Nombre: Mineral House Spa Grupo: G2-07

Informe individual de actividades del proyecto

Datos generales

URL del Repositorio de GitHub: https://github.com/gii-is-DP1/dp1-2020-g2-07
Nombre de usuario en GitHub: fernando-hidalgo
Rama del usuario en Github: miscoX

Participación en el proyecto

Historias de usuario en las que he participado

Historias implementadas en solitario: H22, H23, H24, H25, H32, H33, H34
Historias implementadas con el compañero: No aplica

Funcionalidad implementada

Controladores creados: BalanceController, BonoController
Servicios creados: BalanceService, BonoService
He añadido métodos a los siguientes Servicios no creados por mí: HorarioService
Entidades creadas: Bono, Balance
Vistas creadas: BalanceListing, BalanceEmplEdit, BalancesListing, BonosListing, createToken, RedeemToken

Esto hace un total de 6 clases implementadas por mí y 6 interfaces definidos.

Pruebas implementadas

Pruebas unitarias

Test de los servicios: BonoService, BalanceService
Test de los controladores: BonoController, BalanceController

Eso hace un total de 4 clases de prueba unitarias con un total de 26 métodos anotados con @Test.

Pruebas de Controlador

Controlador BonoController:

- Historia de Usuario H23 – Mostrar Bonos
  - 1 caso positivos
- Historia de Usuario H24 – Consumir Bono
  - 1 caso positivos, 3 negativos
- Historia de Usuario H25 – Eliminar Bono
  - 1 caso positivo, 1 caso negativo

Controlador BalanceController

- Historia de Usuario H32 – Mostrar balance con estadísticas
  - 2 caso positivo, 1 casos negativos
- Historia de Usuario H34 – Empleados cualificados para ver balances
  - 2 caso positivo, 3 casos negativos

Pruebas de Service

- Service BalanceService (7 casos positivos)
- Service BonoService (2 casos positivos, 1 negativo)

Nombre: Mineral House Spa Grupo: G2-07

Ejemplos de funcionalidades implementadas

Entidades (máximo de dos ejemplos)

Bono (/dp1-2020-g2-07/src/main/java/org/springframework/samples/petclinic/model/Bono.java) Esta clase cumple la función de ser unos códigos que los clientes reciben y les permite acceder al spa en horas fuera a las que su suscripción les permite

```java
@Entity
@Table(name = "bonos")
public class Bono extends  BaseEntity {

    @Size(max = 12, message="Code must be 12 characters long or less")
    @Column(name = "codigo")
    private String codigo;

    @NotNull(message = "Price must be filled")
    @Column(name = "precio")
    private Integer precio;

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

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

    @Size(min = 3, max = 100, message="Description must be beetwen 3 and 100 characters")
    @Column(name = "descripcion")
    private String descripcion;

    @Column(name = "usado")
    private Boolean usado;
```

Aquí se muestran los atributos de la clase. A destacar, el código puede introducirse a mano, o generarse automáticamente gracias el código a continuación

```java
  private String getAlphaNumericString(int n) {
      String AlphaNumericString = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                                  + "0123456789"
                                  + "abcdefghijklmnopqrstuvxyz";

      StringBuilder sb = new StringBuilder(n);

      for (int i = 0; i < n; i++) {
          int index  = (int)(AlphaNumericString.length() * Math.random());

          sb.append(AlphaNumericString.charAt(index));
      }

      return sb.toString();
  }
```

Balance (/dp1-2020-g2-07/src/main/java/org/springframework/samples/petclinic/model/Balance.java) Esta clase permite conocer los gastos e ingresos del spa, desglosado por categorías y contando con una gráfica para cada uno de ellos

```java
@Entity
@Table(name = "balances")
public class Balance extends BaseEntity{

     @JoinTable(
                name = "rel_statement_employee",
                joinColumns = @JoinColumn(name="FK_Statement"),
                inverseJoinColumns = @JoinColumn(name="FK_Employee")
            )

    @Column(name = "month")
    private String month;

    @Column(name = "year")
    private String year;

    @Column(name = "subs")
    private Integer subs;

    @Column(name = "bonos")
    private Integer bonos;

    @Column(name = "salaries")
    private Integer salaries;

    @ManyToMany(cascade = CascadeType.ALL)
    private List<Employee> employee;
```

