Participantes: Jordi Galí, Susanna Signoretti, Mariona Queralt

1. Análisis requisitos funcionales y no funcionales

Ejercicio 4

Objetivo de la aplicación (motivo de la creación)

Sistema de reserva de salas de conferencias y posible material.

Clases / Entidades

- **Clase Sistema**
  - **Atributos**:
    -id: Int
  - **Métodos**:
    +enviarRecordatorio(Int telefono String mail, Recordatorios recordatorio): Boolean
    +crearUsuario(Usuario usuario): void
- **Clase Usuarios**
  - **Atributos**:
    -nombre: String
    -apellidos: String
    -rol: String
    -id: Int
    -mail: String
    -contrasena: String,
  - **Métodos**:
    -iniciarSesion(String mail, String contrasena): Boolean (Este método llama a `validarCredenciales` de la clase `Autenticacion`)
    -cerrarsesion(): void
- **Subclase Clientes**
  - **Atributos**:
    -telefono: Int
  - **Métodos**:
- **Subsubclase Empleados**
  - **Atributos**:
  - **Métodos**:
- **Subsubclase Clientes_Externos**
  - **Atributos**:
  - **Métodos**:
- **Subclase Administradores**
  - **Atributos**:
  - **Métodos**:
    +consultarReportes(Reportes reporte): List<Reportes>
    +gestionSalas(String modifSalas): Salas (con el argumento `String modifSalas` tendré que llamar al método/función cada vez y solo podré cambiar 1 solo argumento (éste); en este caso se

haría con un switch –y es más elegante–; también podría poner todos los argumentos de la clase `Sala` y escribir `null` cuando no quisiese modificarlo; poner un argumento `modifSalas` con tipo "clase" `Sala`, no sería correcto en este caso (o si se pudiese hacer tendríamos que poner en la clase `Sala` más métodos de `get` y `set` para luego llamarlos desde este método)

+modificarEquipamiento (Int id): Boolean (con el argumento `Int id` tendré que llamar al método/función cada vez y solo podré cambiar 1 solo argumento (éste); en este caso se haría con un switch –y es más elegante–; también podría poner todos los argumentos de la clase `Equipamiento` y escribir `null` cuando no quisiese modificarlo; poner un argumento `modifEquipam` con tipo "clase" `Equipamiento`, no sería correcto en este caso (o si se pudiese hacer tendríamos que poner en la clase `Sala` más métodos de `get` y `set` para luego llamarlos desde este método)

+eliminarEquipamiento(Int id): void.

- Clase Reserva

❖ **Atributos**:

-id: Int
-fechaReservaInicio: Date
-fechaReservaFin: Date
-horaInicio: Int
-horaFin: Int
-sala: Sala
-precio: Float
-estadoReserva: String
-usuario: Clientes
-codigoReserva: String

❖ **Métodos**: *tener en cuenta que las reservas se añadan a una lista de reservas dentro de cada usuario*

+crearReserva(Int id, Date fechaReservaInicio, Date fechaReservaFin, Int horaInicio, Int horaFin, Sala sala, Float precio, String estadoReserva, Clientes usuario, String codigoReserva): Boolean (ponemos este método en la clase `Reserva` porque es más flexible que una reserva predefinida)

+filtrarDispoSalas(Salas filtroDispo): List<Salas>
+filtrarEquipamiento(Salas filtroEquipam): List<Salas>,
+seleccionarReserva(Reserva infoReserva): Reserva
+modificarReserva(String modifReserva): Reserva (con el argumento `String modifReserva` tendré que llamar al método/función cada vez y solo podré cambiar 1 solo argumento (éste); en este caso se haría con un switch –y es más elegante–; también podría poner todos los argumentos de la clase `Reserva` y escribir `null` cuando no quisiese modificarlo; poner un argumento `modifReserva` con tipo "clase" `Reserva`, no sería correcto en este caso (o si se pudiese hacer tendríamos que poner en la clase `Reserva` más métodos de `get` y `set` para luego llamarlos desde este método)

+cancelarReserva(): void (lo podríamos desarrollar más con `if`s, dentro de la función)

- Clase Pagos

❖ **Atributos**:

-id: Int
-datosPago: String

❖ **Métodos**:

+pagarReserva(Pagos pagar): Pagos
~~+cancelarReserva(): void~~

- ## Clase Salas
  - **Atributos**:
    -id: Int
    -superficieM2: Float
    -capacidad: Int
    -disponibilidad: Boolean
    -ubicacion: String
    -equipamiento: Equipamiento
    -descripcion: String
  - **Métodos**:
    ~~+crearListaInfoSalas(Salas infoSalas): List<Salas>~~ no hace falta poner ni lista ni crear salas porque se haría con el constructor en el código (se inicializaría con el `new`)

- ## Clase Equipamiento
  - **Atributos**:
    -id: Int
    -proyector: Int ~~Boolean~~
    -sistVideo: Int ~~Boolean~~
  - **Métodos**:
    +crearEquipamiento(): List<Equipamiento> (también podría eliminar esta función y pasarlo con el constructor, es decir crear el equipamiento a través del constructor)
    ~~+eliminarEquipamiento(Int id): void~~

