12 DE ENERO DE 2018

CONTROL DE VERSIONES Y DE TRABAJO

PROYECTO FCT PROFILE

MARCO BORREGUERO TRIGUEROS
I.E.S RAMÓN DEL VALLE INCLÁN
2º DAM

Índice

Contenido

Contenidos

Versión: v1.0-CRUD

Objetivos: Implementar en CompanyServiceIMPL el CRUD de Company, utilizando los métodos que nos proporciona MongoDBRepository. Que consisten en creaciones, internas, de queries para el manejo de los datos.

Proceso:

Se hace uso de la inyección de dependencias para el objeto Company:

```
@Autowired
private CompanyRepository companyRepository;
```

Se implementan los métodos triviales:

```
//Insert:
@Override
public Company insert(Company company) {
    return companyRepository.insert(company);
}

//Update:
@Override
public Company update(Company company) {
    return companyRepository.save(company);
}

//SelectAll:
@Override
public Stream<Company> selectAll() {
    return companyRepository.findAll().stream();
}
```

Documentación:

1. https://www.baeldung.com/spring-data-mongodb-tutorial
2. https://docs.spring.io/spring-data/mongodb/docs/1.2.0.RELEASE/reference/html/mongo.repositories.html
3. https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/
4. https://docs.spring.io/spring-data/mongodb/docs/current/api/org/springframework/data/mongodb/repository/MongoRepository.html

Resultado (Commit): Inicio del proyecto. Implementación de los métodos insert, update y selectAll de CompanyService.

Commid ID: 819f18e49f6d6d17e13ec920b3f54fd6e8bc4758

Versión: v1.1-CRUD

Objetivos: Implementar en CompanyServiceIMPL el CRUD de Company, utilizando los métodos que nos proporciona MongoDBRepository. Que consisten en creaciones, internas, de queries para manejar los datos.

Proceso:

Se implementan los métodos restantes:

```
//Select por ID:
```

```java
@Override
public Optional<Company> selectById(int id) {

    String ids = String.valueOf(id);

    return companyRepository.findById(ids);
}
```

```java
//Delete:
@Override
public void delete(int id) {

    String ids = String.valueOf(id);

    companyRepository.deleteById(ids);

}
```

Documentación:

1. https://www.javatpoint.com/spring-mvc-crud-example

2. https://www.journaldev.com/3531/spring-mvc-hibernate-mysql-integration-crud-example-tutorial

Resultado (Commit): Implementados Delete y SelectByID del CRUD de CompanyService.

Commid ID: b4f63eb5f0b530e2a20f97d2637404013f1b52bf

Versión: v1.2-API_CRUD

Objetivos: Implementar el CRUD en la API (paquete controller).

Proceso:

Se ha añadido el TODO a la clase UserApi para crear el Validator del DNI.

```
//TODO Validate DNI with Spring Validator
```

Se han añadido los métodos CRUD básicos a la interfaz de la API de Company (interface CompanyApi):

```
//INSERT (PUT)
@PutMapping
ResponseEntity<Company> insertCompany(@RequestBody Company company);
```

```
//UPDATE (POST)
@PostMapping
ResponseEntity<Company> updateCompany(@RequestBody Company company);
```

```
//DELETE (DELETE)
@DeleteMapping(EndPointUris.ID)
ResponseEntity<Void> deleteCompany(@PathVariable(value = "id") String
id);
```

```
//SELECT ALL (GET)
@GetMapping
ResponseEntity<List<Company>> selectAllCompanies();
```

```
//SELECT BY ID (GET)
@GetMapping(EndPointUris.ID)
ResponseEntity<Company> selectCompanyById(@PathVariable(value = "id")
String id);
```

Antes, se ha añadido la anotación "@RequestMapping" a la interfaz, que especifica la URI que hay que usar para acceder a esta API. En este caso, se ha creado una nueva URI que gestiona las peticiones HTTP para la API de "Company" en la clase EndPointUris:

```
//INTERFAZ CompanyApi

@RequestMapping(EndPointUris.COMPANY)
public interface CompanyApi {…}
```

```
//CLASE EndPointUris
```

```java
public final class EndPointUris {

    //Uri for COMPANY
    public static final String COMPANY = "/company";

    public static final String USER = "/user";

    public static final String ID = "/{id}";
}
```