Nombre: Mineral House Spa Grupo: G2-07

Servicio

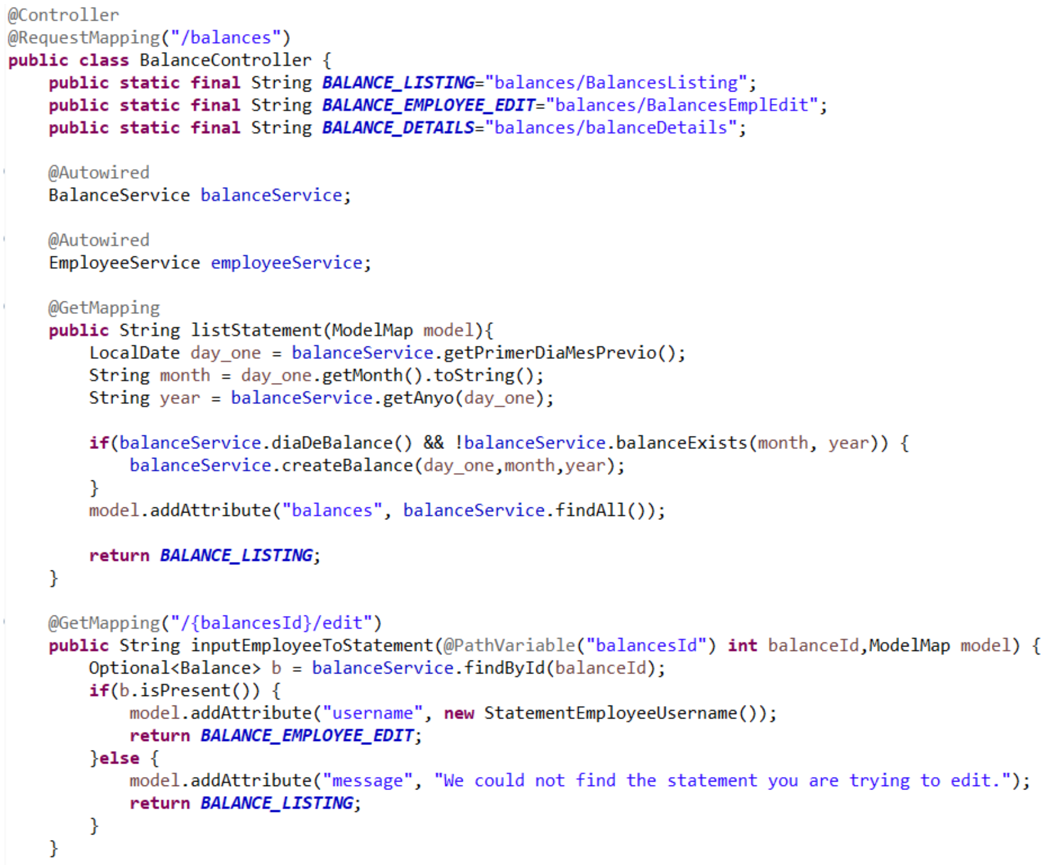
Balance Service (/dp1-2020-g2-07/src/main/java/org/springframework/samples/petclinic/service/BalanceService.java) En este servicio se encuentran métodos de cálculo automático de la fecha actual, el primer y último día del mes pasado, la obtención de los datos de cada una de las categorías que conforman el balance, así como la creación del objeto GSON para mostrar las gráficas de CanvasJS

```java
    public String getAnyo(LocalDate fecha) {
        Integer anyo = fecha.getYear();
        String anyo_text = anyo.toString();
        return anyo_text;
    }


    //Comprueba si hoy es día 1, para calcular el balance del mes pasado
    public boolean diaDeBalance() {
        Boolean tocaBalance = false;
        Integer day_today = LocalDate.now().getDayOfMonth();
        if(day_today.equals(17)) {
            tocaBalance = true;
        }
        return tocaBalance;
       }

    //Devuelve el día 1 del mes previo (Inicio del rango)
    public LocalDate getPrimerDiaMesPrevio() {
        Date previo = Date.from(ZonedDateTime.now().minusMonths(1).withDayOfMonth(1).toInstant());
        return previo.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
    }


    //Devuelve el último día del mes (Fin del rango)
    public LocalDate getUltimoDiaMes(LocalDate selec) {
        LocalDate ultimo = selec.withDayOfMonth(selec.getMonth().length(selec.isLeapYear()));
        return ultimo;
    }

    public Integer getSubs(LocalDate init, LocalDate last) {
        Collection<Pago> total = balanceRepo.findSubsByMonth(init, last);
        Iterator<Pago> iterator = total.iterator();
        int res = 0;
        while (iterator.hasNext()) {
            res = res + iterator.next().getCantidad();
        }
        return res;
    }
public Integer getSalaries(LocalDate init, LocalDate last) {
    Collection<EmployeeRevenue> total = balanceRepo.findSalariesByMonth(init, last);
    Iterator<EmployeeRevenue> iterator = total.iterator();
    Integer res = 0;
    while (iterator.hasNext()) {
        res = res + iterator.next().getQuantity();
    }
    return res;
}

public Boolean balanceExists (String month, String year) {
    Integer x = balanceRepo.findBalanceExists(month, year);
    Boolean res = false;
    if(x==1) {
        res=true;
    }
    return res;
}

public Integer getTokens (LocalDate date_start, LocalDate date_end, Boolean used) {
    Collection<Bono> total = balanceRepo.findUsedTokensByMonth(date_start,date_end, used);
    Iterator<Bono> iterator = total.iterator();
    Integer res = 0;
    while (iterator.hasNext()) {
        res = res + iterator.next().getPrecio();
    }
    return res;
}

public Collection<Bono> getTokensData (LocalDate date_start, LocalDate date_end) {
    return balanceRepo.findUsedTokensByMonth(date_start,date_end, true);
}

public Collection<Pago> getSubsData (LocalDate date_start, LocalDate date_end) {
    return balanceRepo.findSubsByMonth(date_start, date_end);
}

public Collection<EmployeeRevenue> getSalariesData (LocalDate date_start, LocalDate date_end) {
    return balanceRepo.findSalariesByMonth(date_start, date_end);
}

public void createBalance(LocalDate day_one, String month, String year) {
    LocalDate day_last = getUltimoDiaMes(day_one);
    List<Employee> l_e = new ArrayList<Employee>();

    Integer subs = getSubs(day_one, day_last);
    Integer tokens = getTokens(day_one, day_last, true);
    Integer salaries = getSalaries(day_one, day_last);
    Balance b = new Balance(month, year, subs, tokens, salaries, l_e);
    save(b);
}

public String createStats(Balance b) {
    Gson gsonObj = new Gson();
    Map<Object,Object> map = null;
    List<Map<Object,Object>> list = new ArrayList<Map<Object,Object>>();

    map = new HashMap<Object,Object>(); map.put("label", "Subs"); map.put("y", b.getSubs()); list.add(map);
    map = new HashMap<Object,Object>(); map.put("label", "Tokens"); map.put("y", b.getBonos()); list.add(map);
    map = new HashMap<Object,Object>(); map.put("label", "Salaries"); map.put("y", -b.getSalaries()); list.add(map);
    map = new HashMap<Object,Object>(); map.put("label", "Gross Income"); map.put("isIntermediateSum", true); map.put("color", "#55646e"); list.add(map);

    String dataPoints = gsonObj.toJson(list);
    return dataPoints;
}
```

