with the new one. The new employee is returned to the user of the API.
- **deteleteEmployee:** delete an Employee given the ID. In the case in which no Employee is found, no error is returned to the user of the API.

A more intricated example - OpenAPI

POLITECNICO MILANO 1863

```yaml
openapi: "3.0.2"
info:
  title: A (short) tutorial on RESTful web services
  description: RESTful web service. You can find the tutorial at https://spring.io/guides/gs/rest-service/
  version: "1.0"
servers:
  - url: http://localhost:8080
components:
  schemas:
    Employee:
      properties:
        id:
          type: integer
        name:
          type: string
        role:
          type: string
```


paths:
/employees/:
get:
Try it | Audit
summary: Retrieve all Employees
operationId: getAllEmployees
responses:
"200":
description: Successful operation
content:
application/json:
schema:
type: array
$ref: "#/components/schemas/Employee"
post:
Try it | Audit
summary: Add an Employee to the payroll application
operationId: newEmployee
requestBody:
description: Create a new Employee in the payroll application
content:
application/json:
schema:
$ref: "#/components/schemas/Employee"
required: true
responses:
"200":
description: Successful operation
content:
application/json:
schema:
$ref: "#/components/schemas/Employee"

```yaml
/employees/{id}/:
  post:
    Try it | Audit
    summary: Find Employee by ID
    operationId: getEmployeeById
    parameters:
      - name: id
        in: path
        description: ID of Employee to return
        required: true
        schema:
          type: integer
    responses:
      "200":
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/Employee"
      "404":
        description: Employee not found
```

```
put:
  Try it | Audit
  summary: Replace Employee with a new one
  description: if no employee with the specified ID is found,
    the new one is inserted in the system taking the specified ID
  operationId: replaceEmployee
  parameters:
    - name: id
      in: path
      description: id of the Employee that is replaced
      required: true
      schema:
        type: integer
  requestBody:
    description: New Employee that is inserted in the payroll application
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/Employee"
  responses:
    "200":
      description: Successful operation
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/Employee"
```

POLITECNICO MILANO 1863

```
delete:
  Try it | Audit
  summary: Delete an Employee by id
  operationId: deleteEmployee
  parameters:
    - name: id
      in: path
      description: Employee ID to be removed
      required: true
      schema:
        type: integer
  responses:
    "200":
      description: Successful operation
```