Se ha detectado una inconsistencia en la clase CompanyServiceImpl, ya que, según el código dado, a los métodos delete y selectById se les pasa como parámetro una id de tipo int. Pero, en la estructura del DAO de Company, las ids de éstas son de tipo String:

```java
//CLASE CompanyServiceImpl

@Override
public void delete(int id) {

    //Companies have an "id" as a String. So, why is an int here??

    String ids = String.valueOf(id);

    companyRepository.deleteById(ids);

}

@Override
public Optional<Company> selectById(int id) {

    //Companies have an "id" as a String. So, why is an int here??

    String ids = String.valueOf(id);

    return companyRepository.findById(ids);
}
```

```java
//CLASE Company
public class Company {

    @Id
    private String id;

    private String CIF;

    private String name;

    private CompanySize size;
```

```
    private String city;
}
```

Se ha encontrado otro problema en la clase de tipo Enum “CompanySize”, en la cual, la notación @AllArgsConstructor estaba dando problemas. El compilador, indica que se están creando los tipo Enum utilizando un constructor que no requiere parámetros. Esto, creo que se debe a que los atributos de esta clase se colocan después de la creación de los tipo Enum, y la notación @AllArgsConstructor no los detecta.

```
//CLASE ENUM CompanySize
@Getter
//@AllArgsConstructor
public enum CompanySize {

    MICRO(10, "<"), SMALL(50, "<"), MEDIUM(250, "<"), LARGE(250, ">");

    private int size;
    private String symbol;

    CompanySize(int i, String s) {
    }
}
```

Se ha comentado dicha notación Lombok y se ha creado un constructor con dichos parámetros.

Documentación:

1. https://www.baeldung.com/building-a-restful-web-service-with-spring-and-java-based-configuration
2. http://www.wandercosta.com/lombok-constructors/
3. https://www.arquitecturajava.com/spring-mvc-requestmapping/
4. https://www.baeldung.com/spring-requestmapping
5. https://www.adictosaltrabajo.com/2016/02/03/como-reducir-el-codigo-repetitivo-con-lombok/
6. https://jlordiales.me/2012/12/13/the-builder-pattern-in-practice/

Resultado (Commit): Definición de métodos CRUD básicos de CompanyApi.

Commid ID: 0b4a187dff4b95523d308dd39a466919a81b7f36

Versión: v1.3-Company_API_CRUD_Extendido

Objetivos: Implementar el API de Company para que sea totalmente funcional. Implementar los métodos extra para buscar Compañías por ciudad y por tamaño.

Proceso: Primero se definen los métodos pertenecientes al CRUD de Company en la interfaz CompanyService, que es la que nos da los servicios que consumirá nuestra API, añadiéndose los dos restantes:

```
public interface CompanyService {

    Company insert(Company company);

    Company update(Company company);

    //void delete(int id);
    void delete(String id);

    Stream<Company> selectAll();

    //Optional<Company> selectById(int id);
    Optional<Company> selectById(String id);

    //TODO Get companies of a given size
    Stream<Company> selectBySize(int size);

    //TODO Companies grouped by city
    Stream<Company> selectByCity(String city);

}
```

```
//Implementación de los nuevos métodos de
CompanyService
```

```
@Override
public Optional<Company> selectById(String id) {
    return companyRepository.findById(id);
}

//TODO Get companies of a given size
@Override
public Stream<Company> selectBySize(int size) {

    CompanySize cSize =
Arrays.stream(CompanySize.values()).filter(companySize ->
(companySize.getSymbol().equals("<") && size < companySize.getSize())
|| (companySize.getSymbol().equals(">") && size >=
companySize.getSize())).findFirst().get();

    return
companyRepository.findAll(Example.of(Company.builder().size(cSize).bui
ld())).stream();
}

//TODO Companies grouped by city
@Override
public Stream<Company> selectByCity(String city) {
    return
companyRepository.findAll(Example.of(Company.builder().city(city).buil
d())).stream();
}
```

Se ha corregido un error grave relacionado con Lombok. Ya que, el funcionamiento de las anotaciones no era el correcto. Se ha instalado el plugin para el IDE IntelliJ de Lombok, lo que ahora permite la utilización de dichas anotaciones. Esto conlleva, que se realicen cambios, de nuevo, en la clase de tipo Enum CompanySize. Ya que se realizaron cambios debido a una confusión por el anterior error con Lombok en la versión 1.2:

```
//Esta clase vuelve a su estado original. Se
deshacen los cambios realizados anteriormente.
@Getter
@AllArgsConstructor
public enum CompanySize {

    MICRO(10, "<"), SMALL(50, "<"), MEDIUM(250, "<"), LARGE(250, ">");

    private int size;
    private String symbol;

}
```