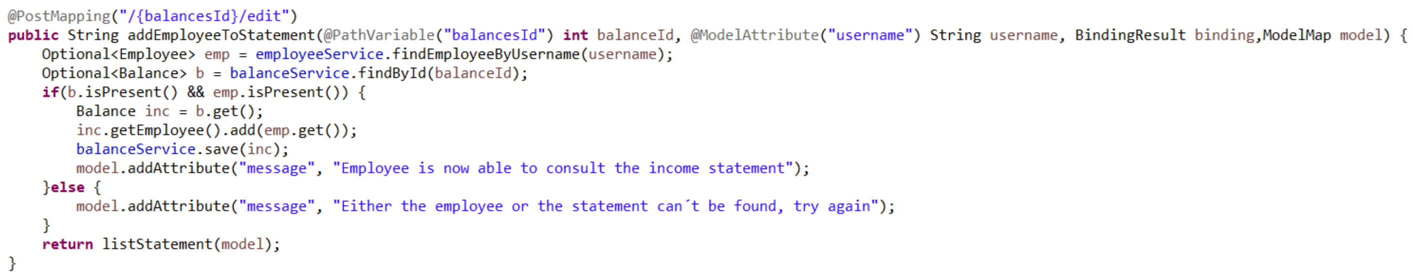
Nombre: Mineral House Spa Grupo: G2-07

Controlador

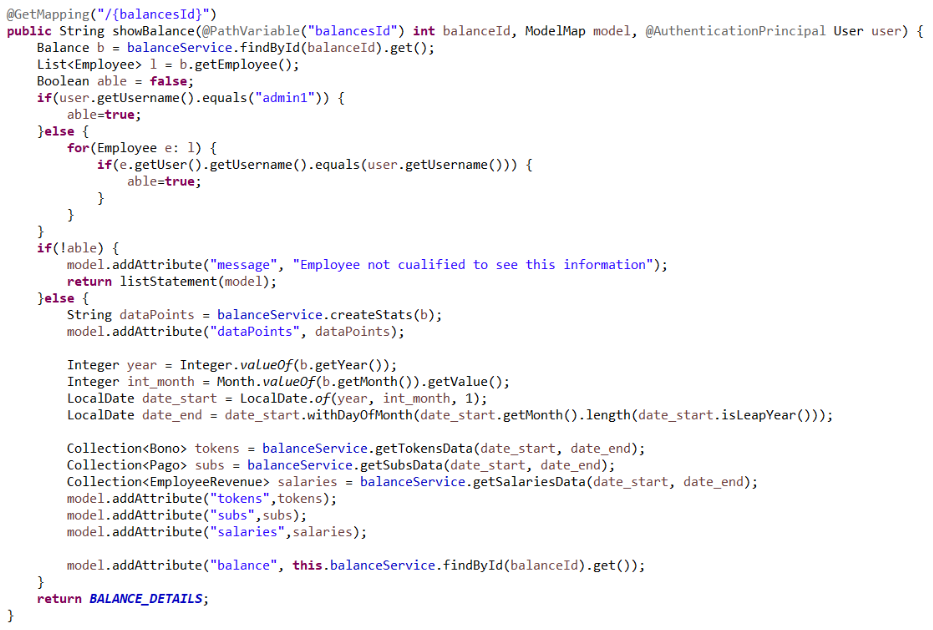
Balance Controller (/dp1-2020-g2-07/src/main/java/org/springframework/samples/petclinic/web/BalanceController.java) Este controlador nos lleva a las vistas de añadir empleados a la lista de empleados de confianza para ver el balance, así como mostrar cada uno de los balances



```java
@Controller
@RequestMapping("/balances")
public class BalanceController {
    public static final String BALANCE_LISTING="balances/BalancesListing";
    public static final String BALANCE_EMPLOYEE_EDIT="balances/BalancesEmplEdit";
    public static final String BALANCE_DETAILS="balances/balanceDetails";

    @Autowired
    BalanceService balanceService;

    @Autowired
    EmployeeService employeeService;

    @GetMapping
    public String listStatement(ModelMap model){
        LocalDate day_one = balanceService.getPrimerDiaMesPrevio();
        String month = day_one.getMonth().toString();
        String year = balanceService.getAnyo(day_one);

        if(balanceService.diaDeBalance() && !balanceService.balanceExists(month, year)) {
            balanceService.createBalance(day_one,month,year);
        }
        model.addAttribute("balances", balanceService.findAll());

        return BALANCE_LISTING;
    }

    @GetMapping("/{balancesId}/edit")
    public String inputEmployeeToStatement(@PathVariable("balancesId") int balanceId,ModelMap model) {
        Optional<Balance> b = balanceService.findById(balanceId);
        if(b.isPresent()) {
            model.addAttribute("username", new StatementEmployeeUsername());
            return BALANCE_EMPLOYEE_EDIT;
        }else {
            model.addAttribute("message", "We could not find the statement you are trying to edit.");
            return BALANCE_LISTING;
        }
    }
```



```java
@PostMapping("/{balancesId}/edit")
public String addEmployeeToStatement(@PathVariable("balancesId") int balanceId, @ModelAttribute("username") String username, BindingResult binding,ModelMap model) {
    Optional<Employee> emp = employeeService.findEmployeeByUsername(username);
    Optional<Balance> b = balanceService.findById(balanceId);
    if(b.isPresent() && emp.isPresent()) {
        Balance inc = b.get();
        inc.getEmployee().add(emp.get());
        balanceService.save(inc);
        model.addAttribute("message", "Employee is now able to consult the income statement");
    }else {
        model.addAttribute("message", "Either the employee or the statement can´t be found, try again");
    }
    return listStatement(model);
}
```