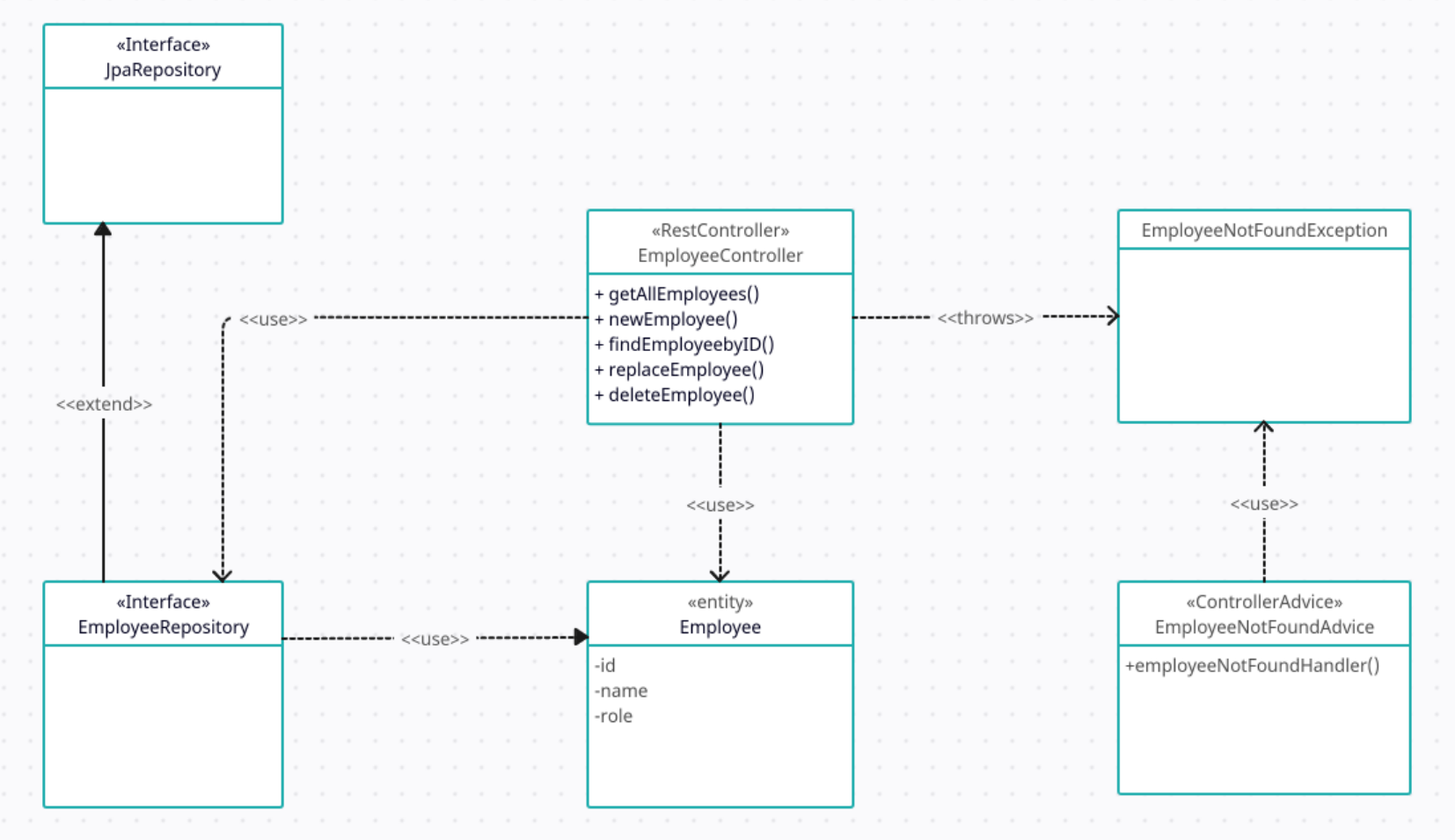
A more intricated example - API summary

POLITECNICO MILANO 1863

The exposed APIs, in short:

- **getAllEmployees**: /employees/, GET method
- **newEmployee:** /employees/, POST method
- **findEmployeeByID:** /employees/{id}, GET method
- **replaceEmployee:** /employees/{id}, PUT method
- **deteleteEmployee:** /employees/{id}, DELETE method

A more intricated example - UML Diagram



```java
@Entity // JPA Annotation to make this object ready for storage in a JPA-based data store.
class Employee {

    // Attributes of our Employees
    private @Id @GeneratedValue Long id; // @Id is a JPA annotation that specifies the primary key (i.e., id).
    private String name;
    private String role;

    Employee() {}

    // Custom constructor when we need a new instance but we do not have an ID
    Employee(String name, String role) {
        this.name = name;
        this.role = role;
    }

    // Getter and setter
    public Long getId() {
        return this.id;
    }

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

    public String getRole() {
        return this.role;
    }

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

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

Model

POLITECNICO MILANO 1863

We start by modelling employees:

JPA (Java Persistent Api)

- @Entity
- @ID
- @GeneratedValues

POLITECNICO MILANO 1863

Model (cont'd)

```
    public void setRole(String role) {
        this.role = role;
    }

    // Comparison, hashcode, and converting an Employee to a string
    @Override
    public boolean equals(Object o) {
        if (this == o)
            return true;
        if (!(o instanceof Employee))
            return false;
        Employee employee = (Employee) o;
        return Objects.equals(this.id, employee.id) && Objects.equals(this.name, employee.name)
            && Objects.equals(this.role, employee.role);
    }

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

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

A more intricated example - Repository

POLITECNICO MILANO 1863

```
import org.springframework.data.jpa.repository.JpaRepository;


interface EmployeeRepository extends JpaRepository<Employee, Long> {

}
```

- Spring Data **JPA repositories** are interfaces with methods supporting creating, reading, updating, and deleting records against a backend data store.
- We just need to declare the repository with **domain type** <Object, IDType>

A more intricated example - Application

POLITECNICO MILANO 1863

```java
@SpringBootApplication
public class PayrollApplication {

    public static void main(String... args) {
        SpringApplication.run(PayrollApplication.class, args);
    }
}
```

- We are already to launch the application
- @SpringBootApplication, a meta annotation to signal the entry point of the application

A more intricated example - Loading DB

POLITECNICO MILANO 1863

```java
@Configuration
class LoadDatabase {

    private static final Logger log = LoggerFactory.getLogger(LoadDatabase.class);

    @Bean
    CommandLineRunner initDatabase(EmployeeRepository repository) {

        return args -> {
            log.info("Preloading " + repository.save(new Employee("Bilbo Baggins", "burglar")));
            log.info("Preloading " + repository.save(new Employee("Frodo Baggins", "thief")));
        };
    }
}
```

- We preload some data in our in-memory database
- Spring Boot run ALL **CommandLineRunner** @Beans once the application starts
- This Runner needs a **copy** of the EntityRepository we created

