

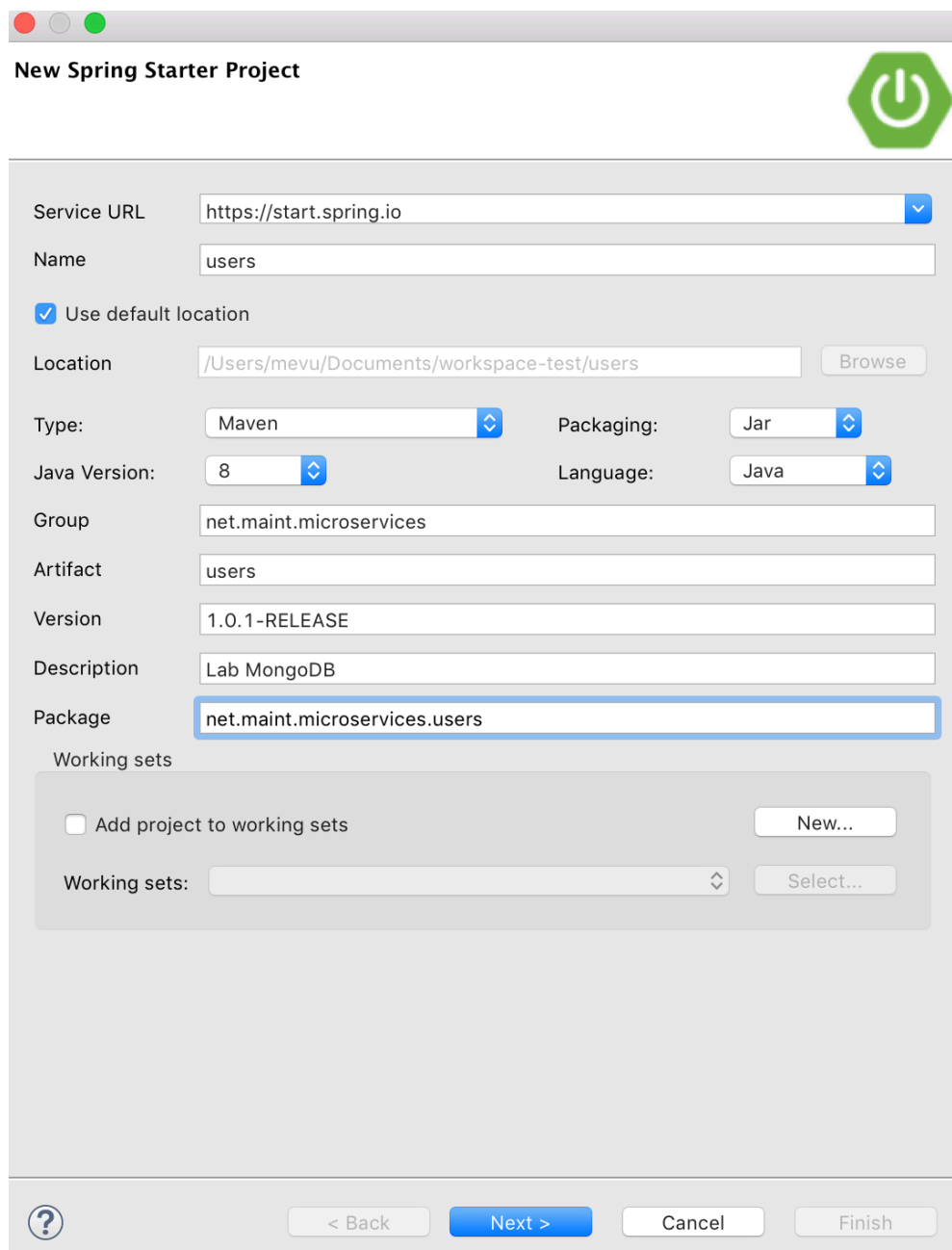
Instalación y ejecución de *MongoDB*

Antes de empezar

Instala MongoDB o comprueba si ya existe en tu sistema.

