Práctica de Internet y Sistemas Distribuidos

3º Curso – Grado en Informática
Curso académico 2017-2018

1 Introducción

La práctica consistirá en la aplicación de los conceptos y tecnologías aprendidos en la asignatura para el desarrollo de un servicio simplificado de transporte privado (al estilo de aplicaciones reales como Uber o Cabify).

La aplicación seguirá una arquitectura en capas como la estudiada en la asignatura, incluyendo capa de acceso a datos, capa de lógica de negocio y capa de servicios. La aplicación podrá ser invocada remotamente usando REST y (opcionalmente) SOAP. La capa de acceso a datos utilizará una base de datos relacional para guardar información de los conductores y los viajes realizados.

Además, de manera opcional, la aplicación se integrará con la API de Facebook para permitir a los usuarios del servicio compartir reseñas sobre los conductores.

2 Planteamiento

2.1 Visión global

La empresa MiniUber decide ofrecer una plataforma de transporte privado para poner en contacto a conductores y a usuarios.

Los conductores podrán darse de alta en el servicio, modificar sus datos y ver la lista de sus viajes pasados.

Los usuarios podrán buscar conductores por distintos criterios, contratar un viaje, puntuar un viaje y ver datos de sus viajes previos.

Para garantizar que los conductores deshonestos o incumplidores son detectados rápidamente, MiniUber dispone de dos mecanismos: 1) Al buscar conductores, el usuario puede ver la puntuación media de sus viajes, 2) Si la puntuación media de un conductor pasa a ser menor

que 5 (sobre 10), MiniUber marca al conductor como ‘inactivo’ y publica automáticamente un aviso en su página de Facebook. El mecanismo por el cuál un conductor puede volver al estado activo queda fuera del alcance de la práctica

La implementación de la integración con Facebook es opcional. En clase de prácticas, se describirán los fundamentos básicos de uso de la API de Facebook

2.2 Funcionalidad de la Capa Modelo

En esta sección se proporciona más información sobre la funcionalidad que debe soportar la capa modelo.

1. Para darse de alta, el conductor proporcionará la siguiente información: nombre, ciudad, modelo de coche, hora (entre 0 y 23) a partir de la que a acepta viajes y hora de fin (entre 0 y 23). Debe validarse que las horas introducidas son válidas. Se registrará también la fecha de alta del conductor.

2. Es posible modificar la ciudad, modelo de coche y horas de inicio y fin de un conductor. No es posible modificar su nombre.

3. También se pueden obtener todos los datos de un conductor a partir de su identificador.

4. Los usuarios podrán ver qué conductores hay disponibles en una ciudad determinada. Los conductores disponibles son los que están en estado ‘activo’ y además aceptan viajes en la hora actual. Además de los datos ya mencionados del conductor, los usuarios verán la puntuación media que se le otorgó por viajes previos.

5. El usuario podrá contratar un viaje con un conductor disponible. Para ello, se indicará id de conductor, dirección de origen, dirección de destino, login de usuario y número de tarjeta de crédito. Se validará que el número de tarjeta de crédito es correcto (usando el mismo método que en el ejemplo ws-movies). El viaje se registrará en la BD de MiniUber, indicando también la fecha de la reserva.

6. Una vez terminado un viaje, el usuario podrá puntuar su experiencia. Para ello indicará el id del viaje, y una puntuación entre 0 y 10. Si la puntuación media del conductor pasa a ser menor que 5, será marcado como inactivo. Si se realiza la parte opcional de integración con Facebook, se publicará además de forma automática un post en la página de Facebook de MiniUber. El post tendrá el texto: “AVISO IMPORTANTE: El conductor <nombre del conductor> ha pasado a estado inactivo”.

7. Será posible obtener los datos de todos los viajes que ha realizado un conductor, así como de todos los viajes que ha realizado un usuario.

NOTA IMPORTANTE: Para simplificar la implementación de la práctica, NO es necesario guardar datos de los usuarios del servicio. Sólo se guardará su login al registrar un viaje.

2.3 Detalles de la Capa de Servicios

La capa de servicios expondrá la funcionalidad de la capa modelo a aplicaciones remotas usando un servicio web REST.