A more intricated example - Controller

POLITECNICO MILANO 1863

```java
@RestController
class EmployeeController {

    private final EmployeeRepository repository;

    // The constructor automatically injects the repository within the controller
    EmployeeController(EmployeeRepository repository) {
        this.repository = repository;
    }

    // Aggregate root
    // tag::get-aggregate-root[]
    @GetMapping("/employees")
    List<Employee> all() {
        return repository.findAll();
    }
    // end::get-aggregate-root[]

    @PostMapping("/employees")
    Employee newEmployee(@RequestBody Employee newEmployee) {
        return repository.save(newEmployee);
    }
```

```java
// Single item

@GetMapping("/employees/{id}")
Employee one(@PathVariable Long id) {

    return repository.findById(id)
            .orElseThrow(() -> new EmployeeNotFoundException(id));
}

@PutMapping("/employees/{id}")
Employee replaceEmployee(@RequestBody Employee newEmployee, @PathVariable Long id) {

    return repository.findById(id)
            .map(employee -> {
                employee.setName(newEmployee.getName());
                employee.setRole(newEmployee.getRole());
                return repository.save(employee);
            })
            .orElseGet(() -> {
                newEmployee.setId(id);
                return repository.save(newEmployee);
            });
}

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

A more intricated example - Error Handling

```
@ControllerAdvice
class EmployeeNotFoundAdvice {

    @ResponseBody
    @ExceptionHandler(EmployeeNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    String employeeNotFoundHandler(EmployeeNotFoundException ex) {
        return ex.getMessage();
    }
}
```

- **@ResponseBody**: this advice is rendered straight into the response body
- **@ExceptionHandler**: configures the advice to only respond if an EmployeeNotFoundException is thrown
- **@ResponseStatus:** issue an HttpStatus.NOT_FOUND, i.e. an HTTP 404.

A more intricated example - Interact \w API

```
$ curl -v localhost:8080/employees
```

This will yield:

```
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8080 (#0)
> GET /employees HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200
< Content-Type: application/json;charset=UTF-8
< Transfer-Encoding: chunked
< Date: Thu, 09 Aug 2018 17:58:00 GMT
<
* Connection #0 to host localhost left intact
[{"id":1,"name":"Bilbo Baggins","role":"burglar"},{"id":2,"name":"Frodo Baggins","role":"thief"}]
```

A more intricated example - Interact \w API

```
$ curl -v localhost:8080/employees/99
```

You get...