Respecto a la API de Company, la interfaz que define sus métodos quedaría así:

```java
@RequestMapping(EndPointUris.COMPANY)
public interface CompanyApi {

    //TODO CRUD (validate CIF with Validator Spring in insert and
update)

    @PutMapping
    ResponseEntity<Company> insertCompany(@RequestBody Company
company);

    @PostMapping
    ResponseEntity<Company> updateCompany(@RequestBody Company
company);

    @DeleteMapping(EndPointUris.ID)
    ResponseEntity<Void> deleteCompany(@PathVariable(value = "id")
String id);

    @GetMapping
    ResponseEntity<List<Company>> selectAllCompanies();

    @GetMapping(EndPointUris.ID)
    ResponseEntity<Company> selectCompanyById(@PathVariable(value =
"id") String id);

    //TODO Get companies of a given size
    @GetMapping(EndPointUris.SIZE)
    ResponseEntity<List<Company>>
selectCompanyBySize(@PathVariable(value = "size") int size);

    //TODO Companies grouped by city
    @GetMapping(EndPointUris.CITY)
    ResponseEntity<List<Company>>
selectCompanyByCity(@PathVariable(value = "city") String city);
}
```

En su implementación, esta API consume los servicios de CompanyService, por lo que es necesaria la implementación de estos servicios, tal y como se ha realizado anteriormente.

La implementación de esta interfaz la realiza el controlador, o Controller, de nuestra aplicación siguiendo la estructura del MVC:

```java
@RestController
public class CompanyController implements CompanyApi {

    @Autowired
    private CompanyService companyService;

    // TODO Validate CIF with Validator Spring
    @Override
```

```
    public ResponseEntity<Company> insertCompany(@RequestBody Company
company) {
        return ResponseEntity.ok(companyService.insert(company));
    }

    // TODO Validate DNI with Validator Spring
    @Override
    public ResponseEntity<Company> updateCompany(@RequestBody Company
company) {
        return ResponseEntity.ok(companyService.update(company));
    }

    @Override
    public ResponseEntity<Void> deleteCompany(@PathVariable(value =
"id") String id) {

        companyService.delete(id);

        return ResponseEntity.ok().build();
    }

    @Override
    public ResponseEntity<List<Company>> selectAllCompanies() {
        return
ResponseEntity.ok(companyService.selectAll().collect(Collectors.toList
()));
    }

    @Override
    public ResponseEntity<Company>
selectCompanyById(@PathVariable(value = "id") String id) {
        return ResponseEntity.of(companyService.selectById(id));
    }

    //TODO Get companies of a given size
    @Override
    public ResponseEntity<List<Company>>
selectCompanyBySize(@PathVariable(value = "size") int size) {
        return
ResponseEntity.ok(companyService.selectBySize(size).collect(Collectors
.toList()));
    }

    //TODO Companies grouped by city
    @Override
    public ResponseEntity<List<Company>>
selectCompanyByCity(@PathVariable(value = "city") String city) {
        return
ResponseEntity.ok(companyService.selectByCity(city).collect(Collectors
.toList()));
    }
}
```

Se utiliza la anotación @RestController para indicar, en la estructura MVC de Spring, que esta clase es la clase controladora del servicio Rest de Company.

Documentación:

1. https://www.baeldung.com/rest-with-spring-series
2. https://www.baeldung.com/building-a-restful-web-service-with-spring-and-java-based-configuration
3. https://projectlombok.org/setup/intellij
4. https://www.arquitecturajava.com/spring-mvc-requestmapping/
5. https://www.baeldung.com/lombok-builder

Resultado (Commit): Implementación completa de la API de Company: métodos selectCompanyByCity y selectCompanyBySize. Corrección de errores en CompanySize.

Commid ID: e7326ff9074dd8041b3036601a7e774ef13cf413

Versión: v1.4-User_API_CRUD_Extendido

Objetivos: Completar la implementación de la API y los servicios que consumirá ésta sobre los datos de User.

