Nombre: Enrique Moreno Vázquez Grupo: G3-14

Informe individual de actividades del proyecto

Datos generales

URL del Repositorio de GitHub: https://github.com/gii-is-DP1/dp1-2020-g3-14
Nombre de usuario en GitHub: enrmorvaz, enrique o enrik
Rama del usuario en Github: enrmorvaz

Participación en el proyecto

Historias de usuario en las que he participado

He implementado completas las historias de usuario HU-1, HU-2, HU-3, HU-10, HU-11, HU-18 además, dichas historias de usuario la he desarrollado con mi compañero José Francisco Regadera.

He implementado completas las historias de usuario HU-2, HU-10, además, dichas historias de usuario la he desarrollado con mi compañero Pedro Jesús Muñoz.

He implementado completas las historias de usuario HU-16, HU-17, HU-7, HU-9, H-19 además, dichas historias de usuario la he desarrollado con mi compañero Javier Hidalgo.

He implementado completas las historias de usuario HU-15 además, dichas historias de usuario la he desarrollado con mi compañero Pedro Pablo Carvajal.

He implementado reglas de negocio completas R1, R2, R4, R5, además, he desarrollado junto a mi compañero José Francisco Regadera la Regla de negocio R1, R2 con todos los compañeros, R4 con mis compañeros José Francisco Regadera y Javier Hidalgo y R5 con mi compañero Javier Hidalgo.

Nombre: Enrique Moreno Vázquez Grupo: G3-14

Funcionalidad implementada

He implementado el controlador ActividadController, AgenActController, ComentarioActividadController, ComentarioHotelController, HabitacionController, HotelController, ReservaActividadController, ReservaHabitacionController, ReservaVueloController y UserController

He añadido varios métodos al servicio ActividadService, AgenActService, ComentarioActividadService, ComentarioHotelService, HabitacionService, HotelService, ReservaActividadService, ReservaHabitacionService y ReservaVueloService,

También he creado la entidad Actividad, AgenAct, ComentarioActividad, ComentarioHotel, Habitación, Hotel, ReservaActividad, ReservaHabitacion y ReservaVuelo

Por ultimo las vistas actividades, agenacts, habitaciones, hoteles, reservaActividad, ReservaHabitacion, ReservaVuelo.

He implementado los validadores específicos para la contraseña, las fechas en las reservas, el número de tarjeta y el CVC en las reservas.

Esto hace un total de 28 clases implementadas por mí y 9 interfaces definidos.

Pruebas implementadas

Pruebas unitarias

He creado tests unitarios para 3 servicios (ReservaHabitacion, ReservaActividad, ReservaVuelo), 3 controladores (VueloController, cometarioActividadController, ReservaActividadControllerTest). Eso hace un total de 6 clases de test unitarios con un total de 28 métodos anotados con @Test.

Pruebas de Controlador

He creado 1 casos de prueba positivo y 1 negativos de controlador para la HU-9

He creado 2 casos de prueba positivo y 1 negativos de controlador para la HU-17

Ejemplos de funcionalidades implementadas

Entidades (máximo de dos ejemplos)

Entidad ReservaActividad

Ruta: /src/main/java/org/springframework/samples/petclinic/model/ReservaActividad.java

```
public class ReservaActividad extends BaseEntity{

    @Column(name = "fechareserva")
    @DateTimeFormat(pattern = "yyyy/MM/dd")
    private LocalDate fechaReserva;

    @Column(name = "entrada")
    @DateTimeFormat(pattern = "yyyy/MM/dd")
    @FutureOrPresent
    private LocalDate entrada;

    @Column(name = "numeroTarjeta")
    @CreditCardNumber
    private String numeroTarjeta;

    @Column(name = "cvc")
    @Pattern(regexp="\\d{3}",message = "Debe contener 3 dígitos")
    private String cvc;

    @Column(name="precio")
    private Double precioFinal;

    @ManyToOne
    @JoinColumn(name = "username")
    private User user;

    @OneToOne
    @JoinColumn(name = "actividad_id")
    private Actividad actividad;
```