```
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8080 (#0)
> GET /employees/99 HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 404
< Content-Type: text/plain;charset=UTF-8
< Content-Length: 26
< Date: Thu, 09 Aug 2018 18:00:56 GMT
<
* Connection #0 to host localhost left intact
Could not find employee 99
```

POLITECNICO
MILANO 1863

An Application Example

Software Engineering 2 Project

Academic Year: 2018-2019

Link to Github folder: riccardopoiani/spring-application-example

An Application Example

POLITECNICO MILANO 1863

- TrackMe is a company that wants to develop a software-based service allowing third parties to monitor the location and health status of individuals.

- This service is called Data4Help. The service supports the registration of individuals who, by registering, agree that TrackMe acquires their data (data acquisition can happen through smartwatches or similar devices).

An Application Example (cont'd)

POLITECNICO MILANO 1863

- Also, it supports the registration of third parties. After registration, these third parties can request access to the data of some specific individuals (we can assume, for instance, that they know an individual by his/her social security number or fiscal code in Italy). In this case, TrackMe passes the request to the specific individuals who can accept or refuse it.

An Application Example (cont'd)

POLITECNICO MILANO 1863

- Futhermore, third parties, can request access to anonymized data of groups of individuals (for instance, all those living in a certain geographical area, all those of a specific age range, etc.). These requests are handled directly by TrackMe that approves them if it is able to properly anonymize the requested data. For instance, if the third party is asking for data about 10-year-old children living in a certain street in Milano and the number of these children is two, then the third party could be able to derive their identity simply having people monitoring the residents of the street between 8.00 and 9.00 when kids go to school. Then, to avoid this risk and the possibility of a misuse of data, TrackMe will not accept the request. For simplicity, we assume that TrackMe will accept any request for which the number of individuals whose data satisfy the request is higher than 1000

An Application Example (cont'd)

POLITECNICO MILANO 1863

