**ANÁLISIS REQUISITOS FUNCIONALES Y NO FUNCIONALES**

## Ejercicio 1

**Objetivo de la aplicación (motivo de la creación):**

* Gestionar un inventario para mejorar el control sobre productos y facilitar los procesos de recepción, almacenamiento y distribución.

**Clases / Entidades:**

* **Usuario**: nombre, apellidos, ID, correo, teléfono, contraseña
* **Producto**: nombre, categoría, precio, cantidad, fecha de caducidad.
* **Reporte**: productos (con sus propiedades), formatos PDF y Excel.
* **Notificación**: producto. \***Ambigüedad**: ¿Las notificaciones sobre un producto se realizan a todos los usuarios o solamente al que ha publicado el producto?

**Requisitos funcionales:**

1. Permitir registrar nuevos productos a los usuarios (relación usuarios-productos):
   * La aplicación debe permitir a los usuarios registrar nuevos productos, se tienen que especificar los detalles del producto: Nombre (podría ser tipo *string*), categoría, precio (podría ser tipo *float*), cantidad y fecha de caducidad.
2. Búsqueda de productos:
   * Los usuarios deben poder buscar productos usando el nombre, categoría y disponibilidad (esto son 3 atributos del propio producto).
3. Generación de reportes:
   * El sistema debe poder generar reportes de forma periódica y permitir descargarlos en PDF y Excel.
4. Notificaciones automáticas:
   * El sistema debe enviar alertas por correo electrónico para productos cercanos a la fecha de caducidad o bajo stock.

Según la información que me ha dado el cliente, ya no tengo más requisitos funcionales. Cuando eso pasa, nosotros pensamos la información lógica básica que se podría añadir y se la mandamos al cliente como propuesta.

Si nos dejamos alguna relación, cuando hagamos el diagrama nos podremos dar cuenta y la podremos añadir. Si en ese momento no nos damos cuenta tampoco, cuando programamos y si no nos damos cuenta, cuando usamos la aplicación allí nos podremos dar cuenta. Aunque lo suyo es tenerlo en las dos primeras fases.

**Requisitos no funcionales:**

1. Seguridad
   * Implementación de autentificación de usuarios para controlar el acceso y proteger la información sensible.
2. Disponibilidad:
   * El sistema tiene que tener una disponibilidad del 99% (no se le pone 100%, para no pillarte los dedos si algún día no funciona por un corte de luz… Hay aplicaciones del gobierno en las que solo se pueden hacer trámites de 8am a 8pm, porque luego hacen actualizaciones, pero no es lo habitual).
3. Rendimiento:
   * La mayoría de las operaciones se deben completar en menos de 2 segundos.
4. Usabilidad
   * Diseño responsivo para ser usable desde distintos tamaños de dispositivos.
   * Interfaz intuitiva.
5. Escalabilidad:
   * Tienen que tener la capacidad de manejar incrementos en el número de usuarios y productos sin afectar el rendimiento.

La disponibilidad y la escalabilidad nos indica que quizás lo mejor sería contratar servidores externos que pagas por cantidad de flujo.

**Relaciones:**

1. Relación usuario-productos
   * Un usuario registra de cero a muchos productos. **Cardinalidad** (como interactúan las distintas partes; *Propietat de les relacions que indica quants individus d'un tipus d'entitat, o quants registres d'una taula, en el cas d'una base de dades, poden correspondre com a mínim i com a màxim a cada individu de l'altre tipus d'entitat, o a cada registre de l'altra taula.*): 0…\*
   * Un producto puede estar asociado con un usuario. **Cardinalidad**: 1.
   * Un producto puede ser buscado por cero o más usuarios. Cardinalidad; 0..\*
2. Relación usuario-búsqueda
   * Un usuario puede buscar de uno a múltiples productos: **Cardinalidad**: 1..\*
3. Relación reportes-productos
   * Un reporte puede tener de cero a múltiples productos. Cardinalidad: 0..\*
   * Un producto puede existir en cero o más reportes. **Cardinalidad**: 0..\* (si fuese “hasta 5”, sería 0..5)
4. Relación notificación-usuarios
   * Un usuario puede recibir de 0 a múltiples notificaciones. **Cardinalidad**: 0..\*
   * Una notificación puede ser enviada de uno a múltiples usuarios. **Cardinalidad**: 1..\*