- ## Clase Reportes
  - **Atributos**:
    -id: Int
    -fechaReporte: Date
    -descripcionReporte: String (si el reporte siempre tiene algún comentario)
    -reserva: Reserva
  - **Métodos**:
    +crearReportes(~~String descripcionReporte~~ (se pondría si quieres tener la opción de escribir una descripción cada vez que se crea un reporte, pero como el administrador solo puede consultar el reporte no lo podemos poner aquí), Reserva infoReservas): Boolean

- ## Clase Recordatorios
  - **Atributos**:
    -id: Int
    -fecha: Date
    -reserva: Reserva
    -usuario: Clientes (estaría bien añadir un comentario en el esquema: "se envía recordatorio por SMS y mail si hay los dos datos, sino se pone `null` y no se envía)
  - **Métodos**:
    +crearRecordatorio(String mensajeRecordatorio, ~~Reserva infoReserva~~ (no hace falta ponerlo aquí porque cuando añadimos `recordatorio1`... ya se crea la reserva porque está en atributos)): Boolean

- ## Clase Autenticación
  - **Atributos**:
    -id: Int

-idUsuario: int
-contrasenaUsuario: String
-rolUsuario: String
(estos tres se cogen automáticamente de alguna forma de la clase usuario)

- **Métodos**:
  +validarCredenciales(): Boolean
  +cambiarcontrasena(String contrasenaNueva): Boolean,
  +recuperarcontrasena(String mailUsuario, Int numeroIntentosEntrarContrasena (para simular si después de tres intentos se recupera contraseña; así es más seguro): Boolean;

Clase seguridad

- **Atributos**:
- **Métodos**:
  +cifrarDatos(): Boolean
  +verificarDatos(): Boolean

Requisitos funcionales

Clientes – reservas:

- El cliente hace reservas
  - Hacer una solicitud
    - Filtrar salas por:
      - Ubicación de las salas
      - Equipamiento
      - Capacidad
      - Consultar disponibilidad salas (ver calendario)
  - Confirmar / pagar la reserva → se envía mail de confirmación reserva+pago
  - Cancelar la reserva

Sistema – clientes

- El sistema envía un recordatorio 24h antes
  - (**Recordatorio – reserva**: El recordatorio se crea a partir de la información de la reserva)

Administradores – salas

- Los administradores pueden gestionar las salas por:
  - Equipamiento
  - Disponibilidad

Administradores – reportes

- Los administradores pueden acceder a reportes detallados sobre el uso de las salas (=reservas)
  - (**Reportes – reservas**: Se crean los reportes a partir de la información de las reservas).

Requisitos no funcionales

Seguridad

- Autenticación mediante credenciales de usuario.
- Diferentes niveles de acceso según el rol (Clientes vs. Administradores)
- Privacidad y seguridad de datos mediante cifrado y cumplimiento de normativa de protección de datos.

- **Disponibilidad**
  - ❖ Tiempo de actividad del 99.5%
- **Rendimiento**
  - ❖ Es capaz de manejar múltiples reservas simultáneas sin degradación del rendimiento.
- **Usabilidad**
  - ❖ Interfaz responsiva y compatible con todos los dispositivos modernos.
  - ❖ Interfaz intuitiva y fácil de utilizar.
- **Escalabilidad**
  - ❖ Puede soportar aumento de usuarios.

Relaciones y cardinalidad

- Sistema-usuario: Composición (sistema = todo)
  - **Cardinalidad: 1..*** | *1 sistema puede tener 1 o varios usuarios.*
- Usuarios-clientes: Herencia
- Clientes – Empleados: Herencia
- Clientes – Clientes_Externos: Herencia
- Usuarios – Administradores: Herencia
- Administradores – Reportes: Asociación
  - **Cardinalidad: 0..*** | *1 administrador puede consultar 0 o varios reportes.*
- Administradores – Salas: Asociación
  - **Cardinalidad: 1..*** | *1 administrador está asociado a 1 o más salas.*
- Reserva – Salas: Composición (salas = todo)
  - **Cardinalidad: 1..*** | *1 reserva tiene que estar asociada a 1 o más salas.*
- Reserva – Pagos: Composición (reserva = todo)
  - **Cardinalidad: 1** | *1 reserva está asociado a 1 solo pago.*
- Reserva – Clientes: Composición (clientes = todo)
- Salas – Usuarios: Asociación
- Equipamiento – salas: Composición (salas = todo) → en nuestro caso concreto.
  - **Cardinalidad: 1** | *1 equipamiento solo puede estar vinculado a 1 sala.*
- Reportes – Reserva: Composición (reserva = todo)
  - **Cardinalidad: 0..*** | *1 reporte puede estar asociado a 0 o varias reservas (0 seguramente será solo al inicio).*
- Recordatorios – Reserva: Composición (reserva = todo)
  - **Cardinalidad: 0..*** | *1 recordatorio puede estar asociado a 0 o varias reservas (0 seguramente será solo al inicio).*
- Sistema – Recordatorios: Asociación

Ambigüedades / Recomendaciones

- **Clases / Entidades**
  1. ¿El equipamiento está vinculado a la sala o se puede no contratar o contratar de manera extra? **Sí, no se puede contratar por separado.**
  2. ~~¿Las salas están abiertas a todo el mundo o solo a clientes específicos y sus empleados/clientes externos?~~
  3. ¿Se requiere un mail de confirmación al reservar la sala? **Sí, el mail de la persona que reserva.**

Participantes: Jordi Galí, Susanna Signoretti, Mariona Queralt

Diagrama