Informe individual de actividades del proyecto

Esta es una plantilla que sirve como guía para realizar este informe. Por favor, mantén las mismas secciones y los contenidos que se indican para poder hacer su revisión mucho más ágil. Este informe sólo es necesario entregarlo en la entrega final del proyecto.

Datos generales

URL del Repositorio de GitHub: https://github.com/gii-is-DP1/dp1-2020-g2-07.git
Nombre de usuario en GitHub: Juanmagc99
Rama del usuario en GitHub: juagarcri1

Participación en el proyecto

Historias de usuario en las que he participado

He implementado completas las historias de usuario H1, H2, H3, H4, H5, H6, H7, H29, H30 además, he desarrollado junto a mi compañero Miguel Molina las historias de usuario H27, H28 Y H31.

Funcionalidad implementada

-**Controladores creados:** ClienteController, EmployeeController, AdminController
-**Servicios creados:** AdminService, ClienteService, EmployeeService, EmailService
-**Entidades creadas:** Admin, Authorities, Email, Employee, EmployeeRevenue, Individual, Pago
-**Vistas creadas:** adminHome,checkUsers,newAnnouncement, newEmail, clienteDetails, clientListing, createOrUptateClientsForm, createOrUptateEmployee, payForm, salaryForm, employeeDetails, employeesListing.
-**Otras implementaciones:** Objeto Email, Enums Profession y SubType

Pruebas implementadas

Pruebas unitarias

He creado tests unitarios para 2 servicios (EmployeeService,ClienteService), 2 controladores (EmployeeController, ClienteController. Eso hace un total de 4 clases de test unitarios con un total de 43 métodos anotados con @Test.

Pruebas de Controlador

He creado 2 caso de prueba positivo y 1 negativos de controlador para la H1.
He creado 2 caso de prueba positivo y 3 negativos de controlador para la H2.
He creado 1 caso de prueba positivo y 1 negativos de controlador para la H3.
He creado 1 caso de prueba positivo y 1 negativos de controlador para la H4.
He creado 2 caso de prueba positivo y 1 negativos de controlador para la H5.
He creado 2 caso de prueba positivo y 2 negativos de controlador para la H6.
He creado 1 caso de prueba positivo y 2 negativos de controlador para la H7.
He creado 1 caso de prueba positivo y 1 negativos de controlador para la H9

He creado 2 caso de prueba positivo y 1 negativos de controlador para la H27.
He creado 2 caso de prueba positivo y 2 negativos de controlador para la H28.

Ejemplos de funcionalidades implementadas

*En esta sección debes poner y explicar **un único ejemplo de cada tipo de elemento de funcionalidad implementado por ti (un controlador, un servicio, un repositorio, y si los has implementado también, un validador y un formateador).** Esto debe incluir el código del elemento correspondiente y el nombre y ruta del fichero donde se encuentra. No obstante, poner el trozo de código únicamente no es suficiente, debe incluir una breve explicación de las responsabilidades de la clase o interfaz. Los ejemplos hay que ponerlos de los puntos que se recogen a continuación. Se deben poner preferentemente ejemplos que se hayan hecho de forma individual. Si alguno se ha hecho en colaboración con un compañero, indícalo explícitamente.*

Entidades (máximo de dos ejemplos)