La información de conductores expuesta por la capa de Servicios **no** incluirá la fecha de alta de los mismos. Esta información es usada exclusivamente para informes internos que se generan accediendo directamente a la capa modelo y, por lo tanto, no se expone a las aplicaciones remotas.

Además, los desarrolladores de la capa de servicios (que no tienen acceso al código fuente de la capa modelo) han detectado que hay aplicaciones cliente que desean obtener datos sobre los conductores que han realizado viajes para un usuario determinado. Por tanto, decidieron ofrecer esta funcionalidad ya implementada en la capa de servicios, evitando así que cada aplicación cliente tenga que implementarla por su cuenta, y optimizando así también el rendimiento de la operación. Para cada conductor que haya hecho algún viaje con el usuario, se mostrará su nombre, ciudad, y la puntuación media que el usuario otorgó a los viajes que hizo con él.

Opcionalmente, la capa de Servicios se implementará también usando un servicio SOAP. En ese caso, sólo será necesario implementar los casos de uso que permiten al conductor darse de alta, ver sus datos, modificarlos (sólo los que es posible modificar), y ver la lista de sus viajes.

2.4 Detalles de la Aplicación Cliente

Es necesario desarrollar dos tipos de aplicación cliente:

- Una aplicación cliente para conductores. Esta aplicación debe permitir al conductor darse de alta, ver sus datos, modificarlos (sólo los que es posible modificar), y ver la lista de sus viajes. El conductor no puede ver el número de tarjeta de crédito utilizado por el viajero en los viajes.
- Una aplicación cliente para los usuarios. Esta aplicación debe permitir buscar conductores, contratar un viaje y puntuarlo. También debe permitir ver los viajes que realizó el usuario y los conductores que le han llevado alguna vez (usando para ello la información de la nueva operación añadida en la capa de servicios).

Los alumnos deben de realizar la capa de acceso al servicio de ambos clientes usando REST. Adicionalmente, los alumnos que realicen la parte opcional de SOAP, deberán implementar la capa de acceso al servicio del cliente para conductores usando dicha tecnología.

Los clientes podrán cambiar de la versión REST a la versión SOAP del servicio modificando simplemente un parámetro de configuración. Incluso si no se implementa la versión SOAP, la arquitectura de la práctica deberá contemplar la posibilidad de que se desarrollase en un futuro para uno o ambos clientes.

3 Normativa y evaluación

3.1 Composición de los grupos

La práctica se realizará en grupos de 3 personas (preferentemente) o de 2 personas.

3.2 Estándar de codificación

Con objeto de escribir código de calidad y fácilmente legible, se seguirá un sistema de codificación común, que define reglas para nombrar clases, atributos y métodos, normas de indentación, etc. Esto permite que en un equipo de desarrollo el aspecto del código sea el mismo, independientemente de qué programador lo haya escrito, lo que facilita el mantenimiento. Para la práctica se utilizará el estándar de codificación [JAVACON]. Los ejemplos de la asignatura siguen estas sencillas convenciones de nombrado. Para no alargar la práctica, no será necesario realizar documentación de las clases (e.g. JavaDoc).

3.3 Formato de entrega de la versión final

Para la entrega de cada iteración de la práctica, se utilizará el repositorio subversion [SVN]. En la entrega de la versión final se presentará además una memoria impresa. Se debe subir al repositorio subversion **sólo los ficheros fuente** (e.g. .java, pom.xml, ficheros de configuración, etc.), **y no los ficheros objeto** (e.g. .class, .war, etc.).

A continuación se detalla el formato de la memoria. Como norma general, los diagramas emplearán la notación UML. **Los diagramas deben ser de calidad** (y no hechos por reingeniería inversa; ¡el diseño precede a la implementación!) y estar explicados de manera breve, pero clara. Es importante destacar que no se pretende que se haga un documento grande, sino un documento que explique breve, pero claramente, cada uno de los apartados que a continuación se describen.

1. Introducción

Debe indicar si se ha hecho la parte opcional.