**Recomendaciones / Ambigüedades:**

Funcionales:

Registro de usuarios:

* + La aplicación debe permitir ingresar y registrar usuarios por ellos mismos o a través de un administrador. 🡪 a través de un administrador.
  + **Ambigüedad**: ¿Tenemos más de un tipo de usuario? ¿Cómo interactúa cada uno? 🡪 Sí, tenemos usuarios y administradores.

Generación de reportes

* + **Recomendación**: Añadir comentarios sobre cada producto en los reportes. 🡪 Sí, tenemos que poder añadir comentarios en los reportes de forma opcional.

Recomendación: ¿Se deben añadir proveedores para los productos?

Recomendación: añadir fecha en los reportes y notificaciones.

Recomendación no funcional: Contratar servidores externos. AMS.

Recomendación no funcional: Crear factores de multiple autentificación.

Recomendación clases: producto = id (de momento, le vendemos esto. Le proponemos que más adelante le podemos hacer ampliaciones respecto al tipo de producto: si es frágil, la ubicación en el almacén, si caduca o no…)

## Ejercicio 2

**Objetivo de la aplicación (motivo de la creación):**

* Desarrollar un sistema de gestión académica.

**Clases / Entidades:**

(clases usuario)

**Classe Sistema:**

Atributos: +id: int,

Métodos: +crearCurso(), +crearUsuario()

**Clase Usuario:**

Atributos: +nombre: String, +apellidos: String, +id: int, +correo: String, contrasena: String, +curso: String, +rol: String,

Métodos: iniciarSesion(String correos,String contrasenas): Boolean, +get y set para todos los atributos, -cerrarSesion():void,

**Subclase Profesor**: (hereda de Usuario)

Atributos:

Métodos: +crearCurso(Curso cursos): Boolean, +eliminarCurso(int ID), +enviarNotaexamen(Int notaexamenes, String correos), +publicarMateriales(String nombremateriales, String correos)

**Subclase Alumnos:** (hereda de Usuario)

Atributos: +curso: String

Métodos: +inscripcionCurso(Curso cursos): Boolean, +bajaCurso(Curso cursos): Boolean, +enviarTareas(Int tareas, String correos): Boolean.

(clases aplicación)

**Clase Curso:**

Atributos: +nombre: String, +id: int, +fechaInicio: Date, +fechaFinal: Date,

Métodos: iniciarSesion(String correos,String contrasenas): Boolean, +get y set para todos los atributos, -cerrarSesion():void

**Clase Examen**:

Atributos: +nombre: String, +id: int, +fechaExamen: Date, +notaExamen: Int,

ID, fecha, (resultados), +descripcionExamen: String,

Métodos: asignarNota(Int notas,String alumnos), enviarNota(Int notas, String correos)

**Clase Tareas:**

Atributos: +nombre: String, +id: int, +fechaTarea: Date, +descripcionTarea: String,

Métodos: asignarTarea(Int id, String alumnos), enviarTareas(Int tareas, String correos)

**Clase Materiales:**

Atributos: +nombre: String, +id: int, +descripcionMaterial: String,

Métodos: +realizarConsultamateriales(String nombres, Int alumnos)

(clases de comunicación)

**Clase Foro:**

Atributos: +nombre: String, +id: int, +fecha: Date, +temaforo: String,

Métodos: +crearForo(Int ids, String temaforos), +enviarAviso(String alumnos, String correos)

**Bases de datos bibiliográficas:**

Atributos: +id: int, +fecha: Date, +textobasedatos: String,

Métodos: +publicarEnlaceBaseDatos(Int ids, String correos)

**Plataformas e-learning:**

Atributos: +id: int, +fecha: Date, +textobasedatos: String,

Métodos: +publicarEnlacePlataformae-learning(Int ids, String correos)

