Desarrollo del back-end para un entorno ubicuo de enseñanza

Brachetta, M.; León, O.; Monetti, J.

Dpto. Ing. en Sistemas, Facultad Regional Mendoza – Universidad Tecnológica Nacional

{mariana.brachetta;oscar.león;julio.monetti}@docentes.frm.utn.edu.ar

Resumen

Se presentan las características del back-end diseñado para la implementación de un entorno ubicuo de enseñanza y aprendizaje, el cual integra tres componentes de software: aplicación nativa Android que permite geolocalizar al estudiante; acceso a Moodle donde se presentan las actividades y evaluaciones a resolver para validar aprendizajes, y en tercer lugar, una solución cloud computing, encargada de resolver la lógica de negocio del entorno, administrar el modelo y persistencia de datos, gestionar la integración con Moodle y ofrecer al usuario los servicios de la nube mediante una aplicación móvil. El trabajo se enfoca en discutir la arquitectura del back-end y los webservices desarrollados para dar soporte a la recreación de un ambiente de trabajo ubicuo.

Palabras clave: enseñanza ubicua, computación móvil, servicios de la nube

Introducción

El entorno ubicuo propuesto (León et al, 2019) (León et al, 2017), permite a un estudiante transitar un recorrido de aprendizaje a través de una secuencia de actividades que debe abordar de forma autónoma. Estas actividades ~~pueden ser~~ son recursos como contenido hipermedia (documentos, videos, gráficos, páginas web, actividades interactivas, etc.). La información del recorrido ejecutado por el estudiante, es almacenada en la base de datos (BD). Las evaluaciones están implementadas en la plataforma Moodle y las orientaciones de ayuda construidas con tecnología de Realidad Aumentada (RA).

El recorrido de aprendizaje se vincula a un recorrido geográfico real, compuesto de tantos puntos o ubicaciones de interés como unidades temáticas incluye la secuencia. Estos puntos o ubicaciones geográficas son personalizadas por cada estudiante, al iniciar su trayectoria (por ejemplo, si habitualmente debe esperar un ómnibus, podría elegir ese lugar geográfico para que se active una tarea a resolver). La Ilustración 1 muestra la organización del entorno ubicuo.

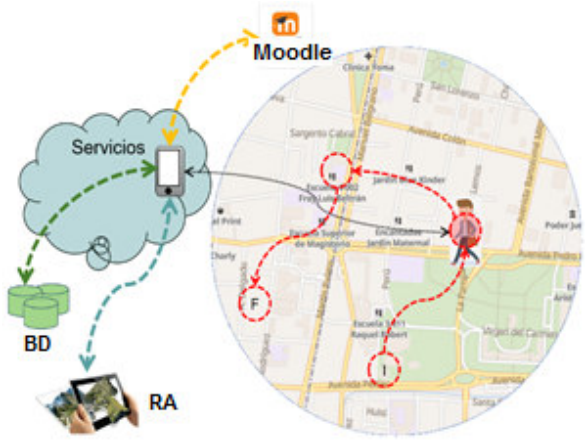

Ilustración 1: Esquema del entorno ubicuo

La estructura del entorno ubicuo propuesto (Durán, 2014), implementa un esquema para que al transitar en las proximidades de puntos

geográficos registrados, se active una alerta que informe al estudiante que tiene una actividad para resolver. Al finalizar la actividad, se abre, en el navegador embebido en la App, la evaluación correspondiente y en caso de superar la instancia, se activa el próximo lugar geográfico del recorrido.

Aspectos a resolver

- Permitir a los estudiantes registrarse y seleccionar ubicaciones geográficas donde realizar sus actividades.
- El registro debe permitir interactuar con el entorno a través de la App móvil, sin que se requiera salir de la misma para acceder a otros componentes (Moodle, RA, etc.).
- Se debe descargar la complejidad de funcionamiento, en los servicios cloud, posibilitando flexibilidad, portabilidad y escalabilidad.
- En un dispositivo móvil debe poder "loguearse" más de un usuario, manteniendo la trazabilidad de lo ejecutado por cada usuario del dispositivo.
- Un estudiante puede "loguearse" en diversos dispositivos, pero sólo podrá estar "logueado" en uno solo a la vez.
- Los valores constantes del entorno, tales como número de zonas de geo-vallado a utilizar, direcciones url, mensajes al usuario, códigos de error, etc. deben ser parametrizables.
- Cada unidad temática en la secuencia de aprendizaje se vincula dinámicamente a una zona de interés geográfica según cómo se vayan transitando las áreas de geo-vallado.
- Se debe integrar un navegador web que permita acceder a los recursos hipermedia provistos. El navegador debe estar embebido en la App para posibilitar que el entorno tenga retroalimentación de las acciones realizadas por el usuario.
- El estudiante debe tener retroalimentación respecto de las unidades temáticas que transitó, con identificación de las aprobadas, las no aprobadas y las aún no cursadas. Cada unidad temática completada debe poder ubicarse en un mapa, vinculada al área geográfica donde la misma fue resuelta.