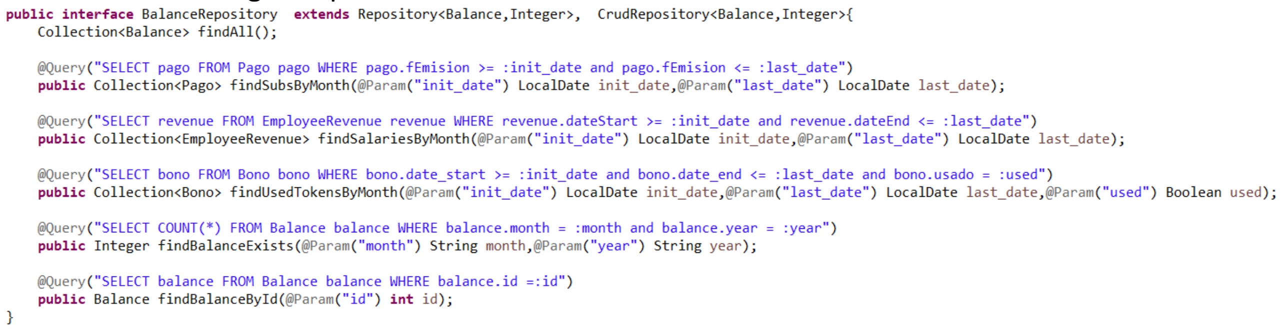


```java
@GetMapping("/{balancesId}")
public String showBalance(@PathVariable("balancesId") int balanceId, ModelMap model, @AuthenticationPrincipal User user) {
    Balance b = balanceService.findById(balanceId).get();
    List<Employee> l = b.getEmployee();
    Boolean able = false;
    if(user.getUsername().equals("admin1")) {
        able=true;
    }else {
        for(Employee e: l) {
            if(e.getUser().getUsername().equals(user.getUsername())) {
                able=true;
            }
        }
    }
    if(!able) {
        model.addAttribute("message", "Employee not cualified to see this information");
        return listStatement(model);
    }else {
        String dataPoints = balanceService.createStats(b);
        model.addAttribute("dataPoints", dataPoints);

        Integer year = Integer.valueOf(b.getYear());
        Integer int_month = Month.valueOf(b.getMonth()).getValue();
        LocalDate date_start = LocalDate.of(year, int_month, 1);
        LocalDate date_end = date_start.withDayOfMonth(date_start.getMonth().length(date_start.isLeapYear()));

        Collection<Bono> tokens = balanceService.getTokensData(date_start, date_end);
        Collection<Pago> subs = balanceService.getSubsData(date_start, date_end);
        Collection<EmployeeRevenue> salaries = balanceService.getSalariesData(date_start, date_end);
        model.addAttribute("tokens",tokens);
        model.addAttribute("subs",subs);
        model.addAttribute("salaries",salaries);

        model.addAttribute("balance", this.balanceService.findById(balanceId).get());
    }
    return BALANCE_DETAILS;
}
```