- As soon as a request for data is approved, TrackMe makes the previously saved data available to the third party. Also, it allows the third party to subscribe to new data and to receive them as soon as they are produced.

An Application Example (cont'd)

POLITECNICO MILANO 1863

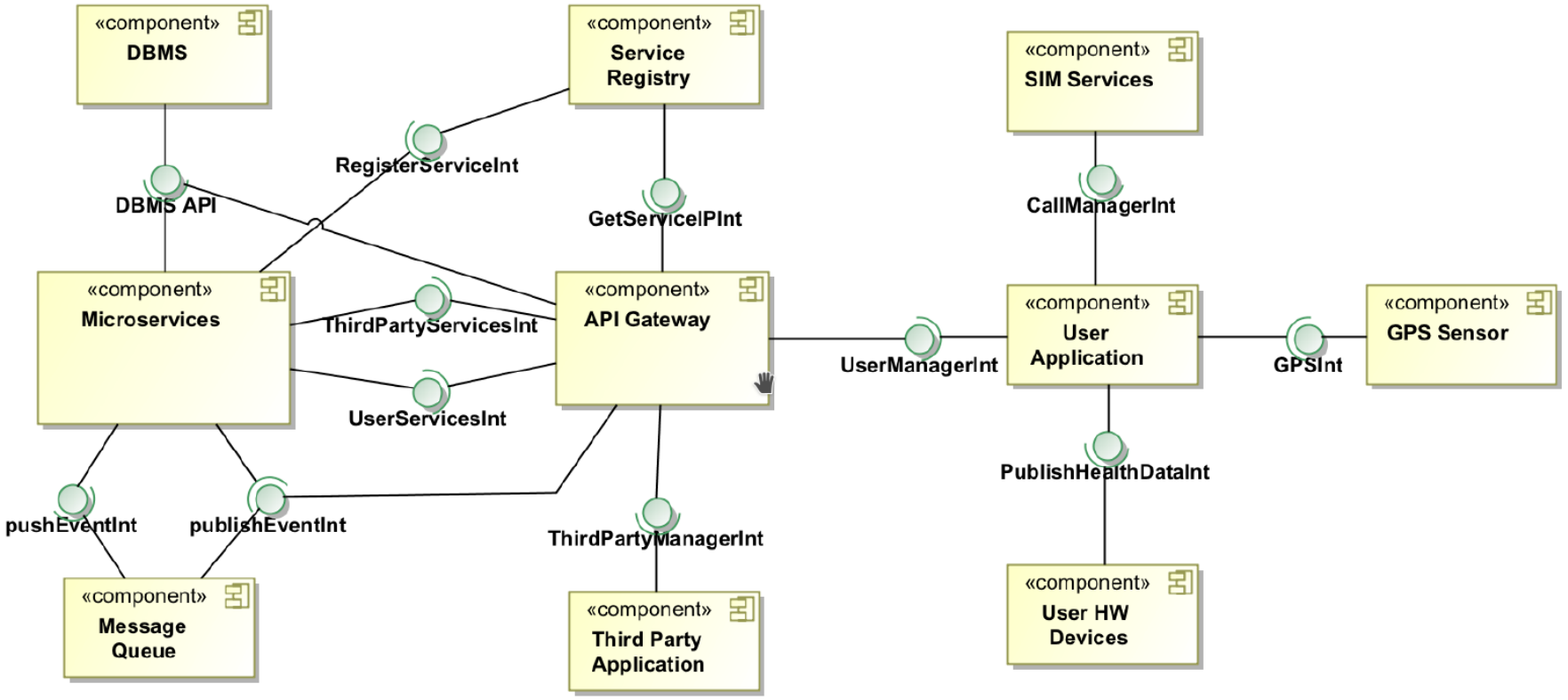
- Imagine now that, after some time, TrackMe realizes that a good part of its third-party customers wants to use the data acquired through Data4Help to offer a personalized and non-intrusive SOS service to elderly people. Therefore, TrackMe decides to build a new service, called AutomatedSOS, on top of Data4Help. AutomatedSOS monitors the health status of the subscribed customers and, when such parameters are below certain thresholds, sends to the location of the customer an ambulance, guaranteeing a reaction time of less than 5 seconds from the time the parameters are below the threshold.

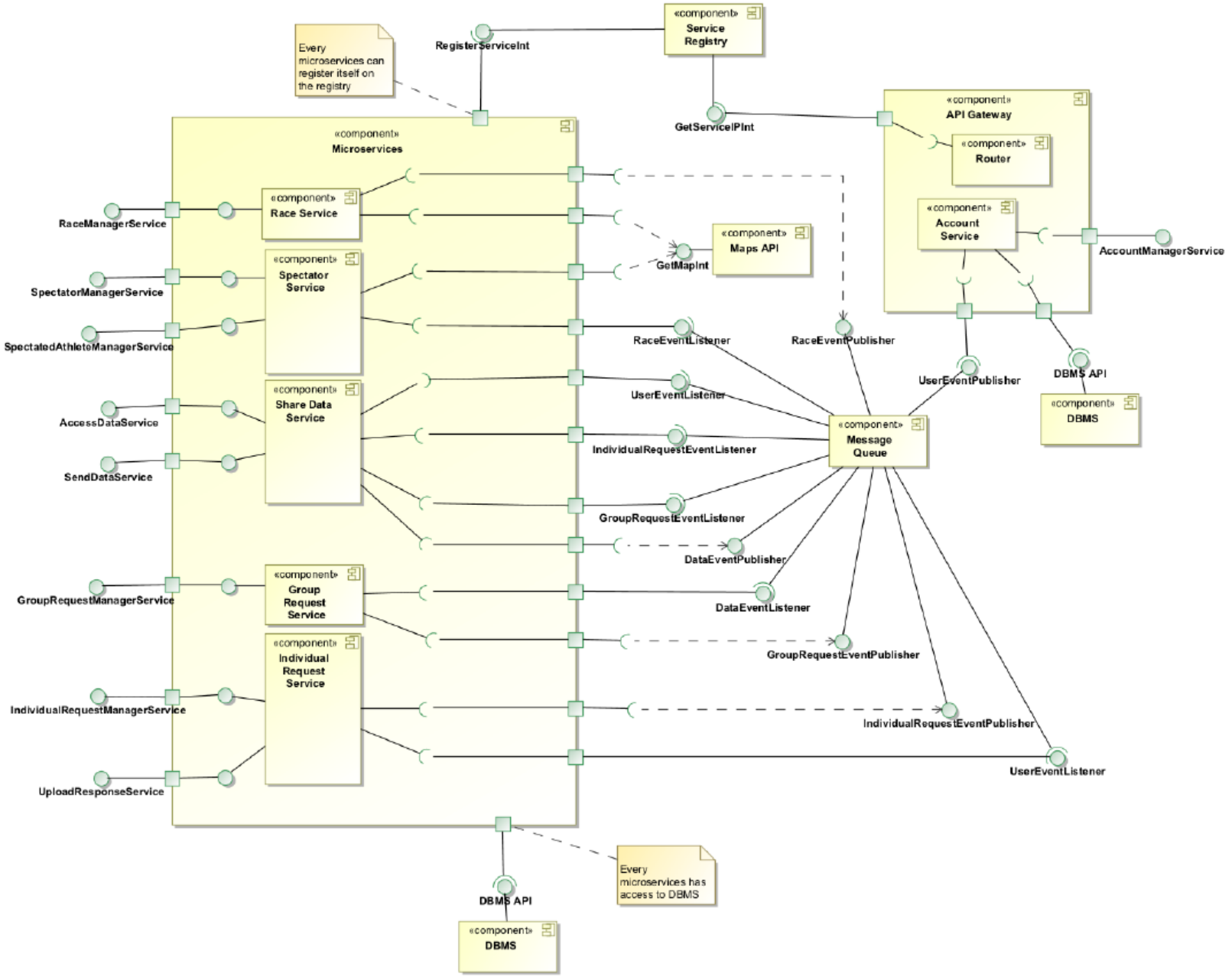
An Application Example (cont'd)

POLITECNICO MILANO 1863

- Finally, TrackMe realizes that another great source of revenues could be the development of a service to track athletes participating in a run. In this case, the service, called Track4Run, should allow organizers to define the path for the run, participants to enroll to the run, and spectators to see on a map the position of all runners during the run. Of course, also in this case, Track4Run will exploit the features offered by Data4He

An Application Example (cont'd)




«component»
Service Registry
RegisterServiceInt
Every microservices can register itself on the registry
GetServiceIPInt
«component»
API Gateway
«component»
Router
«component»
Account Service
AccountManagerService
«component»
Microservices
«component»
Race Service
RaceManagerService
«component»
Maps API
GetMapInt
«component»
Spectator Service
SpectatorManagerService
SpectatedAthleteManagerService
RaceEventListener
RaceEventPublisher
UserEventPublisher
DBMS API
«component»
DBMS
«component»
Share Data Service
AccessDataService
SendDataService
UserEventListener
IndividualRequestEventListener
«component»
Message Queue
GroupRequestEventListener
DataEventPublisher
DataEventListener
«component»
Group Request Service
GroupRequestManagerService
GroupRequestEventPublisher
«component»
Individual Request Service
IndividualRequestManagerService
IndividualRequestEventPublisher
UploadResponseService
UserEventListener
Every microservices has access to DBMS
DBMS API
«component»
DBMS