Individual: src/main/java/org/springframework/samples/petclinic/model/Individual.java
La clase Individual es usada como una súper clase para cliente y empleado, para así ahorrar código, tiene relación 1:1 con User, contiene los datos típicos de una persona (nombre, apellidos,email,dirección,…).

```
@MappedSuperclass
public class Individual extends BaseEntity {

    @Column(name = "first_name")
    @NotEmpty(message = "First name must be filled")
    @Size(min = 3, max = 20)
    @Pattern(regexp = "^[A-Za-z]+((\\s)?((\\'|\\-|\\.)?([A-Za-z])+))*$")
    private String first_name;

    @Column(name = "last name")
    @NotEmpty(message = "Last name must be filled")
    @Size(min = 3, max = 20)
    @Pattern(regexp = "^[A-Za-z]+((\\s)?((\\'|\\-|\\.)?([A-Za-z])+))*$")
    private String last_name;

    @Min(value=18, message="Must be equal or greater than 18")
    private Integer age;

    @Column(name = "DOB")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate DOB;

    @Column(name = "address")
    @NotEmpty(message = "Address must be filled")
    private String address;

    @Column(name = "category")
    @Enumerated(EnumType.ORDINAL)
    private Categoria category;

    @Column(name = "IBAN")
```

```
    @NotEmpty(message = "IBAN must be filled")
    @Pattern(regexp = "([a-zA-
Z]{2})\\s*\\t*(\\d{2})\\s*\\t*(\\d{4})\\s*\\t*(\\d{4})\\s*\\t*(\\d{2})\\s*\
\t*(\\d{10})")
    private String IBAN;

    @Column(name = "email")
    @NotEmpty(message = "Email must be filled")
    @Pattern(regexp = "^[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\\.[a-z0-
9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-
9](?:[a-z0-9-]*[a-z0-9])?$")
    private  String email;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "username", referencedColumnName = "username")
    private User user;

    public String getFirst_name() {
        return first_name;
    }

    public void setFirst_name(String first_name) {
        this.first_name = first_name;
    }

    public String getLast_name() {
        return last_name;
    }

    public void setLast_name(String last_name) {
        this.last_name = last_name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String direccion) {
        this.address = direccion;
    }

    public Categoria getCategory() {
        return category;
    }

    public void setCategory(Categoria categoria) {
        this.category = categoria;
    }

    public String getIBAN() {
        return IBAN;
    }

    public void setIBAN(String IBAN) {
        this.IBAN = IBAN;
    }

    public User getUser() {
        return user;
    }
```

```
    public void setUser(User user) {
        this.user = user;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

   public Integer getAge() {
      return Period.between(DOB, LocalDate.now()).getYears();
   }

    public LocalDate getDOB() {
        return DOB;
    }

    public void setDOB(LocalDate DOB) {
        this.DOB = DOB;
    }

    public Boolean isBirthday() {
        return DOB.getMonthValue() == LocalDate.now().getMonthValue() &&
DOB.getDayOfMonth() == LocalDate.now().getDayOfMonth();
    }
}
```

EmployeeRevenue:

src/main/java/org/springframework/samples/petclinic/model/EmployeeRevenue.java

Es una clase con relación N:1 con Employee, son los salarios asociados a un empleado en específico.

```
@Entity
@Table(name = "revenue")
public class EmployeeRevenue extends BaseEntity{
    @ManyToOne
    @JoinColumn(name = "employee_id")
    private Employee employee;

    @NotNull
    @Column(name = "date_start")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate dateStart;

    @NotNull
    @Column(name = "date_end")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate dateEnd;

    @NotNull
    @Min(0)
    @Max(170)
    @Column(name = "hours_worked")
    private Integer hoursWorked;
```

```
    @NotNull
    @Min(8)
    @Column(name = "quantity")
    private Integer quantity;


    public EmployeeRevenue() {
      super();
   }

   public EmployeeRevenue(Employee employee, LocalDate dateStart, LocalDate
dateEnd,
         @Min(1) @Max(170) Integer hoursWorked, @Min(8) Integer quantity) {
      super();
      this.employee = employee;
      this.dateStart = dateStart;
      this.dateEnd = dateEnd;
      this.hoursWorked = hoursWorked;
      this.quantity = quantity;
   }

   public void setQuantity(Integer quantity) {
        this.quantity = quantity;
    }

    public Employee getEmployee() {
        return employee;
    }

    public void setEmployee(Employee employee) {
        this.employee = employee;
    }

    public LocalDate getDateStart() {
        return dateStart;
    }

    public void setDateStart(LocalDate dateStart) {
        this.dateStart = dateStart;
    }

    public LocalDate getDateEnd() {
        return dateEnd;
    }

    public void setDateEnd(LocalDate dateEnd) {
        this.dateEnd = dateEnd;
    }

    public Integer getHoursWorked() {
        return hoursWorked;
    }

    public void setHoursWorked(Integer hoursWorked) {
        this.hoursWorked = hoursWorked;
    }

    public Integer getQuantity() {
        return getHoursWorked() * getEmployee().getSalary();
    }
}
```

