# Informe individual de actividades del proyecto Parchis&Oca – Septiembre

## Datos generales

URL del Repositorio de GitHub: https://github.com/dantolvil/dp1-2021-2022-g1-septiembre
Nombre de usuario en GitHub: dantolvil
Rama del usuario en Github: dantolvil

## Participación en el proyecto

### Historias de usuario en las que he participado

Al tratarse de un grupo compuesto por un único miembro, todas las historias de usuario han sido implementadas por Daniel Toledo Villalba.

A continuación, se indican algunos ejemplos de estas historias de usuario.
H10 – Listado de jugadores registrados y H11 – Administrar los jugadores.

En total se han implementado 13 historias de usuario.

### Reglas de negocio

Al tratarse de un grupo compuesto por un único miembro, todas las reglas de negocio han sido desarrolladas por Daniel Toledo Villalba.

A continuación, se indican algunos ejemplos de estas reglas de negocio.
R2 – Partidas sin nombre y R5 – Imposibilidad de registro en el sistema sin completar todos los campos.

En total se han aplicado 5 reglas de negocio.

### Funcionalidad implementada

Al tratarse de un grupo compuesto por un único miembro, todas las clases, interfaces y vistas han sido implementadas por Daniel Toledo Villalba.

A continuación, se indican algunos ejemplos de la funcionalidad implementada.
Clases de modelo Java: Game, GameAction, GameBoard.
Interfaces de los repositorios: GameRepository, GamePieceRepository, BoardFieldRepository.
Clases de servicios: GamerService, PlayerService, TurnService.
Clases de controladores: GameController, OcaController, ParchisController.
Vistas jsp con las siguientes carpetas: game, admins, players.

En total se han implementado 10 clases de dominio java, 10 servicios, 10 repositorios y 7 controladores para las diferentes entidades sin contar todo lo relacionado a BaseEntity y Authorities.
Adicionalmente se han implementado 8 archivos.jsp, 3 para la carpeta admins, 3 para game y 2 para players.

Por último, se han implementado dos vistas adicionales que son Welcome y Exception para mostrar las excepciones y el home de la aplicación web.

## Pruebas implementadas

### Pruebas unitarias

Al tratarse de un grupo compuesto por un único miembro, todas las pruebas unitarias han sido implementadas por Daniel Toledo Villalba.

A continuación, se indican algunos ejemplos de estas pruebas unitarias.
Test unitarios asociados a los servicios: BoardfieldServiceTest, y GameServiceTests.
Test unitarios asociados al modelo java: ValidatorTests
Test unitarios asociados a los controladores: GameControllerTests y ParchisControllerTest.

26 métodos con etiqueta @Test

### Pruebas de Controlador

Al tratarse de un grupo compuesto por un único miembro, todas las pruebas positivas y negativas asociadas a los controladores han sido implementadas por Daniel Toledo Villalba.

A continuación, se indican algunos ejemplos de estas pruebas.
Escenarios positivos: H1+E1, H2+E1, H4+E1.
Escenarios negativos: H3-E1, H8-E1, H12-E1.

# Ejemplos de funcionalidades implementadas

## Entidades

### - Entidad Player

**Player**

· RUTA:
src\main\java\org\springframework\samples\parchis_oca\model\ Player.java

· Está clase java engloba todo lo necesario para la entidad Player del proyecto, en ella se detallan los atributos de la clase java Player: email, además hereda los atributos de la clase Peson, firstName y lastName.
Además, se incluyen las relaciones de esta entidad con la entidad Turn y GamePiece.
Su función es persistir el modelo de dominio UML a lenguaje java para la entidad Player y definir los atributos y relaciones de esta entidad.

```
@Getter
@Setter
@Entity
@Table(name = "players")
public class Player extends Person {

	//Attributes

	@Column(name = "email")
	@NotBlank
	private String email;

	//Relationships
	@OneToMany(cascade = CascadeType.ALL)
	private List<Turn> turn;

	@OneToOne(cascade = CascadeType.ALL)
	@JoinColumn(name = "gamePiece")
	private GamePiece gamePiece;

	public boolean checkAlreadyCreatedGames() {

		return false;
	}

}
```

## Servicio

**PlayerService**

· Ruta: src\main\java\org\springframework\samples\parchis_oca\service\PlayerService.java

· Esta clase corresponde al servicio de la entidad Player, en ella se incluyen los métodos Crud y demás métodos que serán nutridos por el repositorio 'PlayerRepository' asociado desde el que se accederá a la base de datos. Estos métodos se usarán en el controlador de la entidad Player 'PlayerController'.