**Requisitos funcionales:**

1. Gestión de cursos:
   * Los profesores pueden generar el material, contenidos del curso*, tareas y exámenes.*
   * Los alumnos deben inscribirse (a la universidad) y a los cursos.
   * Los profesores pueden modificar las calificaciones.
   * Los alumnos pueden acceder al material, a las tareas y a los exámenes.
2. Foro:
   * Interacción entre estudiantes y profesores: [Ver ambigüedad]
   * Tipo de contenido en el foro: [Ver ambigüedad]
3. Integrar sistemas externos:
   * Los profesores pueden modificar la entrada o establecer vínculos a bases de datos bibliográficas externas y plataformas e-learning.
   * Los alumnos pueden acceder a estos sistemas externos.

**Requisitos no funcionales:**

1. Seguridad
   * Sistema de autentificación de dos factores para alumnos y profesores (todos los usuarios)
   * Se encriptarán los datos para proteger la privacidad
2. Disponibilidad
   * La plataforma debe ser accesible el 99,9% del tiempo.
3. Rendimiento
   * Los tiempos de respuesta no deben sobrepasar los 3 segundos.
4. Usabilidad
   * Diseño responsivo compatible con diferentes navegadores y dispositivos (incluyendo móviles y tabletas).
   * Aplicación intuitiva y fácil de usar para todo el mundo.
5. Escalabilidad
   * Permite el número de usuarios y cursos sin degradación en el rendimiento.

**Relaciones:**

* Relación alumnos-alumnos:
  + Un alumno puede enviar 0 o varios mensajes a otro alumno: 0..\*
* Relación alumnos - profesores:
  + Un alumno puede tener uno o varios profesores. Cardinalidad: 1..\*
  + Un profesor puede tener x [ver ambigüedad] o varios alumnos. Cardinalidad: 1..\*
* Relación profesores-profesores
* Relación alumnos-cursos (tareas, exámenes, recursos)
* Relación profesores-cursos
* Relación alumnos-comunicación:
  + Un alumno puede hacer 0 o varias preguntas a un profesor. Cardinalidad: 0..\*
* Relación profesores-comunicación
* Recursos alumnos-sistemas externos (bases de datos, e-learning)
* Relación profesores-sistemas externos

**Tipos de relaciones:**

Profesor – usuario: Herencia.

Alumno – usuario: Herencia.

Profesor – alumno: Asociación.

Profesor – tarea: Agregación.

Profesor – examen: Agregación.

Profesor – materiales: Agregación.

Profesor – base de datos bibliograficas: Asociación.

Profesor – plataforma e-learning: Asociación.

Profesor – foro: Composición.

Alumno – foro: Composición.

Alumno – materiales: Asociación.

Alumno – tareas: Asociación.

Alumno – examen: Asociación.

Alumno – base de datos bibliograficas: Asociación.

Alumno – plataforma e-learning: Asociación.

Curso – materiales: Composición.

Curso – profesor: Composición.

Curso – alumno: Composición.

Curso– base de datos bibliograficas: agregación.

Curso – plataforma e-learning: agregación.

**Ambigüedades / Recomendaciones:**

Clases/Entidades:

* ¿Es necesario poner DNI, si cada ficha de alumno/profesor ya lo tiene vinculado al ID? 🡪 *¿Qué información se requiere de cada usuario?*

Requisitos funcionales:

* ¿Los resultados se requieren para las tareas, los exámenes o ambas?
* ¿Los alumnos ya están inscritos en la universidad o deben inscribirse aún?
* ¿Los profesores son los únicos que pueden iniciar un foro y administrar los foros o los alumnos también pueden iniciar y administrarlos?
* ¿En el foro solo se puede enviar y recibir texto plano o también otro tipo de archivos: vídeos, imágenes, sonido…?
* ¿En la aplicación hay algún tipo de mensaje privado?

Requisitos no funcionales:

Relaciones:

* ¿Cuál es el mínimo de alumnos por curso?