Servicio

EmployeeService:

src/main/java/org/springframework/samples/petclinic/web/AdminController.java

En EmployeeService se implementa la lógica relacionada con los empleados, entre los métodos implementados, están la búsqueda por username o el guardado de salarios.

```
@Slf4j
@Service
public class EmployeeService {

    public EmployeeService(EmployeeRepository employeeRepo, UserService
userService, AuthoritiesService authoritiesService){
        this.employeeRepo = employeeRepo;
        this.userService = userService;
        this.authoritiesService = authoritiesService;
    }

    private EmployeeRepository employeeRepo;

    private UserService userService;

    private AuthoritiesService authoritiesService;

    public Collection<Employee> findAll(){
        return employeeRepo.findAll();
    }

    public Optional<Employee> findById(int id){
        return employeeRepo.findById(id);
    }

    public List<Employee> findEmployeeByProfession(String profession){
        Profession prof = Profession.valueOf(profession);
        return employeeRepo.findAll().stream().filter(e ->
e.getProfession().equals(prof)).collect(Collectors.toList());
    }

    public void save(@Valid Employee employee){
        employeeRepo.save(employee);
        employee.getUser().setEnabled(true);
        userService.saveUser(employee.getUser());
        //creating authorities

authoritiesService.saveAuthorities(employee.getUser().getUsername(),
"employee");

        log.info(String.format("Employee with username %s and ID %d has
been created or updated", employee.getUser().getUsername(),
employee.getId()));

    }

    public void delete(Employee employee) {
        employeeRepo.deleteById(employee.getId());
        userService.delete(employee.getUser());
        log.info(String.format("Employee with username %s and ID %d has
```

```
been deleted", employee.getUser().getUsername(), employee.getId()));
    }

    public void addSalaryToEmployee(int id, EmployeeRevenue salary){
        Employee employee = employeeRepo.findById(id).get();
        employee.addSalary(salary);
        this.save(employee);

        log.info(String.format("A salary with ID %d has been added to
employee with username %s and ID %d", salary.getId(),
employee.getUser().getUsername(), employee.getId()));
    }

    public void addScheduletoEmployee(int id, Horario schedule) {
        Employee employee = employeeRepo.findById(id).get();
       employee.addHorario(schedule);
        this.save(employee);

        log.info(String.format("A schedule with ID %d has been added to
employee with username %s and ID %d", schedule.getId(),
employee.getUser().getUsername(), employee.getId()));
    }

    public Optional<Employee> findEmployeeByUsername(String username){
        return this.findAll().stream().filter(c ->
c.getUser().getUsername().equals(username)).findAny();
    }

    public Optional<Employee> employeeByUsername(String username){
        return this.findAll().stream().filter(e ->
e.getUser().getUsername().equals(username)).findAny();
    }

}
```

Controlador

AdminController:

src/main/java/org/springframework/samples/petclinic/web/AdminController.java
En AdminController es una de las piezas centrales, se puede destacar que gestiona las peticiones para dar de alta a los usuarios, eliminarlos y enviar emails o announcement. En general maneja la mayoría de funciones de las que solo el admin debería ser responsable.

```
@Slf4j
@Controller
@RequestMapping("/admin")
public class AdminController {
    public static final String HOME_ADMIN = "admin/adminHome";
    public static final String ADMIN_EMAIL = "admin/newEmail";
    public static final String ADMIN_ANNOUNCEMENT =
"admin/newAnnouncement";
    public static final String ADMIN_USERS = "admin/checkUsers";
    public static final String ADMINS_FORM
="admin/createOrUpdateAdminsForm";
```

```
    @Autowired
    AdminService adminService;

    @Autowired
    UserService userService;

    @Autowired
    private EmailService emailService;

    @Autowired
    ClienteService clienteService;

    @Autowired
    EmployeeService employeeService;


    @GetMapping
    public String getHomeAdmin(ModelMap model){
        model.addAttribute("petitions", userService.notEnableAdvice());
        model.addAttribute("admins", adminService.findAll());
        return HOME_ADMIN;
    }

    @GetMapping("/users")
    public String getUsers(ModelMap model){
        model.addAttribute("clientUsers",
userService.findByCategory(Categoria.CLIENTE));
        model.addAttribute("employeeUsers",
userService.findByCategory(Categoria.EMPLEADO));
        return ADMIN_USERS;
    }

    @GetMapping("/users/{username}/turn_on")
    public String activeUser(@PathVariable("username") String
username,ModelMap model){
        Optional<User> u = userService.findUser(username);

        if(u.isPresent()){
            Optional<Cliente> c =
clienteService.clientByUsername(username);
            if(c.isPresent()){
                if(c.get().getSuscripcion() == null){
                    model.addAttribute("message", "All client must have a
subscription type before enable their users");
                    return getUsers(model);
                }
            }
            if(!u.get().isEnabled()){
                u.get().setEnabled(true);
                model.addAttribute("message", "State of user " +
u.get().getUsername() + " has been changed");
                userService.saveUser(u.get());
                log.info(String.format("User with username %s is now
enabled", u.get().getUsername()));
            }else{
                model.addAttribute("message","This user is already enable,
delete it if you wont use it anymore.");
            }
        }else{
            model.addAttribute("Ups that username doesnt exist, there must
be a problem");
```

```
        }

        return getUsers(model);
    }

    @GetMapping("/users/{username}/delete")
    public String deleteUserAnsAsociated(@PathVariable("username") String
username,ModelMap model){
        Optional<User> u = userService.findUser(username);
        if(u.isPresent()){
            Optional<Cliente> c =
clienteService.clientByUsername(username);
            if (c.isPresent()){
                clienteService.delete(c.get());
                model.addAttribute("message", "Client deleted
succesfully");
            }else {
                Optional<Employee> e =
employeeService.employeeByUsername(username);
                e.ifPresent(employee -> employeeService.delete(employee));
                model.addAttribute("message", "Employee deleted
succesfully");
            }
        }else{
            model.addAttribute("message","Ups that username doesnt exist,
there must be a problem");
        }
        return getUsers(model);
    }


    @GetMapping("/newEmail")
    public String sendNewEmail(ModelMap model){
        model.addAttribute("email", new Email());
        return ADMIN_EMAIL;
    }

    @PostMapping("/newEmail")
    public String sendNewEmail(@Valid Email email, BindingResult binding,
ModelMap model){
        if (binding.hasErrors()){
            log.info(String.format("Someone tried to send an email with
errors"));
            model.addAttribute("message", "The email couldnt be send");
        }else{
            log.info(String.format("Email sent"));
            model.addAttribute("message", "Email sent succesfully");
            emailService.sendMail(email);
        }
        return getHomeAdmin(model);
    }

    @GetMapping("/newAnnouncement")
    public String sendNewAnnouncement(ModelMap model){
        model.addAttribute("email", new Email());
        return ADMIN_ANNOUNCEMENT;
    }

    @PostMapping("/newAnnouncement")
    public String sendNewAnnouncement(@Valid Email email, BindingResult
binding, ModelMap model){
```

```
        if (binding.hasErrors()){
            log.info(String.format("Someone tried to send an announcement
with errors"));
            model.addAttribute("message", "The announcement couldnt be
send");
        }else{
            /*Segun el tipo de subject que haya escogido adjuntara todos
los emails de cliente,empleado
            o ambos. Tambien parsearemos a String[] ya que es el formato
usado por la API de email*/

            List<String> client_names =
clienteService.findAll().stream().map(c ->
c.getEmail()).collect(Collectors.toList());
            List<String> employee_names =
employeeService.findAll().stream().map(e ->
e.getEmail()).collect(Collectors.toList());

            String[] arr_clients = parsetoArray(client_names);
            String[] arr_employee = parsetoArray(employee_names);

            List<String> all_names = new ArrayList<>(client_names);
            all_names.addAll(employee_names);

            String[] all_arr = parsetoArray(all_names);
            log.info(String.format("Announcement sent"));
            model.addAttribute("message", "Email sent succesfully");
            if(email.getAddress()[0].equals("ALL")){
                email.setAddress(all_arr);
            } else if(email.getAddress()[0].equals("ALL_EMPLOYEES")){
                email.setAddress(arr_employee);
            } else if(email.getAddress()[0].equals("ALL_CLIENTS")){
                email.setAddress(arr_clients);
            }

            emailService.sendMail(email);
        }
        return getHomeAdmin(model);
    }

    public String[] parsetoArray(List<String> list) {
        String[] arr = new String[list.size()];
        arr = list.toArray(arr);
        return arr;
    }
}
```

Repositorio

ClienteRepository:

src/main/java/org/springframework/samples/petclinic/repository/ClienteRepository.java
Una interfaz con las funciones predefinidas de CrudRepository, para guardar, buscar y borrar clientes.

```
public interface ClienteRepository extends CrudRepository<Cliente, Integer>
{
```

```
    Optional<Cliente> findById(Integer id);

    void deleteById(Integer id);

    Optional<Cliente> findById( int id);

    Collection<Cliente> findAll();
}
```

Conversor o Formatter (si aplica)

No aplica.

Validador y anotación asociada (si aplica, máximo de dos ejemplos)

No aplica.

Etiquetas personalizadas (si aplica, máximo de dos ejemplos)

No aplica.