Proceso:

Primero se definen los métodos de los servicios en la interfaz UserService, tanto los CRUD básicos como los extendidos:

```
//Interfaz UserService

public interface UserService {

    User insert(User user);

    User update(User user);

    void delete(String id);

    Stream<User> selectAll();

    Optional<User> selectById(String id);

    Stream<User> selectByCompany(String companyReference);

    Stream<User> selectByCompanyCity(String city);
}
```

En nuestro caso, se definen como nuevos los métodos “selectByCompany” y “selectByCompanyCity”.

Luego, se procede a la implementación de estos nuevos métodos en la clase UserServiceImpl:

```
//Implementación de selectByCompany y
selectByCompanyCity

@Override
public Stream<User> selectByCompany(String companyReference) {

    Company comp =
companyService.selectById(companyReference).orElse(companyService.sele
ctAll().filter(company ->
company.getName().equals(companyReference)).findFirst().get());


    return
userRepository.findAll(Example.of(User.builder().company(comp).build()
)).stream();
}

@Override
public Stream<User> selectByCompanyCity(String city) {

    Stream<Company> companies = companyService.selectByCity(city);

    return userRepository.findAll().stream().filter(user ->
```

```
!user.getCompany().equals(companies.iterator().next()));
}
```

Ahora, será necesario establecer los métodos CRUD en nuestra API, es decir, definir todos los métodos GET, POST, PUT y DELETE que serán llamados cuando se utilice nuestra API.

Esto, se hace en la interfaz UserApi, en la que se definen estos métodos y se mapean como métodos de una api a través de las anotación de Spring (@RequestMapping para el mapeo general de las URIS, @GetMapping para el mapeo del método get, @PutMapping para el put, etc.):

```
//Interfaz UserApi

 @RequestMapping(EndPointUris.USER)
public interface UserApi {

    //TODO Validate DNI with Spring Validator

    @PutMapping
    ResponseEntity<User> insertUser(@RequestBody User user);

    @PostMapping
    ResponseEntity<User> updateUser(@RequestBody User user);

    @DeleteMapping(EndPointUris.ID)
    ResponseEntity<Void> deleteUser(@PathVariable(value = "id") String
id);

    @GetMapping
    ResponseEntity<List<User>> selectAllUsers();

    @GetMapping(EndPointUris.ID)
    ResponseEntity<User> selectUserById(@PathVariable(value = "id")
String id);

    //TODO Through the name or id of a company get all the users
    @GetMapping
    ResponseEntity<List<User>>
selectUsersFromACompany(@PathVariable(value = "companyref") String
companyRef);

    //TODO List of users who do not work in your city
    @GetMapping(EndPointUris.CITY)
    ResponseEntity<List<User>>
selectUsersNotWorkingInYourCity(@PathVariable(value = "city") String
city);

}
```

Luego, se implementan los nuevos métodos añadidos a esta interfaz (“selectUsersFromACompany” y “selectUsersNotInYourCity” en la clase controladora del api de User en nuestra aplicación:

```
//Clase UserController

//TODO Through the name or id of a company get all the users
@Override
public ResponseEntity<List<User>> selectUsersFromACompany(String
companyRef) {
    return
ResponseEntity.ok(userService.selectByCompany(companyRef).collect(Coll
ectors.toList()));
}

//TODO List of users who do not work in your city
@Override
public ResponseEntity<List<User>>
selectUsersNotWorkingInYourCity(String city) {
    return
ResponseEntity.ok(userService.selectByCompanyCity(city).collect(Collec
tors.toList()));
}
```

Documentación:

1. https://www.baeldung.com/rest-with-spring-series

2. https://www.adictosaltrabajo.com/2016/06/23/uso-basico-de-java-8-stream-y-lambdas/

3. https://dzone.com/articles/8-ways-of-creating-a-stream-object-in-java-8

4. https://stackoverflow.com/questions/51988182/spring-boot-service-class-calling-another-service-class

5. https://stackoverflow.com/questions/32798323/mvc-practices-service-within-another-service

Resultado (Commit): Implementación completa de la API de User: métodos selectUsersFromACompany y selectUsersNotWorkingInYourCity.

Commid ID: 077f5125474673321289daf549d700d8968bdac6

Versión: v2.0-API_Improvement

Objetivos: Se cambia de rama (Branch) del proyecto, ya que se encuentran problemas con la BD embebida. Se corrigen errores y se implementan mejoras. Se realizan algunos test iniciales con POSTMAN.