Nombre: Mineral House Spa   Grupo: G2-07

Repositorio

Balance Repository (/dp1-2020-g2-07/src/main/java/org/springframework/samples/petclinic/repository/BalanceRepository.java) Se crean diferentes query, uno para encontrar el balance por ID, saber si el balance ya existe, y varias para obtener datos de cada categoría que conforma un balance



```
public interface BalanceRepository  extends Repository<Balance,Integer>,  CrudRepository<Balance,Integer>{
    Collection<Balance> findAll();

    @Query("SELECT pago FROM Pago pago WHERE pago.fEmision >= :init_date and pago.fEmision <= :last_date")
    public Collection<Pago> findSubsByMonth(@Param("init_date") LocalDate init_date,@Param("last_date") LocalDate last_date);

    @Query("SELECT revenue FROM EmployeeRevenue revenue WHERE revenue.dateStart >= :init_date and revenue.dateEnd <= :last_date")
    public Collection<EmployeeRevenue> findSalariesByMonth(@Param("init_date") LocalDate init_date,@Param("last_date") LocalDate last_date);

    @Query("SELECT bono FROM Bono bono WHERE bono.date_start >= :init_date and bono.date_end <= :last_date and bono.usado = :used")
    public Collection<Bono> findUsedTokensByMonth(@Param("init_date") LocalDate init_date,@Param("last_date") LocalDate last_date,@Param("used") Boolean used);

    @Query("SELECT COUNT(*) FROM Balance balance WHERE balance.month = :month and balance.year = :year")
    public Integer findBalanceExists(@Param("month") String month,@Param("year") String year);

    @Query("SELECT balance FROM Balance balance WHERE balance.id =:id")
    public Balance findBalanceById(@Param("id") int id);
}
```

Conversor o Formatter (si aplica)

No aplica

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

No aplica

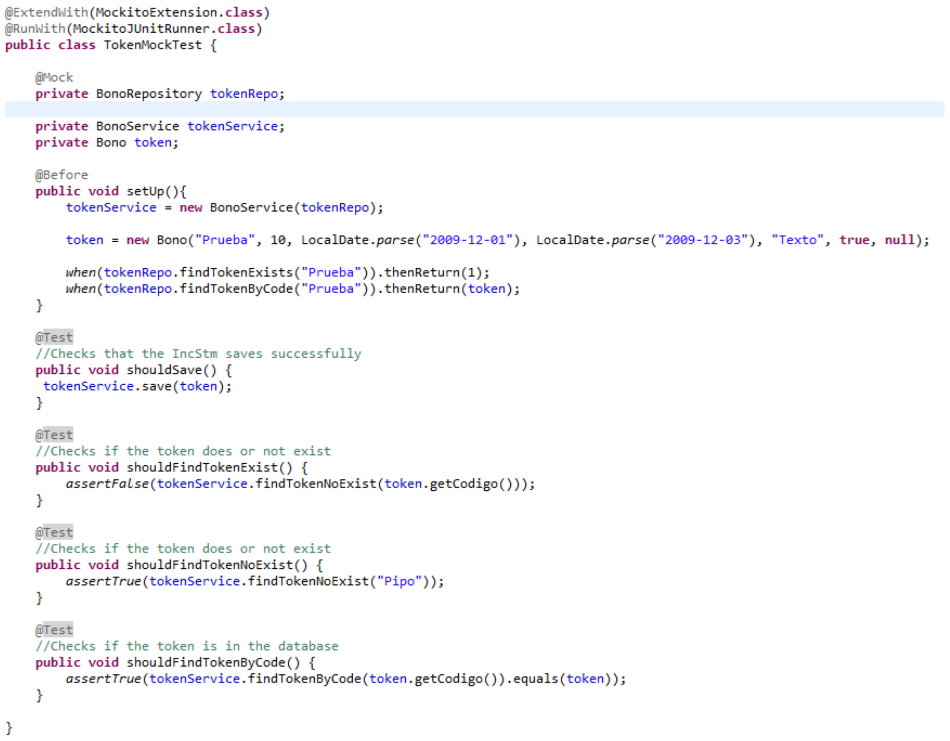
Etiquetas personalizadas (si aplica, máximo de dos ejemplos)

No aplica

Ejemplos de pruebas implementadas

Pruebas unitarias (máximo de dos ejemplos)

TokenMockTest (/dp1-2020-g2-07/src/test/java/org/springframework/samples/petclinic/service/TokenMockTest.java) En el arrange creamos el token a probar, en el act probamos el service en cuestión y comprobamos mediante assert si devuelve lo pedido. Por ejemplo, en el segundo método marcado con @Test, le damos un código que existe, por tanto, el assert es false



```
@ExtendWith(MockitoExtension.class)
@RunWith(MockitoJUnitRunner.class)
public class TokenMockTest {

    @Mock
    private BonoRepository tokenRepo;

    private BonoService tokenService;
    private Bono token;

    @Before
    public void setUp(){
        tokenService = new BonoService(tokenRepo);

        token = new Bono("Prueba", 10, LocalDate.parse("2009-12-01"), LocalDate.parse("2009-12-03"), "Texto", true, null);

        when(tokenRepo.findTokenExists("Prueba")).thenReturn(1);
        when(tokenRepo.findTokenByCode("Prueba")).thenReturn(token);
    }

    @Test
    //Checks that the IncStm saves successfully
    public void shouldSave() {
     tokenService.save(token);
    }

    @Test
    //Checks if the token does or not exist
    public void shouldFindTokenExist() {
        assertFalse(tokenService.findTokenNoExist(token.getCodigo()));
    }

    @Test
    //Checks if the token does or not exist
    public void shouldFindTokenNoExist() {
        assertTrue(tokenService.findTokenNoExist("Pipo"));
    }

    @Test
    //Checks if the token is in the database
    public void shouldFindTokenByCode() {
        assertTrue(tokenService.findTokenByCode(token.getCodigo()).equals(token));
    }

}
```