Ejemplos de pruebas implementadas

*En esta sección debes poner y explicar **ejemplos de cada tipo de prueba que hayas implementado en el proyecto.** Para las unitarias se recomiendan dos ejemplos. Para las pruebas de controlador, si las hubiera, se proporcionarán como máximo dos ejemplos y a ser posible que comprueben un escenario negativo y uno positivo. Esto debe incluir el trozo de código correspondiente y el fichero donde se encuentra. No obstante, poner el trozo de código únicamente no es suficiente, debe incluir una explicación para entender el contexto y para saber qué decisiones se han tomado a la hora de decidir qué probar, y qué es lo que se ha hecho en cada una de las tres partes de cualquier prueba: Arrange, Act y Assert. La explicación no debe consistir en explicar los conceptos teóricos vistos en la asignatura. Es decir, por ejemplo, no hay que explicar qué es una prueba unitaria o qué es un fixture. La explicación ha de referirse específicamente al caso que se está tratando. Los ejemplos hay que ponerlos de los puntos que se recogen a continuación. Se deben poner preferentemente ejemplos que se hayan hecho de forma individual. Si alguno se ha hecho en colaboración con un compañero (algo normal), indícalo explícitamente.*

Pruebas unitarias (máximo de dos ejemplos)

EN LAS PRUEBAS UNITARIAS HE PUESTO EJEMPLOS (NO HE COPIADO TODA LA CLASE) PUES NO SE SI QUIERE LA CLASE ENTERA SOLO LOS TIPOS DE TEST REALIZADOS EN ELLAS

EmployeeMockTest:

src/test/java/org/springframework/samples/petclinic/service/EmployeeMockTest.java

Se ha iniciado con un setUp los datos necesarios para realizar las pruebas así como indicar como deben reaccionar nuestros mocks a las llamadas. En los dos ejemplos, comprobamos en el primero que se guarda de manera correcta el empleado y en el segundo que nos devuelve exitosamente todos los empleados

```
@Before
public void setUp(){
    employeeService = new EmployeeService(employeeRepository, userService,
authoritiesService);
    u = new User();
    e = new Employee();
    employees = new ArrayList<Employee>();

    u.setUsername("Lyle");
    u.setPassword("hola12345");
    u.setEnabled(true);
    e.setFirst_name("Lyle");
    e.setLast_name("Walt");
    e.setCategory(Categoria.EMPLEADO);
    e.setProfession(Profession.MASSAGIST);
    e.setId(1);
    e.setAddress("C/Pantomima");
    e.setEmail("jmgc101099@hotmail.com");
    e.setIBAN("ES4131905864163572187269");
    e.setUser(u);
    e.setSalaries(new ArrayList<EmployeeRevenue>());
    employees.add(e);

    revenue = new EmployeeRevenue(e, LocalDate.parse("2020-11-05",
DateTimeFormatter.ofPattern("yyyy-MM-dd")),
          LocalDate.parse("2020-11-28", DateTimeFormatter.ofPattern("yyyy-
MM-dd")), 60, 510);

    eOptional = Optional.of(e);

    when(employeeRepository.findAll()).thenReturn(employees);
    when(employeeRepository.findById(1)).thenReturn(eOptional);
    when(employeeRepository.save(e)).thenReturn(e);
}

@Test
public void shouldSave(){
    /*Comprobamos que al usar la función save del service llama a las
funciones implicadas ademas de configurar
    el mock para que actue como el repositorio*/

    employeeService.save(e);
    verify(userService).saveUser(e.getUser());

verify(authoritiesService).saveAuthorities(e.getUser().getUsername(),"emplo
yee");
}

@Test
public void shouldFindAllEmployee(){
    /*Le decimos al mock que cuando llame a finAll() del repositorio nos de
una Collection<Employee>*/

    Collection<Employee> employeesExample = this.employeeService.findAll();
    assertThat(employeesExample).hasSize(1);
assertThat(employeesExample.iterator().next().getFirst_name()).isEqualTo("L
yle");
}
```

Pruebas unitarias parametrizadas (si aplica)

No aplica

Pruebas de controlador

EmployeeControllerMockTest:

src/test/java/org/springframework/samples/petclinic/web/EmployeeControllerMockTest.java