***Puesta en común clase***

*Requisitos funcionales*:

1. Inscripción en cursos: Los estudiantes se deben poder inscribir en cursos a través del sistema.
2. Acceder a materiales de estudio: Los estudiantes deben poder acceder a los materiales de estudio.
3. Gestión de los materiales de estudio: Los profesores deben poder gestionar los materiales de estudio.
4. Envío y gestión de tareas: Los estudiantes deben poder enviar tareas y poner comentarios en las que envían. Los profesores deben poder asignar y calificar las tareas.
5. Publicación de calificaciones: Los profesores publican calificaciones y los estudiantes reciben calificaciones. Incluir apartado de comentarios a modo *feedback*.
6. Publicación de resultados: Los profesores pueden publicar los resultados finales y los estudiantes deben poder recibir dichos resultados.
7. Foro de discusión: Los profesores y estudiantes deben poder interactuar con el foro. La interacción debe incluir: *publicar, eliminar, modificar publicaciones y responder*.
8. Integración con sistemas externos: El sistema debe ser capaz de integrarse con plataformas externas y bases de datos. [ver ambigüedad]
9. Creación de una cuenta: El sistema debe permitir crearse una cuenta.
10. Cursos: Los cursos pdeben poder contener materiales, calificaciones, tareas, exámenes. Y dado el caso, permitir clases en *streaming* (confirmar).

*Requisitos no funcionales:*

1. Usabilidad: interfaz intuitiva, fácil de usar y un diseño responsivo adaptado a móviles, tablets y ordenadores. Y accesible desde distintos navegadores.
2. Seguridad: Autenticación de dos factores y encriptación de datos sensibles.
3. Rendimiento: Tiempos de respuesta del sistema iguales o inferiores a 3 segundos.
4. Disponibilidad: Accesibilidad continua con un tiempo de actividad del 99.9%.
5. Escalabilidad: Capacidad de manejar un aumento en el número de usuarios y cursos eficientemente.

*Ambigüedades:*

* ¿En el foro de discusión se puede filtrar por tipos de temas, careras, tópicos, etc.?
* ¿El foro debe tener una moderación para palabras malsonantes o temas inapropiados?
* ¿En el foro los profesores serán administradores?
* ¿Existen otros roles aparte de profesores y estudiantes? ¿Y un estudiante puede ser profesor a la vez? ¿Debe existir algún administrador del sistema o por áreas (*es para que no tengamos que administrarlo siempre desde el departamento de informática*)
* ¿Cómo se gestiona la integración con plataformas y bases de datos externas?
* ¿Qué plataforma de e-learning y bases de datos se van a integrar? (Puede que no se sepan todas desde un inicio, pero las que sepan que nos las digan).
* ¿Se debe incluir un buscador por áreas, libros, cursos, etc.?
* ¿Se quiere incluir un chat para facilitar la comunicación estudiante-profesor, estudiante-estudiante?
* ¿Cómo se gestionan las bajas, excedencias, etc.?
* ¿Cómo se gestionan las sustituciones? ¿Una asignatura puede tener más de un profesor?
* ¿Quieres que el usuario cree una cuenta? ¿O que una vez inscrito en la universidad se cree automáticamente?
* ¿Qué formatos de materiales y exámenes queréis?
* ¿Cómo se realizan las entregas? ¿Con links, con updates, archivos, audio, video?
* ¿La plataforma debe poder permitir clases en streaming?
* Recomendación no funcional: contratación de servidores externos (pero nos podrían decir que quieren sus propios servidores, para tener ellos los datos).
* ¿La asistencia a clase es obligatoria? ¿Se debe controlar el acceso a clase?

Descartat

* ¿En esta plataforma se deben incluir la gestión de títulos, matrículas, tarifas…? Si es así, ¿se tiene que incluir el personal administrativo?

## Ejercicio 3

**Objetivo de la aplicación (motivo de la creación):**

* Implementar una aplicación móvil para compras online.