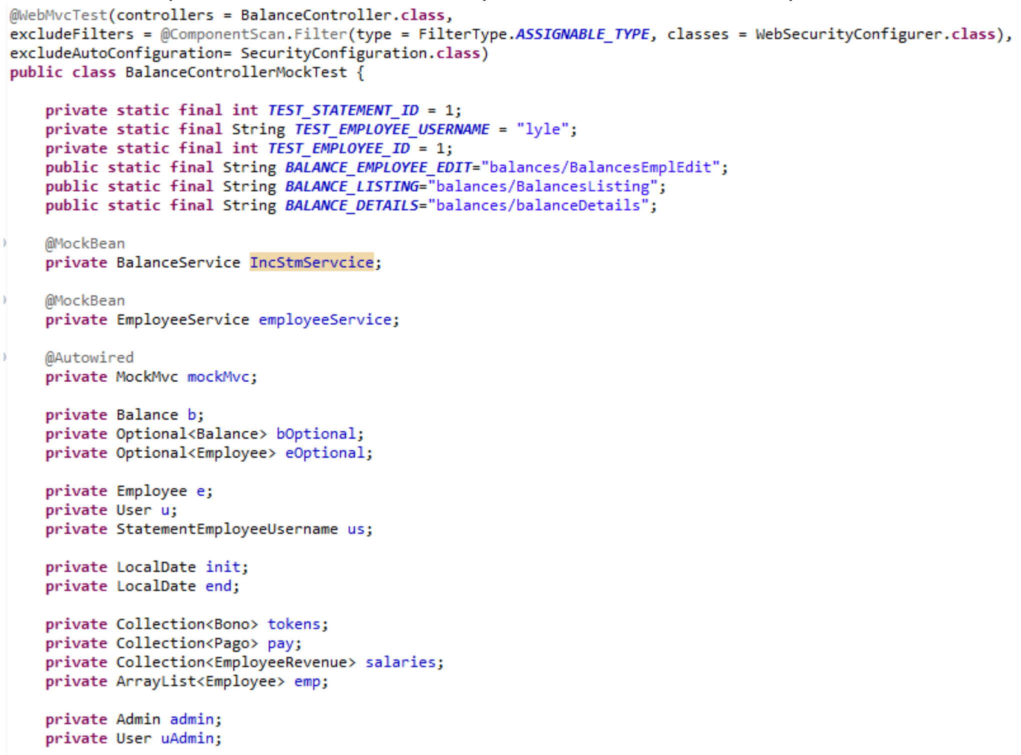
Nombre: Mineral House Spa Grupo: G2-07

Pruebas unitarias parametrizadas (si aplica)