En esta foto tenemos los atributos de la entidad reservaActividad, dichos atributos contienen la información como de la fecha de reserva (fechaReserva) que contiene un format para para pasar el elemento a LocalDate además tenemos la anotación de validacion sintáctica para que la fecha deba de ser presente o futura, entrada dicho atributo es donde se almacena la fecha de la realización de la actividad dicho valor se obtiene en el controlador cuya fecha debe de ser presente o futura, numeroTarjeta que tenemos el algoritmo Luhn por la anotación @CreditCardNumber en la cual debe de introducir un número de tarjeta correcto (como por ejemplo 371449635398431), cvc es el código de seguridad de la tarjeta (ese número se encuentra en la parte trasera) en la cual tenemos un pattern en la que obliga al usuario a poner 3 dígitos en el caso de error muestra el mensaje “Debe contener 3 dígitos” y por último el precio que sería lo que vale la reserva de dicha actividad.

Los dos últimos atributos son las relaciones que tendríamos con las entidades de actividad como son user (ManyToOne) donde se encuentra el usuario en la cual realizar la reserva y el atributo actividad (OneToOne) que es la relación con la actividad de la que ha reservado.
A continuación, tenemos los getters and setters de los atributos mencionados anteriormente:

```
public Double getPrecioFinal() {
    return precioFinal;
}

public void setPrecioFinal(Double precioFinal) {
    this.precioFinal = precioFinal;
}

public LocalDate getFechaReserva() {
    return fechaReserva;
}

public void setFechaReserva(LocalDate fechaReserva) {
    this.fechaReserva = fechaReserva;
}

public LocalDate getEntrada() {
    return entrada;
}

public void setEntrada(LocalDate entrada) {
    this.entrada = entrada;
}

public String getNumeroTarjeta() {
    return numeroTarjeta;
}

public void setNumeroTarjeta(String numTarjeta) {
    this.numeroTarjeta = numTarjeta;
}

public String getCvc() {
    return cvc;
}

public void setCvc(String cvc) {
    this.cvc = cvc;
}
```

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

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

public Actividad getActividad() {
    return actividad;
}

public void setActivdad(Actividad actividad) {
    this.actividad = actividad;
}
```

Entidad ComentarioHotel
Ruta: /src/main/java/org/springframework/samples/petclinic/model/ComentarioHotel.java

```
public class ComentarioHotel extends BaseEntity{

    @Column(name = "puntuacion")
    @Range(min=1,max=10)
    private Integer puntuacion;

    @Column(name = "mensaje")
    @NotEmpty
    private String mensaje;

    @ManyToOne
    @JoinColumn(name = "hotel_id")
    private Hotel hotel;
```

En esta foto tenemos los atributos de la entidad comentarioHotel, el atributo de puntuación es el valor que ha dado sobre el hotel dicho valor lo tenemos en un rango de entre 1 hasta 10 ya que si pone un valor que no se encuentra en dicho rango se le indicara al usuario de que debe de establecer un valor adecuando, el atributo mensaje es el comentario que el usuario debe de decir, debe de ser obligatorio.

Para el ultimo atributos es la relación que contiene ComentarioHotel con Hotel (ManyToOne).

Servicio

ReservaActividadService

Ruta:

/src/main/java/org/springframework/samples/petclinic/service/ReservaActividadService.java /

```
private ReservaActividadRepository reservaActividadRepository;

@Autowired
public ReservaActividadService(ReservaActividadRepository reservaActividadRepository) {
    this.reservaActividadRepository = reservaActividadRepository;
}

@Transactional
public void saveReservaActividad(ReservaActividad reserva) throws DataAccessException {
    reservaActividadRepository.save(reserva);
}

@Transactional
public void deleteReservaActividad(ReservaActividad reserva) throws DataAccessException {
    reservaActividadRepository.delete(reserva);
}

@Transactional(readOnly = true)
public ReservaActividad findReservaActividadById(int id) throws DataAccessException {
    return reservaActividadRepository.findById(id);
}

@Transactional(readOnly = true)
public Collection<ReservaActividad> buscarReservaActividad(String username) throws DataAccessException {
    return reservaActividadRepository.findReservaActividadByUserLike(username);
}
```

Los métodos establecidos en service son:

saveReservaActividad(ReservaActividad reserva): Este método se encargar de guardar en la base de datos la reserva que viene por parámetros.

deleteReservaActividad(ReservaActividad reserva): Este método se encargar de eliminar en la base de datos la reserva que viene por parámetros.

findReservaActividadById(int id): Este método devuelve un objeto reservaActividad que contenga como id el valor que viene por parámetros.

Collection<ReservaActividad> buscarReservaActividad(String username): devuelve una colección de reservaActividad, de las reservas que ha realizado un usuario, para identificar dicho usuario aplicamos como parámetro su username ya que reservaActividad tiene un objeto user y preguntamos por su username.

Controlador

ReservaActividadController

Ruta:

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