En los test de controller vamos a estar testeando diferentes sucesos al "interactuar con la página" asi tras definir el comportamiento de los mocks e inicializar los datos necesarios probaremos que llamadas como ir al perfil de un cliente nos dirige a la página correcta. También se testea que con los parámetros y llamadas correctos podemos añadir o actualizar empleados, o que en caso de equivocarnos con dichos parámetros nos indique la presencia de errores, así mismo testetaremos las authorities pues no todos los usuarios deberían tener acceso a todas las páginas o llamadas.

```
@WebMvcTest(controllers = EmployeeController.class,
    excludeFilters = @ComponentScan.Filter(type =
FilterType.ASSIGNABLE_TYPE, classes = WebSecurityConfigurer.class),
    excludeAutoConfiguration= SecurityConfiguration.class)
public class EmployeeControllerMockTest {
    private static final int TEST_EMPLOYEE_ID = 1;
    private static final String EMPLOYEES_LISTING =
"employees/employeesListing";
    private static final String EMPLOYEES_FORM =
"employees/createOrUpdateEmployeeForm";
    private static final String EMPLOYEES_DETAILS =
"employees/employeeDetails";
    private static final String SALARY_FORM = "salary/salaryForm";

    @MockBean
    private SalaService salaService;

    @MockBean
    private HorarioService horarioService;

    @MockBean
    private UserService userService;

    @MockBean
    private EmployeeService employeeService;

    @Autowired
    private MockMvc mockMvc;

    private Employee e;
    private User u;
    private User uAux;
    private Employee eAux;
    private Collection<Employee> employees;

    private Admin admin;
    private User uAdmin;
```

```
    private Optional<User> uOptionalAdmin;
    private Optional<User> uOptional;
    private Optional<Employee> eOptional;
    private Optional<User> uOptionalAux;
    private Optional<Employee> eOptionalAux;

    private List<Horario> horarios;
    private Horario h;

    @BeforeEach
    public void setUp(){

        /*Authorities Employee*/
        Authorities a = new Authorities();
        Set<Authorities> aSet = new HashSet<Authorities>();
        a.setAuthority("employee");
        aSet.add(a);

        /*Employee*/
        u = new User("lyle","hola",true,aSet);
        e = new Employee();
        e.setFirst_name("Lyle");
        e.setLast_name("Walt");
        e.setCategory(Categoria.EMPLEADO);
        e.setProfession(Profession.MASSAGIST);
        e.setId(TEST_EMPLOYEE_ID);
        e.setDOB(LocalDate.of(2000, 1, 1));
        e.setAddress("C/Pantomima");
        e.setEmail("jmgc101099@hotmail.com");
        e.setIBAN("ES4131905864163572187269");
        e.setUser(u);
        e.setSalaries(new ArrayList<EmployeeRevenue>());
        uOptional = Optional.of(u);
        eOptional = Optional.of(e);

        /*Horarios*/
        h = new Horario(LocalDate.of(2021, 3, 14),e,new
ArrayList<Sesion>());
        horarios = new ArrayList<Horario>();
        horarios.add(h);
        e.setHorarios(horarios);

        /*Employee Aux*/
        uAux = new User("aux","hola",true,aSet);
        eAux = new Employee();
        eAux.setFirst_name("Aux");
        eAux.setLast_name("Aux");
        eAux.setCategory(Categoria.EMPLEADO);
        eAux.setProfession(Profession.MASSAGIST);
        eAux.setId(2);
        eAux.setAddress("C/Pantomima");
        eAux.setEmail("jmgc101099@hotmail.com");
        eAux.setIBAN("ES4131905864163572187269");
        eAux.setUser(uAux);
        eAux.setHorarios(new ArrayList<Horario>());
        uOptionalAux = Optional.of(uAux);
        eOptionalAux = Optional.of(eAux);

        /*Employees*/
        employees = new ArrayList<Employee>();
        employees.add(e);
```

```
        employees.add(eAux);

        /*Authorities Admin*/
        Authorities aD = new Authorities();
        Set<Authorities> aSetD = new HashSet<Authorities>();
        aD.setAuthority("admin");
        aSetD.add(aD);

        /*Admin*/
        admin = new Admin();
        uAdmin = new User("admin","12345", true, aSetD);
        admin.setId(1);
        admin.setUser(uAdmin);
        uOptionalAdmin = Optional.of(uAdmin);



given(this.horarioService.calcDays(e.getId(),"future")).willReturn(horarios
);

given(this.horarioService.calcDays(eAux.getId(),"future")).willReturn(horar
ios);


given(this.employeeService.findById(TEST_EMPLOYEE_ID)).willReturn(eOptional
);
        given(this.employeeService.findById(2)).willReturn(eOptionalAux);

        given(this.userService.findUser("lyle")).willReturn(uOptional);