Proceso:

Ya que se ha cambiado de rama, ha sido necesario realizar el "git checkout" a la nueva rama del proyecto. Luego, se ha puesto al día esta rama con el código antes implementado.

Se han modificado las "end point Uris", o identificadores únicos de recursos, que se usan para referenciar los recursos suministrados por la API. Todo ello siguiendo una serie de buenas prácticas recomendadas para el mejor entendimiento con los potenciales clientes, o usuarios, que quieran consumir dicha API.

```
public final class EndPointUris {

    public static final String COMPANY = "/companies";

    public static final String SIZE = "/size/{size}";

    public static final String USER = "/users";

    public static final String ID = "/{id}";
```

```
    public static final String CITY = "/city/{city}";

    public static final String COMPANYREF =
"/companyref/{companyref}";
}
```

Y se añaden las referencias a éstos, en las APIs correspondientes (UserApi y CompanyApi)

```
//UserApi
@RequestMapping(EndPointUris.USER)
public interface UserApi {

    //TODO Validate DNI with Spring Validator

    @PostMapping
    ResponseEntity<User> insertUser(@RequestBody User user);

    @PutMapping(EndPointUris.ID)
    ResponseEntity<User> updateUser(@PathVariable(value = "id")
@RequestBody User user);

    @DeleteMapping(EndPointUris.ID)
    ResponseEntity<Void> deleteUser(@PathVariable(value = "id") String
id);

    @GetMapping
    ResponseEntity<List<User>> selectAllUsers();

    @GetMapping(EndPointUris.ID)
    ResponseEntity<User> selectUserById(@PathVariable(value = "id")
String id);

    //TODO Through the name or id of a company get all the users
    @GetMapping(EndPointUris.COMPANYREF)
    ResponseEntity<List<User>>
selectUsersFromACompany(@PathVariable(value = "companyref") String
companyRef);

    //TODO List of users who do not work in your city
    @GetMapping(EndPointUris.CITY)
    ResponseEntity<List<User>>
selectUsersNotWorkingInYourCity(@PathVariable(value = "city") String
city);

}
```

```
//CompanyApi
@RequestMapping(EndPointUris.COMPANY)
public interface CompanyApi {

    //TODO CRUD (validate CIF with Validator Spring in insert and
```

```
update)

    @PostMapping
    ResponseEntity<Company> insertCompany(@RequestBody Company
company);

    @PutMapping
    ResponseEntity<Company> updateCompany(@RequestBody Company
company);

    @DeleteMapping(EndPointUris.ID)
    ResponseEntity<Void> deleteCompany(@PathVariable(value = "id")
String id);

    @GetMapping
    ResponseEntity<List<Company>> selectAllCompanies();

    @GetMapping(EndPointUris.ID)
    ResponseEntity<Company> selectCompanyById(@PathVariable(value =
"id") String id);

    //TODO Get companies of a given size
    @GetMapping(EndPointUris.SIZE)
    ResponseEntity<List<Company>>
selectCompanyBySize(@PathVariable(value = "size") int size);

    //TODO Companies grouped by city
    @GetMapping(EndPointUris.CITY)
    ResponseEntity<List<Company>>
selectCompanyByCity(@PathVariable(value = "city") String city);
}
```

Ahora, la estructura de las URIs de los recursos queda así:

- **Company** :
  - Collection: */companies*
  - Singletone: */company/{id}*
  - Filters:
    - Size: */companies/size/{size}*
    - City: */companies/city/{city}*

- **User**:
  - Collection: */users*

- Singletone: */users/{id}*
- Filters:
  - Company Reference (ID or name): */users/companyref/{companyref}*
  - City: */users/city/{city}*

Además, se han realizado algunos tests con Postman para observar si el funcionamiento básico de estas URIs era el correcto. Los resultados de éstos se mostrarán en la próxima versión de este documento. No obstante, se deja una vista previa de las dos pruebas básicas de la colección de datos de Company y User:

GET localhost:8080/companies


```
[
  {
    "id": "5c34e8eba4ac3d204c8c38fe",
    "name": "Profile",
    "size": "MEDIUM",
    "city": "Sevilla",
    "cif": "6as54das65d4"
  },
  {
    "id": "5c34e8eba4ac3d204c8c38ff",
    "name": "Google",
    "size": "SMALL",
    "city": "Barcelona",
    "cif": "6a345khj345"
  },
  {
    "id": "5c34e8eba4ac3d204c8c3900",
    "name": "Amazon",
    "size": "MICRO",
    "city": "Madrid",
    "cif": "43b534j534j"
  },
  {
    "id": "5c34e8eba4ac3d204c8c3901",
    "name": "IBM",
    "size": "MEDIUM",
    "city": "Madrid",
    "cif": "35bn34b5j34"
  },
  {
    "id": "5c34e8eba4ac3d204c8c3902",
    "name": "Shazam",
```

```
    "size": "LARGE",
    "city": "Huelva",
    "cif": "dfgdfkg7854"
  },
  {
    "id": "5c34e8eba4ac3d204c8c3903",
    "name": "Adidas",
    "size": "MEDIUM",
    "city": "Sevilla",
    "cif": "34b5hj345hj"
  } … CONTINUES
```


GET localhost:8080/users


```
[
  {
    "id": "5c34e8eba4ac3d204c8c3905",
    "dni": "00000000A",
    "name": "Juan",
    "surname": "Herrera",
    "city": "Sevilla",
    "company": {
      "id": "5c34e8eba4ac3d204c8c38fe",
      "name": "Profile",
      "size": "MEDIUM",
      "city": "Sevilla",
      "target": {
        "id": "5c34e8eba4ac3d204c8c38fe",
        "name": "Profile",
        "size": "MEDIUM",
        "city": "Sevilla",
        "cif": "6as54das65d4"
      },
      "cif": "6as54das65d4"
    }
  },
  {
    "id": "5c34e8eba4ac3d204c8c3906",
    "dni": "00000000B",
    "name": "Pepe",
    "surname": "Herrera",
    "city": "Madrid",
    "company": {
      "id": "5c34e8eba4ac3d204c8c38ff",
      "name": "Google",
      "size": "SMALL",
      "city": "Barcelona",
      "target": {
        "id": "5c34e8eba4ac3d204c8c38ff",
        "name": "Google",
        "size": "SMALL",
        "city": "Barcelona",
        "cif": "6a345khj345"
```

```
        },
        "cif": "6a345khj345"
    }
},
{
    "id": "5c34e8eba4ac3d204c8c3907",
    "dni": "00000000C",
    "name": "Paco",
    "surname": "Herrera",
    "city": "Sevilla",
    "company": {
        "id": "5c34e8eba4ac3d204c8c3900",
        "name": "Amazon",
        "size": "MICRO",
        "city": "Madrid",
        "target": {
            "id": "5c34e8eba4ac3d204c8c3900",
            "name": "Amazon",
            "size": "MICRO",
            "city": "Madrid",
            "cif": "43b534j534j"
        },
        "cif": "43b534j534j"
    }
}, … CONTINUES
```


Documentación:

1. https://restfulapi.net/http-methods/
2. https://restfulapi.net/resource-naming/
3. https://restfulapi.net/rest-put-vs-post/
4. https://medium.com/@sylwit/what-to-test-on-your-crud-rest-api-f3e659648f12
5. https://en.wikipedia.org/wiki/Uniform_Resource_Identifier
6. https://losandestraining.com/2017/08/30/como-probar-una-api-rest-con-postman/

Resultado (Commit): Cambio de rama y puesta en común con la rama master. Cambios en la estructura de referencias de las URIs.

Commid ID: 8bf756225d9e0bc0a79f595f7d59b0f8ccce29d0

Versión: v2.1-Spring_Validator

Objetivos: Corrección de errores en el funcionamiento de algunos servicios REST del api de User. Implementación del Spring Validator para el CIF y el DNI.