**Clases / Entidades:**

(clases usuario)

**Clase Usuario:**

Atributos: +nombre: String, +apellidos: String, +id: int, +correo: String, contrasena: String, +rol: String,

Métodos: -iniciarSesion(String correos,String contrasenas): Boolean, +get y set para todos los atributos, -cerrarSesion():void,

**Subclase Administrador**: (hereda de Usuario)

Atributos:

Métodos: +gestionarPromociones (Productos ofertas, String correos),

**Subclase Cliente:** (hereda de Usuario)

Atributos: +tipodemovil: String,

Métodos: +inscripcionCurso(Curso cursos): Boolean, +bajaCurso(Curso cursos): Boolean, +enviarTareas(Int tareas, String correos): Boolean,

**Clase Ofertas:**

Atributos: +nombreOferta: String, +idOfertas: int, +nombreProductoOferta: String, +tipoOferta: String, +fechaInicio: Date, +fechaFin: String,

Métodos: +crearOferta(Producto

**Clase Productos:**

Atributos: +nombreProducto: String, +idProductos: int, +cantidad: Int, +precio: Float, +descripción: String, +categoria: String, +disponibilidad: String, +ofertas: String,

Métodos: +crearProducto(Producto productos): Boolean, +eliminarProducto(int ID),

**Classe Inventarios:**

Atributos: +productos: list<Productos>

Métodos: +actualizarInventario (Int producto

**Pedidos:**

Atributos: +id: int, +fechaPedido: Date,

**Subclase Envíos:** (hereda de Pedidos)

Atributos: +id: int, +fechaEnvio: Date, +horarioEnvio: String, +direccionEnvio: String, +localizacionTiemporeal: String,

**Subclase Pagos:** (hereda de Pedidos)

Atributos: +id: int, +datosPago: String,

**Clase Notificaciones:**

Atributos: + id: int, +mensaje: String,

**Clase Reportes/Informes**

Atributos: + id: int, +mensaje: String, +ventas: String, +tendenciasConsumo: String

**Requisitos funcionales:**

1. Cliente compra producto
   * Consultar productos (catálogo, categorías)
   * Seleccionar productos
   * Pagar productos (elegir método de pago)
   * Seleccionar envío (elegir franja horaria y destino)
2. Administrador gestiona productos
   * Crear inventario: modificar, actualizar productos y sus datos.
   * Establece precios
   * Gestiona promociones
   * Mira la disponibilidad de los productos
   * Se informa de los KPI
   * Elaborar los reportes
   * Crear notificaciones (ofertas generales, envío…) [ver ambigüedad]

**Requisitos no funcionales:**

1. Seguridad
   * Gestionar la privacidad de datos sensibles, como los financieros y el domicilio
   * Autentificación segura para los usuarios.
2. Disponibilidad
3. Rendimiento
4. Usabilidad
   * Solo usable para dispositivos móviles, de todos los tamaños y sistemas operativos
5. Escalabilidad
   * Debe soportar aumento en épocas concretas de gran aforo.

**Relaciones:**

* Relación cliente-producto
* Relación cliente-envío
* Relación cliente-notificación
* Relación administrador-producto
* Relación administrador-notificación
* Relación administrador-reportes
* Relación producto-notificación
* Relación producto-reportes
* Relación producto-envío

**Ambigüedades / Recomendaciones:**

Clases/Entidades:

* ¿Es obligatorio que todos los clientes den una dirección postal o solo aquellos que quieren recibir los productos “en su casa” y cuando decidan apostar por esta opción?
* ¿La tienda de supermercados tiene reparto propio o se externaliza (algunas de las notificaciones también dependerían de esta decisión)?
* ¿Los clientes pueden usar todas las formas de pago posible: tarjeta, cuenta, cheque, efectivo, otros métodos de pago?
* ¿Las notificaciones que reciben los clientes se generan de forma automática?