Por defecto las peticiones se escuchan en el puerto **27017**

Crea un proyecto basado en Spring:



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

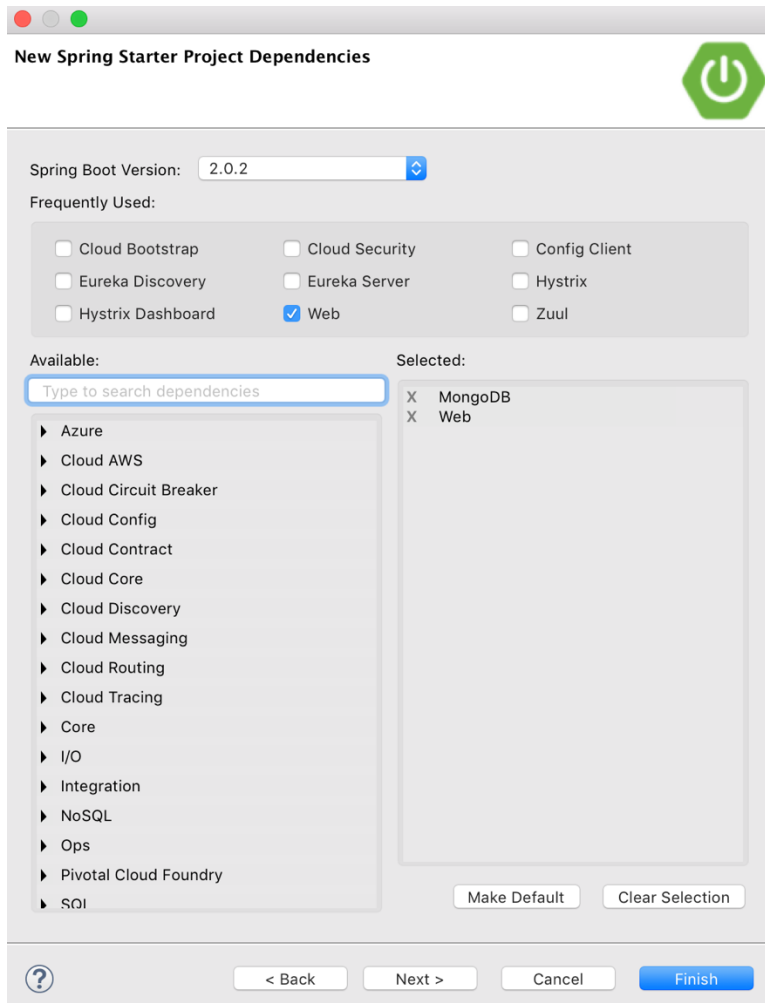
Description:

Package:

Working sets

☐ Add project to working sets

Working sets:



La configuración del fichero ***pom.xml*** del proyecto es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>net.maint.microservices</groupId>
  <artifactId>users</artifactId>
  <version>1.0.1-RELEASE</version>
  <packaging>jar</packaging>

  <name>users</name>
  <description>Lab MongoDB</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.2.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
```

```

    <properties>
        <project.build.sourceEncoding>UTF8
    </project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF8
    </project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter</artifactId>
            <version>2.0.0.BUILD-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>org.mongodb</groupId>
            <artifactId>mongo-java-driver</artifactId>
            <version>3.6.3</version>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.0.1</version>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger-ui</artifactId>
            <version>2.0.2</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

```

```
        </plugins>
    </build>
```

```
</project>
```

Implementación del microservicio

1 - Crear objeto de dominio

Crearemos un POJO muy sencillo para representar un ***User***, que será el que almacenemos en *MongoDB*

```
package net.maint.microservices.users.model;

import com.fasterxml.jackson.annotation.JsonPropertyOrder;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document
import java.io.Serializable;
import javax.validation.constraints.NotNull;

@Document(collection = "users")
@JsonPropertyOrder({"userId", "name"})
public class User implements Serializable{

    private static final long serialVersionUID = -7788619177798333712L;

    @Id
    @NotNull
    private String userId;
    @NotNull
    private String name;

    public String getUserId() {
        return userId;
    }
    public void setUserId(String userId) {
        this.userId = userId;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

Utilizamos la anotación **@Document** para definir un nombre de una colección cuando el objeto se guarde en *MongoDB*. En este caso, cuando el objeto “*user*” se guarde, se hará dentro de la colección “*users*”.