Proceso:

Tras ciertos tests iniciales, se encontraron fallos en el funcionamiento de dos de los servicios de tipo GET del api de User, en concreto dos que contenían filtros. Estos dos servicios se corresponden a dos métodos del api, que son:

```
//Interfaz UserApi

@GetMapping(EndPointUris.COMPANYREF)
ResponseEntity<List<User>> selectUsersFromACompany(@PathVariable(value
= "companyref") String companyRef);

@GetMapping(EndPointUris.CITY)
ResponseEntity<List<User>>
selectUsersNotWorkingInYourCity(@PathVariable(value = "city") String
city);
```

Se ha cambiado la implementación de ambos servicios, que no devolvían los recursos correctamente. Esta implementación se realiza en el paquete de servicios, donde se encuentra la lógica de la aplicación.

```
//Clase UserServiceImpl

@Override
public Stream<User> selectByCompany(String companyReference) {

    Company comp =
companyService.selectById(companyReference).orElseGet(() ->
companyService.selectAll().filter(company ->
company.getName().equals(companyReference)).findFirst().get());

    return
userRepository.findAll(Example.of(User.builder().company(comp).build()
)).stream();
}

@Override
public Stream<User> selectByCompanyCity(String city) {

    List<String> companies =
companyService.selectByCity(city).map(Company::getId).collect(Collecto
rs.toList());
    List<User> users = userRepository.findAll().stream().filter(user -
>
!companies.contains(user.getCompany().getId())).collect(Collectors.toL
ist());


    return users.stream();
}
```

Además, en esta versión, se ha realizado la implementación de los validadores para los atributos DNI, del tipo User, y CIF, del tipo Company.

Estos validadores se han implementado utilizando las funcionalidades de Spring, que se concentran en lo llamado Spring Validator. Lo cual consiste en crear una nueva anotación “@AtributteConstrait” que le indica al sistema qué formato, o propiedades, ha de tener un atributo concreto de un tipo. Para, luego, indicarselo al controlador Rest y que se puedan controlar los datos introducidos por los usuarios.

Para ello, primero se debe añadir las dependencias correspondientes a Maven, en este caso la relacionada con Sprind Validator.

Luego, debemos crear una interfaz por cada anotación que deseemos

crear. En nuestro caso, se crean dos interfaces que supondrán dos nuevas anotaciones para validar ambos atributos, DNI y CIF:

```
//Interfaz CIFConstraint

@Constraint(validatedBy = CIFValidator.class)
@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface CIFConstraint {

    String message() default "Invalid CIF number";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

}
```

```
//Interfaz DNIConstraint

@Constraint(validatedBy = DNIValidator.class)
@Target({ElementType.METHOD, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface DNIConstraint {

    String message() default "Invalid DNI number";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

}
```

En esta interfaz, se establece el mensaje de error por defecto, entre otras cosas. Con la anotación "@Constraint(validatedBy = …)", indicamos qué clase será la encargada de validar el campo. Ésta, será una clase que implementará ciertos métodos que indicarán las reglas de validación para el campo deseado.

En nuestro caso, tendremos dos clases, una referida a las reglas de validación del DNI y otra para el CIF:

```
//Clase CIFValidator
```

```java
public class CIFValidator implements
ConstraintValidator<CIFConstraint, String> {


    private final String cifControlDigit = "JABCDEFGHI";
    private final String cifChar = "KPQRSNW";


    @Override
    public void initialize(CIFConstraint constraintAnnotation) {

    }

    @Override
    public boolean isValid(String cif, ConstraintValidatorContext
constraintValidatorContext) {
        return cif != null && cif.length() <= 9 &&
cif.equals(calculateCif(cif.substring(0, 8)));
    }

    private String calculateCif(String cif) {

        System.out.println("cif=" + cif + " res=" +
calculateControlDigit(cif));


        return cif + calculateControlDigit(cif);
    }

    private String calculateControlDigit(String cif) {
        String str = cif.substring(1, 8);
        String head = cif.substring(0, 1);
        int evenNumbersSum = 0;
        int oddNumbersSum = 0;
        int totalSum;

        for (int i = 1; i < str.length(); i += 2) {
            int aux = Integer.parseInt("" + str.charAt(i));
            evenNumbersSum += aux;
        }

        for (int i = 0; i < str.length(); i += 2) {
            oddNumbersSum += evenPosition("" + str.charAt(i));
        }

        totalSum = evenNumbersSum + oddNumbersSum;
        totalSum = 10 - (totalSum % 10);

        totalSum = totalSum == 10 ? 0 : totalSum;

        str = cifChar.contains(head) ? "" +
cifControlDigit.charAt(totalSum) : "" + totalSum;

        return str;
    }

    private int evenPosition(String str) {
        int aux = Integer.parseInt(str);
        aux = aux * 2;
```

```
        aux = (aux / 10) + (aux % 10);

        return aux;
    }

}
```

```
//Clase DNIValidator

public class DNIValidator implements
ConstraintValidator<DNIConstraint, String> {

    private final String dniChars = "TRWAGMYFPDXBNJZSQVHLCKE";


    @Override
    public void initialize(DNIConstraint constraintAnnotation) {
    }

    @Override
    public boolean isValid(String dni, ConstraintValidatorContext
constraintValidatorContext) {
        return dni != null && dni.length() <= 9 && validateDni(dni);
//&& dni.equals(calculateDni(dni.substring(0, 8)));
    }


    private boolean validateDni(String dni) {
        String res = "", aux = dni.substring(0, 8);
        res = !(aux.matches("[0-9]+")) ? res : aux +
dniChars.charAt(Integer.parseInt(aux) % 23);


        return dni.equals(res);

    }


}
```