Requisitos funcionales:

Requisitos no funcionales:

Relaciones:

## 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, (llama a validar cuenta) -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(Salas consultaSala, *String cambio* – tengo que llamar al método/función cada vez y solo puedo cambiar 1 solo argumento; se haría con un switch (pero es más elegante) o *poner todos los argumentos y poner null* ): Salas, +modificarEquipamiento (Int id *String cambio* – tengo que llamar al método/función cada vez y solo puedo cambiar 1 solo argumento; se haría con un switch (pero es más elegante) o *poner todos los argumentos y poner null*): Boolean; +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, … [TODOS LOS PARAMETROS DE RESERVA): Boolean (en reserva porque es más flexible que ya una reserva poredefinida) +filtrarDispoSalas(Salas filtroDispo): List<Salas>, +filtrarEquipamiento(Salas filtroEquipam): List<Salas>, +seleccionarReserva(Reserva infoReserva): Reserva, +modificarReserva(Reserva modifReserva *String cambio* – tengo que llamar al método/función cada vez y solo puedo cambiar 1 solo argumento; se haría con un switch (pero es más elegante) o *poner todos los argumentos y poner null*): Reserva, +cancelarReserva(): void (lo podríamos desarrollar más con ifs, dentro de la función). Va aquí porque

* **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, +ubicación: 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 (new… no se que)

* **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) ~~+modificarEquipamiento (Int id~~ *~~String cambio~~* ~~– tengo que llamar al método/función cada vez y solo puedo cambiar 1 solo argumento; se haría con un switch (pero es más elegante) o~~ *~~poner todos los argumentos y poner null~~*~~): Boolean;~~ ~~+eliminarEquipamiento(Int id): void.~~

* **Clase Reportes:**

Atributos: +id: Int, +fechaReporte: Date, +descripcionReporte: String (si el reporte siempre tiene algun comentario), +reserva: Reserva

Métodos: +crearReportes(*~~String descripcionReporte~~ (si tienes la opción de escribir cada vez que se crea. Pero como el admin solo puede consultar no podemos ponerlo aqui)*, Reserva infoReservas): Boolean

* **Clase Recordatorios:**

Atributos: +id: Int, +fecha: Date, +reserva: Reserva, +usuario: Clientes (añadir comentario en el esquema: se envia recordatorio por sms y mail si hay los dos datos y sino se pone null y no se envia)

Métodos: +crearRecordatorio(String mensajeRecordatorio, ~~Reserva infoReserva~~ cuando añadimos recordatorio1… ya se crea la reserva porque está en atributos ): Boolean

**Requisitos funcionales:**

1. 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 de reserva+pago
     + Cancelar la reserva
2. Sistema – clientes:
   * El sistema envía un recordatorio 24h antes.
     + (recordatorio – reserva: El recordatorio se crea a partir de la información de la reserva.)
3. Administradores – salas:
   * Los administradores pueden gestionar las salas, por:
     + Equipamiento
     + Disponibilidad
4. 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:**

1. 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.

* 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

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

**Relaciones:**

Sistema – Usuario: Composición

* Cardinalidad:

Usuarios – Clientes: Herencia

Clientes – Empleados: Herencia

Clientes – Clientes\_Externos: Herencia

Usuarios – Administradores: Herencia

Administradores – Reportes: Asociación

Administradores – Salas: Asociación

Reserva – Salas: Composición (salas = todo)

Reserva – Pagos: Composición (reserva = todo)

Reserva – Clientes: Composición (clientes = todo)

Salas – Usuarios: Asociación

Equipamiento – salas: Composición (salas = todo) 🡪 en nuestro caso concreto.

Reportes – Reserva: Composición (reserva = todo)

Recordatorios – Reserva: Composición (reserva = todo)

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.

Requisitos funcionales:

Requisitos no funcionales:

Relaciones:

## Ejercicio 5

**Objetivo de la aplicación (motivo de la creación):**

**Clases / Entidades:**

**Requisitos funcionales:**

**Requisitos no funcionales:**

**Relaciones:**

**Ambigüedades / Recomendaciones:**

Clases/Entidades:

Requisitos funcionales:

Requisitos no funcionales:

Relaciones:

## Ejercicio 6

**Objetivo de la aplicación (motivo de la creación):**

**Clases / Entidades:**

**Requisitos funcionales:**

**Requisitos no funcionales:**

**Relaciones:**

**Ambigüedades / Recomendaciones:**

Clases/Entidades:

Requisitos funcionales:

Requisitos no funcionales:

Relaciones:

Los atributos siempre serán privados.

Escribir código:

1. Empezar a escribir las clases más genéricas y luego clases y métodos que se vayan relacionando.