```java
@Controller
@RequestMapping("/actividades/{actividadId}")
public class ReservaActividadController {

    private ReservaActividadService reservaActividadService;
    private ActividadService actividadService;
    private UserService userService;
    private static final String VIEWS_RESERVAACTIVIDAD_CREATE_FORM = "reservaActividad/createReservaActividadForm";

    @Autowired
    public ReservaActividadController(final ReservaActividadService reservaActividadService,final ActividadService actividadService, final UserService userService) {
        this.reservaActividadService = reservaActividadService;
        this.actividadService=actividadService;
        this.userService= userService;
    }

    @GetMapping(value = "reservaActividad/new")
    public String initCreationForm(Map<String, Object> model) {
        ReservaActividad reservaActividad = new ReservaActividad();
        model.put("reservaactividad", reservaActividad);
        return VIEWS_RESERVAACTIVIDAD_CREATE_FORM;
    }

    @PostMapping(value = "reservaActividad/new")
    public String processCreationForm(@PathVariable("actividadId") int actividadId,

            @Valid ReservaActividad reservaActividad, BindingResult result,Map<String, Object> model) {
        reservaActividad.setFechaReserva(LocalDate.now());
        Actividad a = this.actividadService.findActividadById(actividadId);
        reservaActividad.setEntrada(a.getFecha());
        if (result.hasErrors()) {
            model.put("reservaactividad", reservaActividad);

            return VIEWS_RESERVAACTIVIDAD_CREATE_FORM;
        } else {
            Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
            UserDetails userDetails = null;
            if (principal instanceof UserDetails) {
                userDetails = (UserDetails) principal;
                }
            String userName = userDetails.getUsername();
            User user= this.userService.findByUsername(userName);
            Set<User> ls = a.getUsers();
            ls.add(user);
            a.setUsers(ls);
            reservaActividad.setPrecioFinal(Double.valueOf(a.getPrecio()));
            reservaActividad.setActivdad(a);
            reservaActividad.setUser(user);
            this.reservaActividadService.saveReservaActividad(reservaActividad);
            return "redirect:"+reservaActividad.getId();
        }
    }

    @GetMapping("reservaActividad/{reservaActivdadId}")
    public ModelAndView showReservaActivdad(@PathVariable("reservaActivdadId") int reservaActividadId) {
        ModelAndView mav = new ModelAndView("reservaActividad/reservaActividadDetails");
        mav.addObject("reservaActividad", this.reservaActividadService.findReservaActividadById(reservaActividadId));
        return mav;
    }
```

El procesado de creación realiza lo siguiente, establecemos como fecha de la reserva de forma automática la fecha actual por LocalDate.now() una vez establecida dicho atributo de la reserva buscamos la actividad a través del id dicho id viene por parámetros en el método. Una vez obtenido modificamos en la reserva el atributo entrada ya que dicho atributo obtienes la fecha de la actividad que has reservado. Después preguntamos se si tiene errores en el caso de que tenga te reenvía al formulario de creación. En otro caso se realiza la reserva completa, Obtenemos el usuario a través del servicio user como parámetro el username. Una vez obtenido dicho usuario obtenemos todos los usuarios que han reservado dicha actividad mediante un conjunto, después añadimos al usuario y modificamos la actividad con la nueva lista con el usuario que ha realizado la reserva, después modificamos los atributos de

reservaActividad en la cual el precio final sería el valor que contiene el precio de la actividad, añadimos la actividad a la reserva y el usuario una vez realizado todo esto guardamos la reserva y se redirige a la vista detalla de la reserva.

Repositorio

```
public interface ReservaActividadRepository extends JpaRepository<ReservaActividad, Integer>{

    ReservaActividad findById(int id);