2. Capa modelo

2.1. Entidades

*Incluye un diagrama UML que muestra el diseño de las entidades. Para cada entidad **sólo** se deben mostrar sus atributos privados (no se mostrarán los métodos get/set). En el texto del apartado, se deben comentar/justificar los detalles que se consideren relevantes.*

2.2. DAOs

*Incluye un diagrama UML que muestra **únicamente** las interfaces de los DAOs. Cada interfaz debe especificar la firma completa de sus operaciones. No es necesario especificar las excepciones, ni incluir los diagramas UML de las mismas. En el texto del apartado, se deben comentar/justificar los detalles que se consideren relevantes.*

2.3. Fachadas

*Incluye un diagrama UML que muestra **únicamente** las interfaces de las fachadas de la capa modelo. Cada interfaz debe especificar la firma completa de sus operaciones. No es necesario especificar las excepciones, ni incluir los diagramas UML de las mismas. En el texto del apartado, se deben comentar/justificar los detalles que se consideren relevantes.*

3. Capa de servicios

3.1. DTOs

*Incluye un diagrama UML que muestra el diseño de los DTOs que utiliza el servicio para recibir y devolver datos. Para cada DTO **sólo** se deben mostrar sus atributos privados (no se mostrarán los métodos get/set).*
En el texto del apartado, se deben comentar/justificar los detalles que se consideren relevantes.

3.2. REST

Se especifica cómo invocar cada caso de uso que ofrece el servicio, indicando para cada uno:

- *URL simplificada del recurso (e.g. /product/{id}, /products)*
- *Método de invocación (GET, POST, PUT o DELETE)*
- *Parámetros o DTO de entrada (si aplicable)*
- *DTO devuelto (si aplicable)*

En el texto del apartado, se deben comentar/justificar los detalles que se consideren relevantes. En particular, debe explicarse el criterio seguido para escoger los códigos HTTP de respuesta asociados a cada excepción.

3.3. SOAP

*Incluye un diagrama UML que muestra **únicamente** las clases de implementación de los servicios SOAP. Cada clase debe especificar la firma completa de sus operaciones. No es necesario especificar las excepciones, ni incluir los diagramas UML de las mismas. En el texto del apartado, se deben comentar/justificar los detalles que se consideren relevantes.*

4. Aplicaciones cliente

4.1. DTOs

*Incluye un diagrama UML que muestra el diseño de los DTOs que se utilizan en las capas de acceso al servicio de las aplicaciones cliente. Para cada DTO **sólo** se deben mostrar sus atributos privados (no se mostrarán los métodos get/set).*
En el texto del apartado, se comentan/justifican los detalles que se consideren relevantes.

4.2. Aplicación cliente de conductores: capa de acceso al servicio

*Incluye un diagrama UML que muestra **únicamente** las interfaces de la capa de acceso al servicio que utiliza esta aplicación. Cada interfaz debe especificar la firma completa de sus operaciones. No es necesario especificar las excepciones, ni incluir los diagramas UML de las mismas. En el texto del apartado, se deben comentar/justificar los detalles que se consideren relevantes.*

4.3. Aplicación cliente de usuarios: capa de acceso al servicio

*Incluye un diagrama UML que muestra **únicamente** las interfaces de la capa de acceso al servicio que utiliza esta aplicación. Cada interfaz debe especificar la firma completa de sus operaciones. No es necesario especificar las excepciones, ni incluir los diagramas UML de las mismas. En el texto del apartado, se deben comentar/justificar los detalles que se consideren relevantes.*

5. Parte opcional

*Se explica el diseño de la parte opcional, apoyándose en diagramas UML que muestren **únicamente** los detalles relevantes. Se valorará el diseño (por ejemplo: la utilización de patrones como factoría) y la implementación*

6. Errores conocidos

Se indican los errores que se conoce que tiene el código.

3.4 Iteraciones y entregas

Para la realización de cada aplicación se seguirá un enfoque basado en iteraciones, de manera que cada iteración incorpora más funcionalidad sobre la anterior, hasta que en la última iteración se termina con un software que implementa toda la funcionalidad.