· Se han incluido comentarios en el código para detallar los diferentes métodos.

· Se ha usado @Autowired para inyectar dependencias con otras entidades dentro de Spring.

```
@Service
public class PlayerService {

	private PlayerRepository playerRepository;

	@Autowired
	public PlayerService(PlayerRepository playerRepository) {
		this.playerRepository = playerRepository;
	}

//Recuperar el jugador actual
public Optional <Player> getCurrentPlayer() {
	 Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
	 String currentPrincipalName = authentication.getName();
	 return findPlayer(currentPrincipalName);
    }
```

```
    public Optional <Player> findPlayer(String player) {
        return playerRepository.findById(player);
    }

    //Guardar y crear un jugador @Transactional
      @Transactional
      public void savePlayer(Player player) throws DataAccessException {
            //creating player
            playerRepository.save(player);

      }

}
```

## Controlador

**PlayerController**

RUTA:
src\main\java\org\springframework\samples\parchis_oca\web\PlayerController.java

· En el controlador de la entidad Player se han incluido una serie de métodos para administrar las peticiones HTTP y realizar las redirecciones a las vistas correspondientes.
Mediante @Autowired se realizan las inyecciones de dependencia entre entidades.

· Se han incluido comentarios en el código para detallar los diferentes métodos.

```
@Controller
public class PlayerController {

    private static final String VIEWS_PLAYER_CREATE_OR_UPDATE_FORM =
    "players/createOrUpdateOwnerForm";

    private static final String VIEWS_PLAYERS_CREATE_OR_UPDATE_FORM = null;

    @Autowired
    private PlayerService playerService;

    @InitBinder
    public void setAllowedFields(WebDataBinder dataBinder) {
            dataBinder.setDisallowedFields("id");
    }

    //HTTP(GET). Mediante GET se recuperan los datos de la entidad Player.
    @GetMapping(value = "/player/new")
    public String initCreationForm(Map<String, Object> model) {
        Player player = new Player();
        model.put("player", player);
        return VIEWS_PLAYER_CREATE_OR_UPDATE_FORM;
    }
```

```
    //HTTP(POST). Si no se producen errores en el modelo, se guardan los datos en el
    servicio usando el método save(). // Por el contrario, si se produce un error se
    nos redirecciona a la vista con el formulario para la creación.
    @PostMapping(value = "/players/new")
      public String processCreationForm(@Valid Player player, BindingResult result)
      {
           if (result.hasErrors()) {
             return VIEWS_PLAYER_CREATE_OR_UPDATE_FORM;
           }
           else {
             this.playerService.savePlayer(player);

             return "redirect:/players/" + player.getId();
           }
      }

     @GetMapping(value = "/players/find")
      public String initFindForm(Map<String, Object> model) {
             model.put("player", new Player());
             return "players/findPlayers";
      }

     //Mediante el uso del servicio PlayerService se obtiene el jugador que va a
     //ser modificado a través de su Id y se redirecciona a la vista de edición.
     @GetMapping(value = "/players/{playerId}/edit")
      public String initUpdatePlayerForm(@PathVariable("playerId") String playerId,
      Model model) {
             Optional<Player> player = this.playerService.findPlayer(playerId);
             model.addAttribute(player);
             return VIEWS_PLAYER_CREATE_OR_UPDATE_FORM;
      }


     //Al realizar la edición del jugador se comprueba si presenta errores,
     //si todo ha ido bien se re direcciona a la vista de los detalles del jugador.
     //Por el contrario, si se encuentran errores se devolverá a la vista anterior.
     @PostMapping(value = "/players/{playerId}/edit")
      public String processUpdatePlayerForm(@Valid Player player, BindingResult
      result, @PathVariable("playerId") int playerId) {
             if (result.hasErrors()) {
                    return VIEWS_PLAYERS_CREATE_OR_UPDATE_FORM;
             }
             else {
                    player.setId(playerId);
                    this.playerService.savePlayer(player);
                    return "redirect:/players/{playerId}";
             }
      }

     //Se muestra el jugador que se obtiene por url a través del playerId.
     @GetMapping("/players/{playerId}")
      public ModelAndView showOwner(@PathVariable("playerId") String playerId) {
             ModelAndView mav = new ModelAndView("players/playerDetails");
             mav.addObject(this.playerService.findPlayer(playerId));
             return mav;
      }

}
```

### Repositorio

**PlayerRepository**

RUTA:
src\main\java\org\springframework\samples\parchis_oca\repository \PlayerRepository.java

· A través del repositorio de jugador(PlayerRepository) se obtienen los registros de la base de datos mediante consultas JPQL.
· En este caso no se ha implementado ninguna consulta JPQL para la entidad Player.

```
public interface PlayerRepository extends Repository<Player, Integer> {

     void save(Player player) throws DataAccessException;

     Optional<Player> findById(String player);

}
```

### Validador y anotación asociada.

Validador para la creación de contraseña

Se ha implementado una validación para la creación de las contraseñas en el sistema como se indica a continuación

@Pattern(regexp = "^(?=.[a-z])(?=.[A-Z])(?=.*\d)[a-zA-Z\d]{8,}$", message=" La contraseña debe incluir como mínimo 8 caracteres, una letra mayúscula, una letra minúscula y al menos un número").