given(this.userService.findUser(uAdmin.getUsername())).willReturn(uOptional
Admin);
        given(this.userService.findUser("aux")).willReturn(uOptionalAux);


given(this.employeeService.findEmployeeByUsername("lyle")).willReturn(eOpti
onal);

given(this.employeeService.findEmployeeByUsername("aux")).willReturn(eOptio
nalAux);
    }

    @WithMockUser(value = "lyle")
    @Test
    public void showEmployeeDetailsSuccess() throws Exception {
        mockMvc.perform(get("/employees/{employeeId}", TEST_EMPLOYEE_ID))
            .andExpect(model().attributeExists("employee"))
            .andExpect(model().attributeExists("horarios"))
            .andExpect(status().isOk())
            .andExpect(view().name(EMPLOYEES_DETAILS));
    }

    @WithMockUser(value = "lyle")
    @Test
    public void showEmployeeDetailsWithoutPermission() throws Exception {
        mockMvc.perform(get("/employees/{employeeId}", 2))
            .andExpect(status().is3xxRedirection())
            .andExpect(view().name("redirect:/employees/1"));
    }

    @WithMockUser(value = "admin")
```

```
    @Test
    public void getNewEmployeetFormSucces() throws Exception{
        mockMvc.perform(get("/employees/new"))
            .andExpect(status().isOk())
            .andExpect(model().attributeExists("employee"))
            .andExpect(view().name(EMPLOYEES_FORM));
    }

    @WithMockUser(value = "admin")
    @Test
    public void postNewEmployeeFormSuccess() throws Exception{
        mockMvc.perform(post("/employees/new")
            .param("first_name", "Celes").param("last_name", "Walt")
            .with(csrf())
            .param("id","3")
            .param("address", "C/From the hood ma boy")
            .param("IBAN", "ES2620952473193739231755")
            .param("email", "pikachu_1@gmail.com")
            .param("DOB","2000-01-01")
            .param("profession","LIFE_GUARD")
            .param("user.username", "hola")
            .param("user.password", "12345"))
            .andExpect(status().isOk())
            .andExpect(view().name(EMPLOYEES_LISTING));
    }

    @WithMockUser(value = "admin")
    @Test
    public void postNewEmployeeFormFail() throws Exception{
        mockMvc.perform(post("/employees/new")
            .param("first_name", "Celes1").param("last_name", "Walt")
            .with(csrf())
            .param("id","3")
            .param("address", "C/From the hood ma boy")
            .param("IBAN", "XXXXX_XXXXX")
            .param("email", "pikachu_1@gmail.com")
            .param("age","21")
            .param("profession","LIFE_GUARD")
            .param("user.username", "hola")
            .param("user.password", "12345"))
            .andExpect(model().attributeHasErrors("employee"))

.andExpect(model().attributeHasFieldErrors("employee","first_name"))
            .andExpect(model().attributeHasFieldErrors("employee","IBAN"))
            .andExpect(status().isOk())
            .andExpect(view().name(EMPLOYEES_FORM));
    }

    @WithMockUser(value = "lyle")
    @Test
    public void getUpdateClientFormSucces() throws Exception{
        mockMvc.perform(get("/employees/{employeeId}/edit",
TEST_EMPLOYEE_ID))
            .andExpect(model().attributeExists("employee"))
            .andExpect(status().isOk())
            .andExpect(view().name(EMPLOYEES_FORM));
    }

    @WithMockUser(value = "lyle")
    @Test
    public void getUpdateClientFormNoAccess() throws Exception{
```

```
        mockMvc.perform(get("/employees/{employeeId}/edit", 2))
            .andExpect(view().name("redirect:/employees/1"));
    }

    @WithMockUser(value = "admin")
    @Test
    public void postUpdateEmployeeSucces() throws Exception{
        mockMvc.perform(post("/employees/{employeeId}/edit",
TEST_EMPLOYEE_ID)
            .param("first_name", "Celes").param("last_name", "Walt")
            .with(csrf())
            .param("address", "C/From the hood ma boy")
            .param("IBAN", "ES2620952473193739231755")
            .param("suscripcion","AFTERNOON")
            .param("email", "pikachu_1@gmail.com")
            .param("user.username", "hola")
            .param("user.password", "12345"))
            .andExpect(status().is3xxRedirection())
            .andExpect(view().name("redirect:/employees/1"));
    }

    @WithMockUser(value = "admin")
    @Test
    public void postUpdateEmployeeFail() throws Exception{
        mockMvc.perform(post("/employees/{employeeId}/edit",
TEST_EMPLOYEE_ID)
            .param("first_name", "Celes").param("last_name", "Walt1")
            .with(csrf())