La anotación de *Jackson* **@JsonPropertyOrder** nos permite especificar el orden en que los campos del objeto java deberían ser serializados en JSON.

2 - Crear User repository

Lo primero de todo será crear el **interface** del repositorio que permita realizar varias operaciones CRUD sobre el objeto *User*

```
package net.maint.microservices.users.repository;

import net.maint.microservices.users.model.User;

public interface UserRepository{

    Optional<List<User>> findAll();

    public User saveUser(User user);

    public void updateUser(User user);

    public void deleteUser(String userId);
}
```

Y a continuación la implementación del *interface*:

```
package net.maint.microservices.users.repository.impl;

import net.maint.microservices.users.repository.UserRepository;
import net.maint.microservices.users.model.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoOperations;
import org.springframework.stereotype.Repository;
import org.springframework.util.Assert;

import java.util.List;
import java.util.Optional;

import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
```

```

@Repository
public class UserRepositoryImpl implements UserRepository{

    private final MongoOperations mongoOperations;

    @Autowired
    @Autowired
    public UserRepositoryImpl(MongoOperations mongoOperations) {
        Assert.notNull(mongoOperations);
        this.mongoOperations = mongoOperations;
    }

    //Find all users
    public Optional<List<User>> findAll() {
        List<User> users = this.mongoOperations.find(new Query(), User.class);
        Optional<List<User>> optionalUsers = Optional.ofNullable(users);
        return optionalUsers;
    }

    public Optional<User> findOne(String userId) {
        User d = this.mongoOperations.findOne(new
        Query(Criteria.where("userId").is(userId)), User.class);
        Optional<User> user = Optional.ofNullable(d);
        return user;
    }

    public User saveUser(User user) {
        this.mongoOperations.save(user);
        return findOne(user.getUserId()).get();
    }

    public void updateUser(User user) {
        this.mongoOperations.save(user);
    }

    public void deleteUser(String userId) {
        this.mongoOperations.findAndRemove(new
        Query(Criteria.where("userId").is(userId)), User.class);
    }
}

```

3 - Implementar *User Service*

Lo siguiente que haremos será crear el servicio (interface + implementación) que se conecte al repositorio de usuarios del paso anterior:

```

package net.maint.microservices.users.service;

import net.maint.microservices.users.model.User;
import java.util.List;

public interface UserService {

    List<User> findAll();

    User findById(String userId);

    User saveUser(User user);

    void updateUser(User user);

    void deleteUser(String userId);
}

```

A continuación la implementación de la *interface UserService*:

```

package net.maint.microservices.users.service.impl;

import net.maint.microservices.users.exception.UserNotFoundException;
import net.maint.microservices.users.model.User;
import net.maint.microservices.users.repository.UserRepository;
import net.maint.microservices.users.service.UserService;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

@Service("userService")
@Transactional
public class UserServiceImpl implements UserService {

    private static final Log log = LogFactory.getLog(UserServiceImpl.class);
    private UserRepository userRepository;

    @Autowired
    public UserServiceImpl(UserRepository userRepository){
        this.userRepository = userRepository;
    }

    public User findById(String userId) {

```

```

        Optional<User> user = userRepository.findOne(userId);
        if (user.isPresent()) {
            log.debug(String.format("Read userId '{}", userId));
            return user.get();
        } else {
            throw new UserNotFoundException(userId);
        }

    public List<User> findAll() {
        Optional<List<User>> user = userRepository.findAll();
        return user.get();
    }

    public User saveUser(User user) {
        return userRepository.saveUser(user);
    }

    public void updateUser(User user) {
        userRepository.updateUser(user);
    }

    public void deleteUser(String userId) {
        userRepository.deleteUser(userId);
    }
}

```

En el caso de no localizar un usuario por su *userId*, se lanzará la excepción *UserNotFoundException*

```

package net.maint.microservices.users.exception;

import org.springframework.core.NestedRuntimeException;

public class UserNotFoundException extends
    NestedRuntimeException {
    public UserNotFoundException(String userId) {
        super(String.format("User with Id '%s' not founded", userId));
    }
}

```

4 - Definir Controller

A continuación, la implementación del *UserController*:

```

package net.maint.microservices.users.controller;

import net.maint.microservices.users.exception.UserNotFoundException;
import net.maint.microservices.users.model.User;