## Ejemplos de pruebas implementadas

### Pruebas unitarias

**GameActionServiceTest**

RUTA:
src\test\java\org\springframework\samples\parchis_oca\service\
GameActionServiceTest.java

· Se quiere comprobar que se puede encontrar una acción de juego por su número de acción, su nombre de acción y su elección.

```
@Test
  void findAndSaveGameAction()
  {
       GameAction gameAction = new GameAction();
       gameAction.setAction("test");
       gameAction.setActionChoose(true);
       gameAction.setActionNumber(1);
       this.gameActionService.save(gameAction);

       Assertions.assertNotNull(this.gameActionService.findById(1));
```

```
    }
```

· **Arrange/Fixture**: Se crea una nueva acción de juego

```
        this.gameActionService.save(gameAction);
```

· **Assert**: Se comprueba que es posible encontrar la acción de juego creada anteriormente.

```
        Assertions.assertNotNull(this.gameActionService.findById(1));
```

## Pruebas de controlador

**GameControllerTests**

RUTA:
src\test\java\org\springframework\samples\parchis_oca\web\GameControllerTests.java

· **Arrange/Fixture:**

```
@ExtendWith(SpringExtension.class)
@WebMvcTest(value = GameController.class, includeFilters =
@ComponentScan.Filter(type = FilterType.ASSIGNABLE_TYPE), excludeFilters =
@ComponentScan.Filter(type = FilterType.ASSIGNABLE_TYPE, classes =
WebSecurityConfigurer.class), excludeAutoConfiguration =
SecurityAutoConfiguration.class)
public class GameControllerTests {


    @Autowired
    private GameController gameController;


    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private PlayerService playerService;

    @MockBean
    private TurnService turnService;

    @MockBean
    private GameService gameService;

    private Optional<Game> createTestCreatedGame(){
        Game game = new Game();
        Player creator = createTestPlayer().get();

        game.setCreator(creator);
        game.setMaxPlayer(2);
        game.setName("newgame");
        game.setMaxPlayer(2);
        game.setCurrentPlayers(creator);

        game.setStatus(GameStatus.INITIED);

        return Optional.of(game);
    }
```

```
    private Optional<Player> createTestPlayer(){
      Player testPlayer = new Player();

      testPlayer.setFirstName("Dan");
      testPlayer.setLastName("Toledo");
      testPlayer.setEmail("dtv@web.es");

      Optional<Player> userOptional = Optional.of(testPlayer);
      return userOptional;
   }

   private List<Game> createTestGame(){
      List<Game> games = new ArrayList<>();
      return games;
   }

   @BeforeEach
   void init(){
      when(gameController.findAllCreatedGames())
         .thenReturn(createTestGame());

      when(playerService.getCurrentPlayer())
            .thenReturn(createTestPlayer());
   }

}
```

**Test Negativo:**

```
    @Test
    void testJoinGameShouldFail() throws Exception{

      mockMvc.perform(post("/game/join"))
          .andDo(print())
          .andExpect(status().isOk()) .andExpect(view().name("exception"));
   }
```

**Test Positivo:**

```
    @Test
    @WithMockUser(value = "player1")
    void testViewForJoinGame() throws Exception{
      mockMvc.perform(get("/game/join"))
         .andDo(print())
         .andExpect(status().isOk())
         .andExpect(view().name("game/joinGameForm"));

    }
```

Nombre: Daniel Toledo Villalba Grupo: G1-Septiembre 2022

## Principales problemas encontrados

**Sincronización IDE Eclipse y Github:**
Al usar el IDE eclipse he encontrado varios problemas para configurar la sincronización del repositorio local de Eclipse con el repositorio de Github. Tras varios intentos con mal resultado y dado que debía seguir avanzando en la implementación del proyecto para la entrega, decidí realizar todos los commits y subir todos los documentos directamente en la herramienta Github sin usar el conector de Eclipse.

## Otros comentarios