    @Query("select a from ReservaActividad a where a.user.username like %?1")
    Collection<ReservaActividad> findReservaActividadByUserLike(String username);
}
```

FindById(int id): realiza un consulta para obtener de la base de datos la actividad que contenga dicho identificador. Dicha consulta la hemos utilizado para poder trabajar con la reserva.
findReservaActividadByUserLike(String username): realiza una consulta donde obtiene todas las reservas que ha realizado el usuario con el username que viene por parámetros (Objeto user y preguntamos por el parámetro username). Devuelve una colección de reserva ya que puede que el usuario haya realizado más de una reserva de actividad. Dicha consulta la hemos utilizado para obtener un historial de todas las reservas que ha realizado el usuario, no únicamente de ReservaActividad si no también se ha realizado para ReservaVuelo y ReservaHotel en sus respectivos repositorios.

Entidades, Servicios, controlador se ha realizado con el compañero Javier Hidalgo.

Conversor o Formatter (si aplica)

No aplica.

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

Criterios de seguridad en la contraseña: Hemos utilizado un pattern para establecer una contraseña correcta. Dicho Pattern se muestra a continuación:

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

String password;

Este validador fue realizado junto Pedro Jesús Muñoz, Pedro Pablo Carvajal, José Francisco Regadera y Javier Hidalgo.

Etiquetas personalizadas (si aplica, máximo de dos ejemplos)

No aplica.

Ejemplos de pruebas implementadas

Pruebas unitarias (máximo de dos ejemplos)

ReservaActividadServiceTests

Ruta:

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

```java
@Test
@Transactional
public void shouldInsertReservaActividad() {
    Collection<ReservaActividad> reservaActividades = this.reservaActividadService.buscarReservaActividad("enrmorvaz");
    //No debe existir la Reservaactividad con ese asociado a ese nombre de usuario en la base de datos
    int found = reservaActividades.size();
    //Tamaño 0
    System.out.println("============================================================");
    System.out.println("ReservaActividad Usuario enrmorvaz no encontrada. Found= "+found);
    System.out.println("============================================================");

    //Creamos la actividad
    Actividad actividad = new Actividad();

    actividad.setNombre("ACTIVIDAD");
    actividad.setDescripcion("DESC");
    actividad.setValoracion(3);
    actividad.setDireccion("DIRECCION");
    actividad.setProvincia("Sevilla");
    actividad.setPrecio(12);
    actividad.setFecha(LocalDate.of(2021, 3, 3));
    this.actividadService.saveActividad(actividad);
    //Creamos la reserva
    ReservaActividad reservaActividad = new ReservaActividad();
    reservaActividad.setFechaReserva(LocalDate.now());
    reservaActividad.setEntrada(actividad.getFecha());
    reservaActividad.setNumeroTarjeta("371449635398431");
    reservaActividad.setCvc("123");
    reservaActividad.setPrecioFinal(20.0);
    reservaActividad.setActivdad(actividad);
    //Creamos usuario
    User usuario = new User();
    usuario.setUsername("enrmorvaz");
    usuario.setDni("42932326Q");
    usuario.setTelefono("999999999");
    usuario.setPassword("Admin123");
    reservaActividad.setUser(usuario);
    this.userService.saveUser(usuario);
    this.reservaActividadService.saveReservaActividad(reservaActividad);

    System.out.println("assertThat"+reservaActividad.getId());
    assertThat(reservaActividad.getId()).isNotEqualTo(0);

    //Comprobamos que se ha añadido sin problemas
    reservaActividades = this.reservaActividadService.buscarReservaActividad("enrmorvaz");
    System.out.println("====================================");
    System.out.println(reservaActividades);
    System.out.println("====================================");
    assertThat(reservaActividades.size()).isEqualTo(found+1);
    System.out.println("reservaActividad enrmorvaz encontrada. Found= "+reservaActividades.size());
    System.out.println("============================================================");
}
```

Esta prueba consiste en comprobar que se realiza de forma correcta la reserva de una actividad para un usuario.

Partes del test:

Arrange/fixture: comprobamos que el usuario no tiene reserva, una vez comprobado, empezamos a crear la actividad estableciendo valores correctos, después empezamos a establecer valores para la reserva y el usuario.

Act: una vez establecido todo realizamos la operación de guardar la actividad y el usuario.

Assert: Finalmente establecemos una búsqueda de todas las reservas de dicho usuario y comprobamos que tiene la reserva creada anteriormente.

ReservaVueloServiceTests
Ruta:
/src/test/java/org/springframework/samples/petclinic/service/ReservaVueloServiceTests.java

