Informe de actividades

Proyecto: Sistema de gestión y registro para AplanchadosConAmor.

Durante las reuniones con el grupo, hemos cubierto por completo los requerimientos para la entrega del proyecto, donde utilizamos Java, Spring Boot y bases de datos H2. Las principales áreas de enfoque incluyen:

1. **Documentación de entidades:**
    a. **DocumentType**: Tipos de documentos identificativos para personas.
        i. **Relaciones:** Vinculado a múltiples *Person.*

    b. **Expense:** Registra los gastos incurridos, asociados a personas y distintos conceptos y métodos de pago.
        i. **Relaciones:** Conecta con *Person, PaymentMethod, PaymentType, ExpenseConcept, User.*

    c. **ExpenseConcept:** Categorías para clasificar gastos.
        i. **Relaciones:** Agrupa múltiples *Expense.*

    d. **Income:** Ingresos generados por personas, registrados bajo distintos conceptos y métodos de pago.
        i. **Relaciones:** Asociado con *Person, PaymentMethod, PaymentType, IncomeConcept, User.*

    e. **IncomeConcept:** Categorías para clasificar ingresos.
        i. **Relaciones:** Agrupa múltiples *Income.*

    f. **PaymentMethod:** Métodos de pago disponibles para transacciones.
        i. **Relaciones:** Usado en *Income* & Expense.

    g. **PaymentType:** Tipos de pago como contado, crédito, entre otros.
        i. **Relaciones:** Aplicado en *Income & Expense.*

    h. **Permission:** Permisos disponibles en el sistema para control de acceso.
        i. **Relaciones:** Aplicado en *RolePermission.*

    i. **Person:** Individuos con transacciones registradas en el sistema.
        i. **Relaciones:** Central para *Income & Expense.*

    j. **Role:** Roles de usuarios para definir niveles de acceso.
        i. **Relaciones:** Asignado a *User* y relacionado con *RolePermission.*

k. **User:** Usuarios del sistema con roles asignados.
    i. **Relaciones:** Relaciona roles y registra *Income* & Expense.

2. **Documentación de Servicios:**
    a. **PermissionService:** Búsqueda, creación y eliminación de servicios.
        i. **Métodos importantes:** *findAllPermissions(), savePermission(), deletePermission().*

    b. **RoleService:** Manejo de roles incluyendo creación y eliminación.
        i. **Métodos importantes:** *findAllRoles(), saveRole(), deleteRole().*

    c. **UserService:** Gestión de usuarios incluyendo registro, actualización y eliminación.
        i. **Métodos importantes:** *findAllUsers(), saveUser(), deleteUser().*

3. **Realizado:**
    a. **Se hizo la integración de JPA:** Configuración de las entidades del modelo, definición de las relaciones entre las entidades según las reglas del negocio.

    b. **Pruebas Unitarias:** Se realizaron las pruebas con JUnit con cobertura total de los métodos claves.

    c. **Carga de Esquema y Datos Iniciales:** Se crearon 3 scripts de SQL:
        i. *Para la carga del esquema inicial (schema.sql).*
        ii. *Para la carga de datos en la base de datos (data.sql).*
        iii. *Para la eliminación del esquema inicial (drop.sql).*

    d. **Integración de Backend:**
        i. **Creación de Repositorios:**
            1. **Implementación de Repositorios JPA:** Para cada entidad especificada en el modelo de datos (*DocumentType, Expense, Income, PaymentMethod, etc)* se crearon interfaces de repositorio usando Spring Data JPA.

        ii. **Creación de Servicios para Autenticación:** Se crearon servicios específicos para manejar la lógica de negocio asociada con las entidades de autenticación, que incluyen *User, Role, Permission.* Estos servicios encapsulan la autenticación necesaria para la administración de usuarios, roles y permisos, asegurando que las operaciones cumplan con las políticas de seguridad y acceso al sistema.

e. **Integración de herramientas para revisar cobertura:**
    i. **Instalación, configuración de JaCoCo:**
        1. JaCoCo se configuró en el archivo *build.gradle* del proyecto para ser utilizado con Gradle. Se añadió el plugin de JaCoCo y se configuraron las tareas para ejecutar el análisis de cobertura junto con las pruebas.

    ii. **Utilización de JaCoCo para generar reportes:**
        1. Con JaCoCo configurado, cada vez que se ejecutan las pruebas con Gradle usando *./gradlew test* JaCoCo recopila datos de cobertura de código. Al final de la ejecución de las pruebas, se genera un reporte detallado en el directorio especificado.

4. **Tareas pendientes:**
    a. Para la entrega actual, de acuerdo a la rúbrica proporcionada por el profesor, no hay ninguna tarea pendiente.

5. **Dificultades encontradas:**
    a. El cambio del dialecto para los scripts de H2.

6. **Reflexiones y conclusiones:**
    a. La integración de JaCoCo como herramienta para revisar la cobertura de pruebas ha sido crucial para mantener la calidad del código, la configuración y manejo de la base de datos mediante scripts SQL ha resaltado la importancia de diseñar scripts que sean idempotentes, es decir, que puedan ejecutarse múltiples veces sin causar errores o cambios no deseados, garantizando la estabilidad y la reproducibilidad de la configuración de la base de datos, facilitando las pruebas. La estructura del proyecto y la manera en que se han definido los servicios y repositorios muestran la importancia de la escalabilidad y el mantenimiento, ya que diseñar el sistema de manera modular y con una separación clara de responsabilidades no solo ayuda en la fase temprana del proyecto, sino que también para futuras expansiones y facilita la mantenibilidad a largo plazo.