```



```

import net.maint.microservices.users.service.UserService;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import javax.validation.Valid;

@RestController
@RequestMapping("users")
public class UsersController {

    private static final Log log = LogFactory.getLog(UsersController.class);

    private final UserService userService;
    private User user;

    @Autowired
    public UsersController(UserService userService) {
        this.userService = userService;
    }

    @RequestMapping(value="/{userId}",method = RequestMethod.GET)
    @ApiOperation(value = "Find an user", notes = "Return a user by Id" )
    public ResponseEntity<User> userById(@PathVariable String
userId) throws UserNotFoundException{
        log.info("Get userById");
        try{
            user = userService.findByUserId(userId);
        }catch(UserNotFoundException e){
            user = null;
        }
        return ResponseEntity.ok(user);
    }

    @RequestMapping(method = RequestMethod.GET)
    public ResponseEntity<List<User>> userById(){
        log.info("Get allUsers");
        return ResponseEntity.ok(userService.findAll());
    }

    @RequestMapping(method = RequestMethod.GET)
    @ApiOperation(value = "Find all user", notes = "Return all users" )
    public ResponseEntity<List<User>> userById(){
        log.info("Get allUsers");
        return ResponseEntity.ok(userService.findAll());
    }
}

```

```

    }

    @RequestMapping(value="/{userId}",method = RequestMethod.DELETE)
    @ApiOperation(value = "Delete an user", notes = "Delete a user by Id")
    public ResponseEntity<Void> deleteUser(@PathVariable String userId){
        log.info("Delete user " + userId);
        usersService.deleteUser(userId);
        return ResponseEntity.noContent().build();
    }

    @RequestMapping(method=RequestMethod.POST)
    @ApiOperation(value = "Create an user", notes = "Create a new user")
    public ResponseEntity<User> saveUser(@RequestBody @Valid User
user){
        log.info("Save new user");
        return ResponseEntity.ok(usersService.saveUser(user));
    }
}

```

5 - Crear aplicación ejecutable

Por último, solo nos queda crear la clase que ejecute la aplicación *Spring Boot*:

```

package net.maint.microservices.users;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
public class UsersApplication {

    public static void main(String[] args) {
        SpringApplication.run(UsersApplication.class, args);
    }

}

```

Por defecto, cuando inicias una aplicación spring boot , se busca un fichero llamado `application.properties` o `application.yml` para acceder a su

configuración, el cual deberá estar ubicado en la carpeta resources de nuestro proyecto. Su configuración es la siguiente:

```
# Spring properties
spring:
  data:
    mongodb:
      host: localhost
      port: 27017
      uri: mongodb://localhost/test

# HTTP Server
server:
  port: 4444 # HTTP (Tomcat) port
```

Prueba que el microservicio funciona correctamente

Probaremos el servicio por medio del interface RESTful que ofrece. Para facilitarte esta tarea, te recomiendo utilizar un cliente que te permita realizar peticiones HTTP, como Postman o Advance REST Client

Antes de nada, recuerda tener levantado el demonio de *MongoDB* mediante ***mongod***.

A continuación, te indico la configuración que debes establecer en las peticiones HTTP para ejecutar las distintas operaciones CRUD del microservicio:

- **Crear nuevo usuario:**
 - *POST*: http://localhost:4444/users
 - *Header*
 - *Content-Type: application/json*
 - *Accept: application/json*
 - *Body*:
 - `{"userId": "1", "name": "Rob"}`
 - `{"userId": "2", "name": "Peter"}`
 - `{"userId": "1", "name": "Rob"}`
- **Obtener un usuario por *userId***
 - *GET*: http://localhost:4444/users/1
 - *Nota: userId=1*
- **Obtener todos los usuario**
 - *GET*: http://localhost:4444/users
- **Modificar usuario:**

- *PUT*: http://localhost:4444/users
- *Header*
 - *Content-Type: application/json*
 - *Accept: application/json*
- *Body*:
 - *{“userId”:”1”,”name”:”John”}*
- **Eliminar un usuario por *userId***
 - *DELETE*: http://localhost:4444/users/1
 - *Nota: userId=1*