            .param("address", "C/From the hood ma boy")
            .param("IBAN", "ES2620952473193739231755")
            .param("suscripcion","AFTERNOON")
            .param("email", "__ --- __")
            .param("user.username", "hola")
            .param("user.password", "12345"))
            .andExpect(model().attributeHasErrors("employee"))

.andExpect(model().attributeHasFieldErrors("employee","last_name"))
            .andExpect(model().attributeHasFieldErrors("employee","email"))
            .andExpect(status().isOk())
            .andExpect(view().name(EMPLOYEES_FORM));
    }

    @WithMockUser(value = "admin")
    @Test
    public void getDeleteEmployeeSucces() throws Exception{
        mockMvc.perform(get("/employees/{employeeId}/delete", 2))
            .andExpect(model().attribute("message","Employee deleted
successfully!!"))
            .andExpect(status().isOk())
            .andExpect(view().name(EMPLOYEES_LISTING));
    }

    @WithMockUser(value = "admin")
    @Test
    public void getDeleteEmployeeFailHaveWork() throws Exception{
        mockMvc.perform(get("/employees/{employeeId}/delete",
TEST_EMPLOYEE_ID))
            .andExpect(model().attribute("message","The employee can´t be
deleted if they have work left to do"))
            .andExpect(status().isOk())
            .andExpect(view().name(EMPLOYEES_LISTING));
```

```
    }

    @WithMockUser(value = "admin")
    @Test
    public void getDeleteEmployeeFailNotExist() throws Exception{
        mockMvc.perform(get("/employees/{employeeId}/delete", 3))
            .andExpect(model().attribute("message","Cant find the employee
you are looking for"))
            .andExpect(status().isOk())
            .andExpect(view().name(EMPLOYEES_LISTING));
    }

    @WithMockUser(value = "admin")
    @Test
    public void getNewSalaryFormSucces() throws Exception{
        mockMvc.perform(get("/employees/{employeeId}/newSalary",
TEST_EMPLOYEE_ID))
            .andExpect(model().attributeExists("revenue"))
            .andExpect(model().attributeExists("employee"))
            .andExpect(status().isOk())
            .andExpect(view().name(SALARY_FORM));
    }

    @WithMockUser(value = "admin")
    @Test
    public void postNewPayFormSucces() throws Exception{
        mockMvc.perform(post("/employees/{employeeId}/newSalary",
TEST_EMPLOYEE_ID)
            .param("dateStart","2021-02-01")
            .param("dateEnd", "2021-02-28")
            .with(csrf()))
            .andExpect(status().is3xxRedirection())
            .andExpect(view().name("redirect:/employees/1"));
    }

    @WithMockUser(value = "admin")
    @Test
    public void postNewPayFormFailDifferentMonth() throws Exception{
        mockMvc.perform(post("/employees/{employeeId}/newSalary",
TEST_EMPLOYEE_ID)
            .param("dateStart","2021-02-01")
            .param("dateEnd", "2021-03-28")
            .with(csrf()))
            .andExpect(model().attribute("message","Revenues must have a
length of only 1 month"))
            .andExpect(status().isOk())
            .andExpect(view().name(SALARY_FORM));
    }

    @WithMockUser(value = "admin")
    @Test
    public void postNewPayFormFailFirstDateOlder() throws Exception{
        mockMvc.perform(post("/employees/{employeeId}/newSalary",
TEST_EMPLOYEE_ID)
            .param("dateStart","2021-02-25")
            .param("dateEnd", "2021-02-01")
            .with(csrf()))
            .andExpect(model().attribute("message","Start date must be
before end date"))
            .andExpect(status().isOk())
            .andExpect(view().name(SALARY_FORM));
```

```
        }
}
```

Principales problemas encontrados

El principal problema fue a la hora de realizar los test de los Service, pues desde el test no detectaba la configuración de la aplicación(application.properties) y por lo tanto no era capaz de usar el @Autowire para crear JavaMailSender, haciendo que el proyecto colapsase al intentar testearlo, la solución acabo siendo la inclusión una anotación.

Por otra parte al principio, la falta de experiencia a la hora de trabajar en grupo nos hacía estar descoordinados provocando que el repositorio se volviese en ocasiones caóticos y sobrescribiendo código que ya había sido arreglado, añadiendo de nuevo clases borradas previamente, etc.

Otros comentarios

Añade en esta sección cualquier otra tarea que hayas llevado a cabo en el proyecto y que no haya quedado claramente recogida en los apartados anteriores. También puedes mencionar, si procede, ***tu participación en los A+*** *que haya elaborado el grupo.*