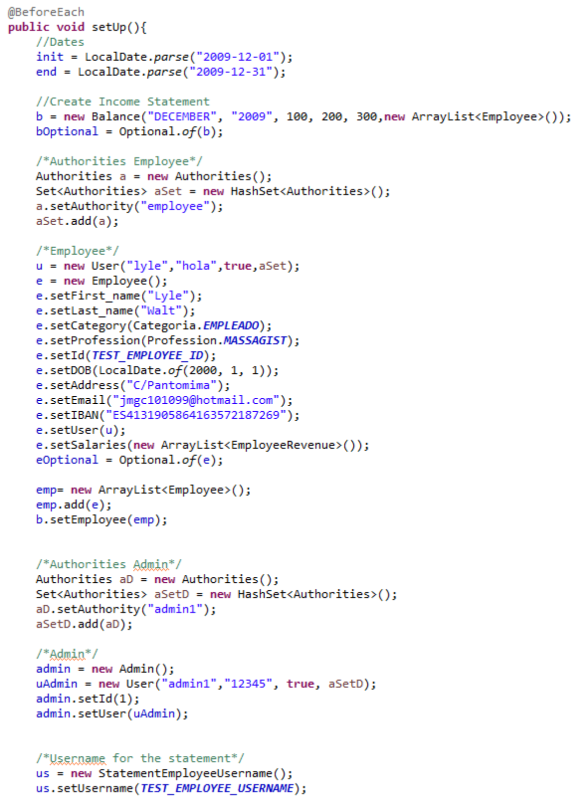
No aplica

Pruebas de Controller

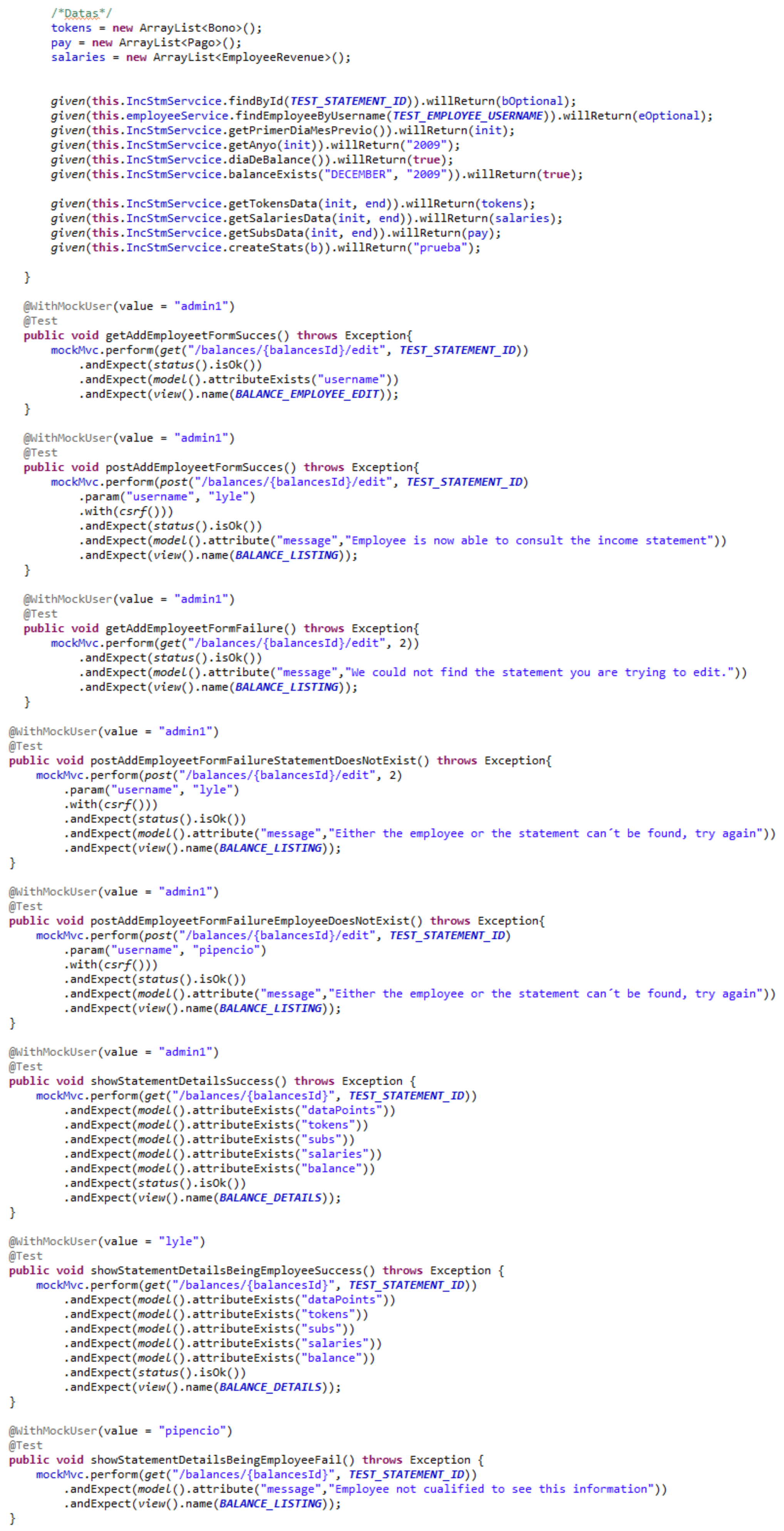
<u>BalanceControllerMockTest</u>(/dp1-2020-g2 07/src/test/java/org/springframework/samples/petclinic/service/BalanceControllerMockTest.java)
En el arrange creamos todos los objetos necesarios para llevar a cabo el testeo, en este caso el balance, empleado, usuario fechas, admin y diversas colecciones. En los métodos vamos comprobando si el balance existe, el empleado existe o si un empleado está cualificado para consultar el balance en cuestión



```java
@WebMvcTest(controllers = BalanceController.class,
excludeFilters = @ComponentScan.Filter(type = FilterType.ASSIGNABLE_TYPE, classes = WebSecurityConfigurer.class),
excludeAutoConfiguration= SecurityConfiguration.class)
public class BalanceControllerMockTest {

    private static final int TEST_STATEMENT_ID = 1;
    private static final String TEST_EMPLOYEE_USERNAME = "lyle";
    private static final int TEST_EMPLOYEE_ID = 1;
    public static final String BALANCE_EMPLOYEE_EDIT="balances/BalancesEmplEdit";
    public static final String BALANCE_LISTING="balances/BalancesListing";
    public static final String BALANCE_DETAILS="balances/balanceDetails";

    @MockBean
    private BalanceService IncStmServcice;

    @MockBean
    private EmployeeService employeeService;

    @Autowired
    private MockMvc mockMvc;

    private Balance b;
    private Optional<Balance> bOptional;
    private Optional<Employee> eOptional;

    private Employee e;
    private User u;
    private StatementEmployeeUsername us;

    private LocalDate init;
    private LocalDate end;

    private Collection<Bono> tokens;
    private Collection<Pago> pay;
    private Collection<EmployeeRevenue> salaries;
    private ArrayList<Employee> emp;

    private Admin admin;
    private User uAdmin;
```



```java
@BeforeEach
public void setUp(){
    //Dates
    init = LocalDate.parse("2009-12-01");
    end = LocalDate.parse("2009-12-31");

    //Create Income Statement
    b = new Balance("DECEMBER", "2009", 100, 200, 300,new ArrayList<Employee>());
    bOptional = Optional.of(b);

    /*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>());
    eOptional = Optional.of(e);

    emp= new ArrayList<Employee>();
    emp.add(e);
    b.setEmployee(emp);


    /*Authorities Admin*/
    Authorities aD = new Authorities();
    Set<Authorities> aSetD = new HashSet<Authorities>();
    aD.setAuthority("admin1");
    aSetD.add(aD);

    /*Admin*/
    admin = new Admin();
    uAdmin = new User("admin1","12345", true, aSetD);
    admin.setId(1);
    admin.setUser(uAdmin);


    /*Username for the statement*/
    us = new StatementEmployeeUsername();
    us.setUsername(TEST_EMPLOYEE_USERNAME);
```



```java
        /*Datas*/
        tokens = new ArrayList<Bono>();
        pay = new ArrayList<Pago>();
        salaries = new ArrayList<EmployeeRevenue>();


        given(this.IncStmServcice.findById(TEST_STATEMENT_ID)).willReturn(bOptional);
        given(this.employeeService.findEmployeeByUsername(TEST_EMPLOYEE_USERNAME)).willReturn(eOptional);
        given(this.IncStmServcice.getPrimerDiaMesPrevio()).willReturn(init);
        given(this.IncStmServcice.getAnyo(init)).willReturn("2009");
        given(this.IncStmServcice.diaDeBalance()).willReturn(true);
        given(this.IncStmServcice.balanceExists("DECEMBER", "2009")).willReturn(true);

        given(this.IncStmServcice.getTokensData(init, end)).willReturn(tokens);
        given(this.IncStmServcice.getSalariesData(init, end)).willReturn(salaries);
        given(this.IncStmServcice.getSubsData(init, end)).willReturn(pay);
        given(this.IncStmServcice.createStats(b)).willReturn("prueba");

    }

    @WithMockUser(value = "admin1")
    @Test
    public void getAddEmployeetFormSucces() throws Exception{
        mockMvc.perform(get("/balances/{balancesId}/edit", TEST_STATEMENT_ID))
            .andExpect(status().isOk())