En particular, la aplicación se hará en tres iteraciones. Es **obligatorio** entregar las dos primeras iteraciones en plazo y con los contenidos definidos para cada una de ellas totalmente implementados. En otro caso, la práctica será calificada con un cero. La entrega de la tercera iteración es opcional.

En la entrega de cada iteración deben estar presentes todos los miembros del grupo. En la entrega se realizará una demo y se harán preguntas individualizadas sobre el diseño y la implementación.

A pesar de ser obligatoria su entrega, la corrección de la primera iteración no llevará una nota asociada ni será necesario entregar una memoria. Bastará con mostrar los diagramas relativos a esta iteración realizados con cualquier herramienta de modelado UML. El objetivo de la corrección de esta iteración es detectar errores importantes, y en ese caso, orientar al alumno hacia su resolución.

En la segunda iteración se completará la parte obligatoria de la práctica. En la corrección de la segunda iteración se pondrá una nota y se deberá entregar la memoria. La nota puede ser diferente para cada miembro del grupo en función de sus respuestas a las preguntas personalizadas.

La tercera iteración es opcional. Los alumnos que decidan entregarla tendrán que implementar al menos una de las partes opcionales: el servicio web SOAP y/o la integración con Facebook. Además, siempre que hayan entregado al menos una de las partes opcionales, podrán corregir los errores detectados en la segunda iteración para mejorar su nota. Entregarán una memoria actualizada con los cambios realizados durante la iteración.

A continuación se describe cada una de las tres iteraciones:

- **Primera iteración**. Se implementará como mínimo la parte de la capa modelo necesaria para las funcionalidades de dar de alta conductores, modificar sus datos, buscar conductor por identificador, buscar conductores disponibles y contratar un viaje. Se **recomienda fuertemente implementar también en esta iteración el resto de casos de uso de la capa modelo**, pero no es obligatorio. **Plazo de entrega: domingo 12 de Noviembre.** La defensa de esta iteración se realizará en horas de prácticas durante la semana del 13 al 17 de Noviembre.
- **Segunda iteración**. Se completará la capa modelo y se implementará obligatoriamente el servicio web REST que permite el acceso remoto a la capa modelo, así como los

clientes de línea de comandos necesarios para acceder al mismo. Los alumnos que lo deseen pueden presentar también una o ambas de las partes opcionales de la práctica. **Plazo de entrega: domingo 17 de Diciembre.** La defensa de esta iteración se realizará en horas de prácticas durante la semana del 18 al 22 de Diciembre.

- **Tercera iteración**. Se presentarán una o ambas partes opcionales. Si se presenta al menos una de las partes opcionales, será posible presentar también correcciones de los problemas detectados en la segunda iteración. **Plazo de entrega: 28 de Enero**. Los alumnos pueden defender esta iteración en cualquier momento hasta el 2 de Febrero (inclusive). Las fechas y horas de defensa se concertarán previamente con los profesores.

3.5 Evaluación

La puntuación máxima de cada parte será como sigue:

- Parte obligatoria: 8 puntos.
- Parte opcional servicio SOAP: 1 punto.
- Parte opcional de integración con Facebook: 1 punto

Para la puntuación de cada parte, se tendrá en cuenta:

- Su correcto funcionamiento.
- La calidad del diseño.
- La calidad del código.
- La calidad de la memoria.

Una práctica copiada significará un suspenso para el grupo que ha dejado copiar y el que ha copiado; a todos los efectos, no se hará ninguna distinción. Los suspensos por práctica copiada tendrán que realizar una práctica distinta, que además deberán proponer (y ser aceptada).

Si alguno de los miembros de un grupo no supera la defensa de la segunda o la tercera iteración pero el resto sí lo hace, el alumno suspenso deberá defender en solitario en Julio una versión extendida de la práctica con casos de uso adicionales.

Para la convocatoria extraordinaria de Julio se hará la misma práctica (excepto los suspensos en alguno de los dos casos anteriores), sin posibilidad de entregar la primera iteración: se presentará directamente la segunda iteración (o la tercera, si se decide hacer también las partes opcionales).

4 Referencias

[JAVACON] Sun Microsystems, “Java Code Conventions”, http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

[SVN] Subversion: http://subversion.apache.org/.
