Sistema para la Mejora del Acceso Alimentario en Contextos de Vulnerabilidad Socioeconómica

JUSTIFICACIONES DE DISEÑO

GRUPO 7
DISEÑO DE SISTEMAS

Diseño de Sistemas
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Colaboradores
Nuestro sistema contempla dos tipos de colaboradores: personas humanas y personas jurídicas. Ambas en tidades poseen comportamiento en común, por lo que decidimos diseñar una clase abstracta "Colaborador", con los atributos y comportamientos comunes, para luego crear dos clases concretas que hereden de ella: PersonaHumana y PersonaJurídica.

Tipos de Colaboraciones
Hasta el momento, los tipos de colaboraciones pueden distribuirse en tres grupos: aquellas que son exclusivas de personas humanas, aquellas que son exclusivas de personas jurídicas, y por último aquellas que pueden ser realizadas por ambos tipos de colaboradores.
Si bien todas tienen un comportamiento particular, al mismo tiempo deben responder a un diseño que las trate con homogeneidad. Para ello, decidimos diseñar una interfaz, "IContribuible", que expone el método "realizarContribución". Esta interfaz es implementada por tres otras interfaces, que buscan agrupar las contribuciones: "IContribuiblePersonaHumana", "IContribuiblePersonaJurídica", e "IContribuibleGeneral". Así, finalmente llegamos a las colaboraciones concretas, que diseñamos como clases que implementan las interfaces correspondientes (por ej, la clase "DonaciónDeDinero" implementa la interfaz "IContribuibleGeneral", ya que es un tipo de colaboración que pueden realizar ambos tipos de colaboradores.
Para tratar este polimorfismo desde el punto de vista de los Colaboradores, optamos por un camino de sobrecarga de métodos. Desde la clase abstracta "Colaborador" exponemos un método protegido "realizarContribución", que recibe como parámetro una contribución de tipo "IContribuibleGeneral". Asimismo, en las clases concretas "PersonaHumana" y "PersonaJurídica" exponemos los métodos "realizarContribución", que reciben como parámetro una contribución de tipo "IContribuiblePorPersonaHumana" e "IContribuiblePorPersonaJurídica, respectivamente". De esta forma, el método "realizarContribución" puede recibir como parámetro cualquier tipo de contribución, y a través de la sobrecarga se valida que el tipo corresponda con el tipo de colaborador que está queriendo realizarla.

Calculador de Puntos
El sistema debe incluir una forma de reconocer el trabajo de los colaboradores, asignándoles puntos según el trabajo que realicen. Estos puntos son luego utilizados en una fórmula para calcular el puntaje acumulado del colaborador correspondiente.
Para encarar este requisito, decidimos diseñar una clase "CalculadorDePuntos", que cuenta por el momento con cinco atributos, correspondientes a los coeficientes del polinomio lineal que se utiliza para el cálculo del puntaje:

- coeficientePesosDonados
- coeficienteViandasDistribuidas
- coeficienteViandasDonadas
- coeficienteTarjetasRepartidas
- coeficienteHeladerasRepartidas

La clase expone el método "calcularPuntos", que recibe como parámetro un colaborador, y dentro del mismo se desarrolla el cálculo del puntaje.
La razón por la cual el método debe recibir como parámetro a un colaborador es que se necesitan conocer características específicas de las donaciones hechas por ese colaborador (por ej, la cantidad de pesos que donó durante su actividad).
En la clase Colaborador agregamos como atributo un calculadorDePuntos, asociado a una instancia de CalculadorDePuntos.

Como el sistema debe permitir que los coeficientes puedan ser modificados mes a mes, nuestra solución da la posibilidad de utilizar dos caminos posibles: utilizar una instancia compartida de CalculadorDePuntos e ir modificando los valores atributos periódicamente, o instanciar distintas clases con los valores de atributos correspondientes, permitiendo que en el futuro se pueda asignar un cálculo distinto según el colaborador.

Direcciones geográficas
En varias entidades necesitamos registrar direcciones geográficas (principalmente domicilios). Por una cuestión de consistencia de datos, decidimos modelar esto con una clase "Dirección", cuyos atributos son "calle", "altura", "códigoPostal" y "localidad".

Colocación de Heladeras: consumo de API REST
Para la colaboración de "Colocación de Heladeras", es necesario consumir los servicios de una API externa (que aún se encuentra en desarrollo). El funcionamiento será el siguiente: desde el sistema se ingresará un punto geográfico (latitud & longitud) y un radio, a lo cual la API responderá con sugerencia de puntos cercanos donde se podrá instalar una heladera.
Para implementar este requisito, recurrimos al patrón de diseño Adapter, ya que la API es externa a nuestro sistema (y además aún no está completa). Dentro de la clase "ColocaciónDeHeladeras" agregamos un atributo "adapterUbicacionSugerida".

Sensor de Temperatura y Movimiento: consumo de API
Las heladeras cuentan con un sensor de temperatura, que permite registrar la temperatura actual de la misma y sirve para alertar cuando la temperatura se excede de los límites aceptables, y un sensor de movimiento, que emite una alerta ante intentos de robo.
Este requerimiento lo implementamos utilizando el patrón Adapter, al igual que en el caso de la Colocación de Heladeras. Consideramos que la lógica detrás de los sensores será desarrollada (si es que ya no existe) por un equipo externo, y desde nuestro sistema nos acoplaremos utilizando un adapter.

Carga Masiva
El sistema permite la carga de archivos csv, para gestionar en forma masiva colaboraciones. Para ello, utilizamos el paquete "java.io", que proporciona las herramientas necesarias para manejar la entrada y salida (I/O) de datos, ya sea desde archivos, dispositivos de red, flujos de datos, entre otros.

Validador de Contraseñas
En primer lugar, nuestro sistema tiene que verificar que las contraseñas que deseen crear los usuarios coincidan con los estándares de seguridad establecidos. Uno de ellos es que la contraseña no puede estar dentro de un listado de las 10.000 contraseñas más débiles. Para ello, tenemos un archivo txt con dichas contraseñas, que luego gestionamos utilizando el paquete "java.io".
Además, implementamos una encriptación y hasheo de contraseñas, utilizando una dependencia (jBCrypt), para alcanzar los estándares de seguridad requeridos.

Utilización de Mapas
Necesitamos contar con un mapa en donde se pueda ver la ubicación de las heladeras instaladas. Para ello, utilizamos una biblioteca de JavaScript (Leaflet) que permite la la implementación de mapas interactivos. La misma, junto con la utilización de HTML y CSS, nos permite colocar puntos (simbolizando las heladeras) en un mapa interactivo que es mostrado en el frontend de la aplicación.

Enviador de mails
Para implementar un enviador de mails, recurrimos a una biblioteca de Java, Jakarta Mail, que proporciona una API para la gestión de correo electrónico. Esta API nos permite enviar, recibir y manipular mensajes de correo electrónico.