            .andExpect(model().attributeExists("username"))
            .andExpect(view().name(BALANCE_EMPLOYEE_EDIT));
    }

    @WithMockUser(value = "admin1")
    @Test
    public void postAddEmployeetFormSucces() throws Exception{
        mockMvc.perform(post("/balances/{balancesId}/edit", TEST_STATEMENT_ID)
            .param("username", "lyle")
            .with(csrf()))
            .andExpect(status().isOk())
            .andExpect(model().attribute("message","Employee is now able to consult the income statement"))
            .andExpect(view().name(BALANCE_LISTING));
    }

    @WithMockUser(value = "admin1")
    @Test
    public void getAddEmployeetFormFailure() throws Exception{
        mockMvc.perform(get("/balances/{balancesId}/edit", 2))
            .andExpect(status().isOk())
            .andExpect(model().attribute("message","We could not find the statement you are trying to edit."))
            .andExpect(view().name(BALANCE_LISTING));
    }

@WithMockUser(value = "admin1")
@Test
public void postAddEmployeetFormFailureStatementDoesNotExist() throws Exception{
    mockMvc.perform(post("/balances/{balancesId}/edit", 2)
        .param("username", "lyle")
        .with(csrf()))
        .andExpect(status().isOk())
        .andExpect(model().attribute("message","Either the employee or the statement can´t be found, try again"))
        .andExpect(view().name(BALANCE_LISTING));
}

@WithMockUser(value = "admin1")
@Test
public void postAddEmployeetFormFailureEmployeeDoesNotExist() throws Exception{
    mockMvc.perform(post("/balances/{balancesId}/edit", TEST_STATEMENT_ID)
        .param("username", "pipencio")
        .with(csrf()))
        .andExpect(status().isOk())
        .andExpect(model().attribute("message","Either the employee or the statement can´t be found, try again"))
        .andExpect(view().name(BALANCE_LISTING));
}

@WithMockUser(value = "admin1")
@Test
public void showStatementDetailsSuccess() throws Exception {
    mockMvc.perform(get("/balances/{balancesId}", TEST_STATEMENT_ID))
        .andExpect(model().attributeExists("dataPoints"))
        .andExpect(model().attributeExists("tokens"))
        .andExpect(model().attributeExists("subs"))
        .andExpect(model().attributeExists("salaries"))
        .andExpect(model().attributeExists("balance"))
        .andExpect(status().isOk())
        .andExpect(view().name(BALANCE_DETAILS));
}

@WithMockUser(value = "lyle")
@Test
public void showStatementDetailsBeingEmployeeSuccess() throws Exception {
    mockMvc.perform(get("/balances/{balancesId}", TEST_STATEMENT_ID))
        .andExpect(model().attributeExists("dataPoints"))
        .andExpect(model().attributeExists("tokens"))
        .andExpect(model().attributeExists("subs"))
        .andExpect(model().attributeExists("salaries"))
        .andExpect(model().attributeExists("balance"))
        .andExpect(status().isOk())
        .andExpect(view().name(BALANCE_DETAILS));
}

@WithMockUser(value = "pipencio")
@Test
public void showStatementDetailsBeingEmployeeFail() throws Exception {
    mockMvc.perform(get("/balances/{balancesId}", TEST_STATEMENT_ID))
        .andExpect(model().attribute("message","Employee not cualified to see this information"))
        .andExpect(view().name(BALANCE_LISTING));
}
```

Nombre: Mineral House Spa Grupo: G2-07

Principales problemas encontrados

El mayor problema que me he encontrado fue la generación de estadísticas para los Balances. Consulté varias librerías hasta que opté por la combinación de CanvasJS con GSON.

Sin embargo, la configuración inicial de GSON sugería que el objeto GSON se crease directamente en el JSP, opción que descarté tanto por la baja seguridad que esto significaba, como por incompatibilidades con los Services, además de saltarse la arquitectura del proyecto.

Tras los cambios oportunos, el objeto GSON pasó a crearse dentro del Service

Otros comentarios

He realizado la propuesta 1 a A+, consultar documento