La clase “ConstraintValidator” es la que proporciona los métodos que debemos implementar y la lógica interna para la validación a nuestra aplicación. Además, el método “isValid()” de tipo booleano, es sobre el que recae toda la implementación de las reglas de validación propias que deseemos.

Luego, para indicarle a la aplicación qué campos son los que necesitan ser validados, colocaremos la nueva anotación sobre los atributos de las

clases a las que pertenezcan:

```
//Clase del tipo User

@DNIConstraint
private String dni;
```

```
//Clase del tipo Company

@CIFConstraint
private String CIF;
```

A la vez, en el controlador, se debe indicar dónde se aplicará esta validación, cuando se introduzcan los datos. En nuestro caso, la validación se realizará sobre los métodos “insert” y “update”. Por lo que se le añadirá la anotación “@Valid” que indicará que ciertos campos del cuerpo de datos que introduzca el usuario, deben ser validados:

```
//UserController

@Override
public ResponseEntity<User> insertUser(@RequestBody @Valid User user)
{
    return ResponseEntity.ok(userService.insert(user));
}

@Override
public ResponseEntity<User> updateUser(@RequestBody @Valid User user)
{
    return ResponseEntity.ok(userService.update(user));
}
```

```
//CompanyController

@Override
public ResponseEntity<Company> insertCompany(@RequestBody Company
company) {
    return ResponseEntity.ok(companyService.insert(company));
}

@Override
public ResponseEntity<Company> updateCompany(@RequestBody Company
company) {
    return ResponseEntity.ok(companyService.update(company));
}
```

Con esto, nuestra aplicación ya tiene implementada una validación de campos introducidos por el usuario.

Documentación:

1. https://www.baeldung.com/spring-mvc-custom-validator
2. https://github.com/agarimo/Validador/blob/master/src/principal/CalculaNif.java
3. https://www.journaldev.com/2668/spring-validation-example-mvc-validator
4. https://www.logicbig.com/tutorials/spring-framework/spring-core/creating-custom-validation-constraints.html

Resultado (Commit): Corrección de errores en la API de User. Implementación de Spring Validator para el DNI y el CIF.

Commid ID: 4382cb0e225ddce9c73d18f9da306bade975d386

Versión: **vX.Y-NAME**

Objetivos: Objetivos a realizar en esta versión.

Proceso:

Lo que se hace

```
Código que se ha escrito/cambiado
```

Documentación: Urls de las webs donde se ha buscado información.

Resultado (Commit): Mensaje del COMMIT. Descripción de lo que finalmente se ha hecho.

Commit ID: ID del Commit resultante de esta versión.

PLANTILLA

Lista de tareas:

1. In the project there are several TODO:

~~2. Implement CRUD in CompanyApi and CompanyService~~

~~3. Implement Spring Validator for CIF and DNI (with Java annotation) in CompanyApi and UserApi~~

~~4. Implement a service that returns a list of companies grouped by city in CompanyApi~~

~~5. Implement a service that returns the companies of a given city in CompanyApi~~

~~6. Implement a service that returns all the users of a company based on their id or name in UserApi~~

~~7. Implement a service that returns all users who do not work in your city~~

2. Implement Swagger for the documentation of the APIs.

~~To do this you must add the corresponding dependency~~, configure swagger and implement it in the corresponding APIs

Optional

1. Implement a global exception handler to the application by using Spring

2. Perform unit tests through Junit 5 and Mockito of the code that you have implemented

3. Perform integration test with the embedded mongo database that is already configured