```
@Test
@Transactional
public void shouldInsertReservaVuelo() {
    Collection<ReservaVuelo> reservaVuelos = this.reservaVueloService.buscarReservaVuelo("enrmorvaz");
    //No debe existir la ReservaVuelo con ese asociado a ese nombre de usuario en la base de datos
    int found = reservaVuelos.size();
    //Tamaño 0
    System.out.println("============================================================");
    System.out.println("ReservaVuelo Usuario enrmorvaz no encontrada. Found= "+found);
    System.out.println("============================================================");

    Vuelo vuelo = new Vuelo();
    vuelo.setBilletes(451);
    vuelo.setDestino("Cadiz");
    vuelo.setOrigen("Sevilla");
    vuelo.setPrecio(12);
    LocalDate fechaIda=LocalDate.of(2021, 10, 26);
    vuelo.setFechaIda(fechaIda);
    LocalDate fechaVuelta=LocalDate.of(2021, 11, 4);
    vuelo.setFechaIda(fechaVuelta);
    vuelo.setBilletes(3);

    ReservaVuelo reservaVuelo = new ReservaVuelo();
    reservaVuelo.setFechaReserva(LocalDate.now());
    reservaVuelo.setIda(vuelo.getFechaIda());
    reservaVuelo.setVuelta(vuelo.getFechaVuelta());
    reservaVuelo.setNumeroTarjeta("371449635398431");
    reservaVuelo.setCvc("123");
    reservaVuelo.setPrecioFinal(20.0);
    User usuario = new User();
    usuario.setUsername("enrmorvaz");
    reservaVuelo.setUser(usuario);

    this.reservaVueloService.saveReservaVuelo(reservaVuelo);

    System.out.println("assertThat"+reservaVuelo.getId());
    assertThat(reservaVuelo.getId()).isNotEqualTo(0);

    //Comprobamos que se ha añadido sin problemas
    reservaVuelos = this.reservaVueloService.buscarReservaVuelo("enrmorvaz");
    assertThat(reservaVuelos.size()).isEqualTo(found+1);
    System.out.println("reservaVuelo enrmorvaz encontrada. Found= "+reservaVuelos.size());
    System.out.println("============================================================");
}
```

Esta prueba consiste en comprobar que se realiza de forma correcta la reserva de un vuelo para un usuario.
Partes del test:
Arrange/fixture: comprobamos que el usuario no tiene reserva, una vez comprobado, empezamos a crear el vuelo estableciendo valores correctos, después empezamos a establecer valores para la reserva y el usuario.
Act: una vez establecido todo realizamos la operación de guardar del vuelo y el usuario.
Assert: Finalmente establecemos una búsqueda de todas las reservas de vuelo de dicho usuario y comprobamos que tiene la reserva creada anteriormente.

Pruebas unitarias parametrizadas (si aplica)

No aplica.

Pruebas de controlador

VueloControllerTests.
Ruta: /src/test/java/org/springframework/samples/petclinic/web/VueloControllerTests.java

Test positivo:

```
@BeforeEach
void setup() {
    vuelo = new Vuelo();
    vuelo.setId(TEST_VUELO_ID);
    vuelo.setBilletes(2);
    vuelo.setDestino("Malaga");
    vuelo.setOrigen("Sevilla");
    vuelo.setPrecio(12);
    LocalDate fechaIda=LocalDate.of(2020, 10, 26);
    vuelo.setFechaIda(fechaIda);
    LocalDate fechaVuelta=LocalDate.of(2020, 11, 4);
    vuelo.setFechaVuelta(fechaVuelta);

    given(this.vueloService.findVueloById(TEST_VUELO_ID)).willReturn(vuelo);
    System.out.println(vuelo);
}

    @WithMockUser(value = "spring")
        @Test
    void testProcessCreationFormSuccess() throws Exception {
        mockMvc.perform(post("/vuelos/new").param("billetes", "2")
                            .with(csrf())
                            .param("destino", "Malaga")
                            .param("origen", "Sevilla")
                            .param("precio", "12")
                            .param("fechaIda", "2020/10/26")
                            .param("fechaVuelta", "2020/11/04"))
                    .andExpect(status().is3xxRedirection());
    }
```

Realizamos una prueba con parámetros correcto, en la cual se comprueba de que dichos valores están correcto una vez comprobado te reenvía a la vista de detalles del vuelo.
Partes de la prueba:
Arrange/Fixure: Setup() es donde establecemos los valores del del vuelo.
Act: testProcessCreationFormSuccess() comprueba que los datos están establecido de forma correcta.
Assert: En caso de que sea así te redirige a la pagina de detalles de la reserva.

Test negativo:

```
@BeforeEach
void setup() {
    vuelo = new Vuelo();
    vuelo.setId(TEST_VUELO_ID);
    vuelo.setBilletes(2);
    vuelo.setDestino("Malaga");
    vuelo.setOrigen("Sevilla");
    vuelo.setPrecio(12);
    LocalDate fechaIda=LocalDate.of(2020, 10, 26);
    vuelo.setFechaIda(fechaIda);
    LocalDate fechaVuelta=LocalDate.of(2020, 11, 4);
    vuelo.setFechaVuelta(fechaVuelta);

    given(this.vueloService.findVueloById(TEST_VUELO_ID)).willReturn(vuelo);
    System.out.println(vuelo);
}
@WithMockUser(value = "spring")
    @Test
void testProcessCreationFormHasErrors() throws Exception {
    mockMvc.perform(post("/vuelos/new")
                        .with(csrf())
                        .param("billetes", "3")
                        .param("destino", "")
                        .param("origen", "Sevilla")
                        .param("precio", "24")
                        .param("fechaIda", "2020/10/26")
                        .param("fechaVuelta", "2020/11/04"))
            .andExpect(status().isOk())
            .andExpect(model().attributeHasErrors("vuelo"))
            .andExpect(model().attributeHasFieldErrors("vuelo", "destino"))
            .andExpect(view().name("vuelos/createOrUpdateVueloForm"));
}
```

Realizamos una prueba con parámetros correcto y vacíos, en la cual se comprueba de que dichos valores están correcto una vez comprobado te reenvía al mismo formulario de creación de vuelo indicando los errores establecidos.
Partes de la prueba:
Arrange/Fixure: Setup() es donde establecemos los valores del del vuelo.
Act: testProcessCreationFormSuccess() comprueba que los datos están establecido de forma correcta y el campo destino lo dejamos a vacío.
Assert: Detectara un error en la cual reenvía al formulario de creación de vuelo.

Las pruebas unitarias y de controlador se ha realizado junto al compañero Pedro Jesús Muñoz.

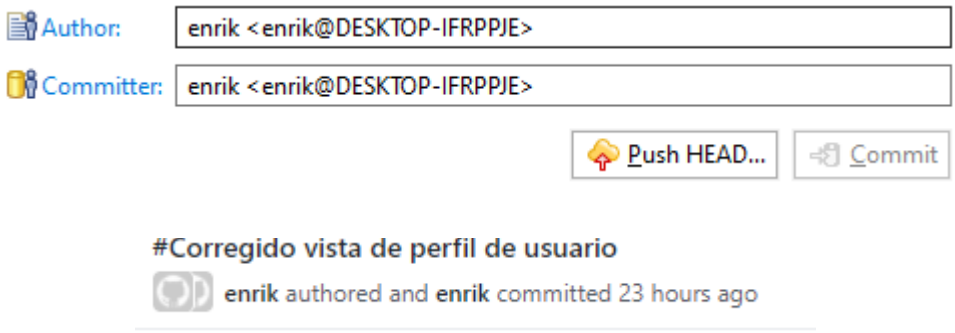
Nombre: Enrique Moreno Vázquez Grupo: G3-14

Principales problemas encontrados

Los problemas que he encontrado han sido a la hora de establecer una identificación que se diferencia los comentarios de actividad y hotel, la solución aplicada es separar los comentarios de hotel y actividad en dos entidades diferentes. Este mismo problema se aplica a la hora de realizar las reservas de habitación, actividad y vuelo que hemos establecido una entidad para cada una de ella.

Otros comentarios

A la hora de realizar los commits se puede establecer varios nombres como enrmorvaz, enrik o enrique. Esto se debe que en eclipse establece un nombre y no me he dado cuenta. Le Dejo una demostración.



No hemos trabajado en nuestras ramas creadas al principio del curso ya que el equipo tomo la decisión de que trabajaríamos utilizando el modelo de ramificación para git, GitFlow.