Descripción del desarrollo

El entorno tiene una arquitectura (Ilustración 1) cliente-servidor, en la que el back-end se implementa como plataforma de software como servicios (SAS) y ofrece un conjunto de servicios web, del tipo RESTful, que son consumidos por una aplicación front-end implementada como una App nativa Android.

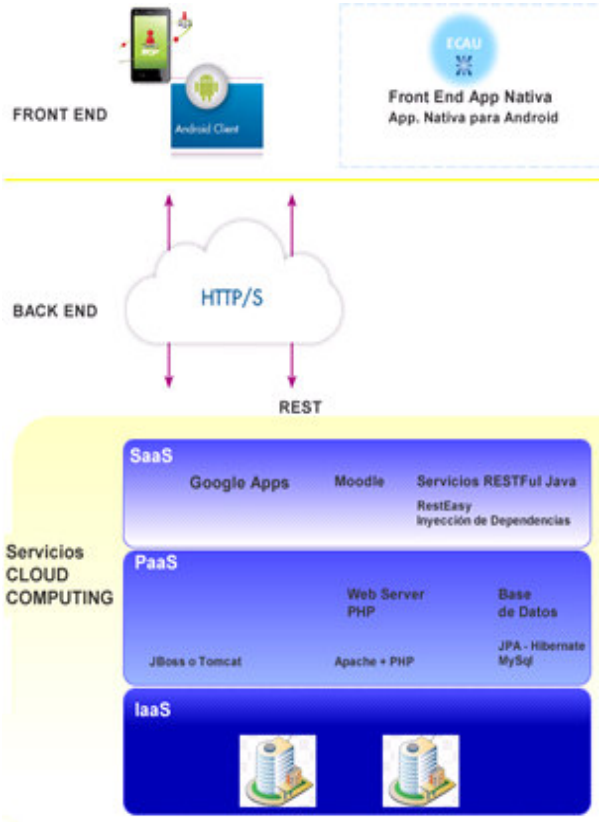

Ilustración 1: Arquitectura general

Arquitectura

La aplicación que presta servicios del lado del servidor (SAS), se despliega sobre un servidor de aplicaciones Java implementado en la capa de PAS (Plataforma como servicios). Aunque se usó Wildfly 10.0, el desarrollo es portable y puede desplegarse en otros applications servers como JBoss, Tomcat, GlassFish, WebLogic, etc. Se sigue una arquitectura de capas (Rosado da Cruz & Paiva, 2016) como muestra la Ilustración 3.

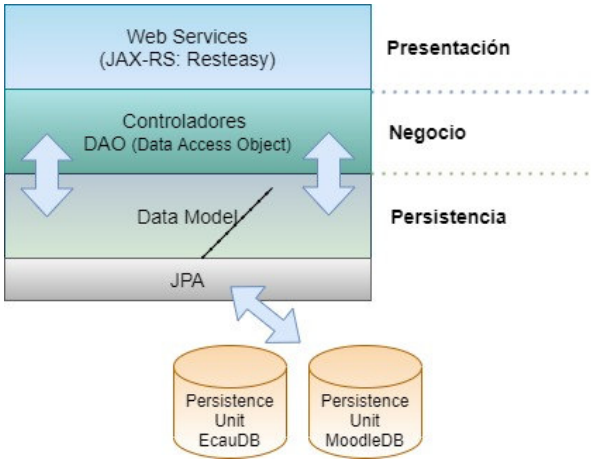

Ilustración 3: Esquema de capas

Capa de Presentación: constituye la interfase del back-end con el front-end. Allí se ubican los servicios web que el servidor brinda a la App Nativa Android. Estos servicios son un conjunto de recursos disponibles a través de protocolo HTTP que reciben peticiones de los usuarios (vía clientes HTTP-REST), las procesan y envían una respuesta. Los servicios web, fueron desarrollados usando Resteasy, debido a su simplicidad de configuración y programación, además de su integración directa con Wildfly.

La App nativa Android intercambia información con los servicios web mediante serialización de objetos con JSON (JavaScript Object Notation).

No hay capa web o de vista en el desarrollo del back-end, ya que solo se accede a él a través de los servicios web implementados en la capa de presentación.

Capa de Negocio: incluye las clases de control del modelo Orientado a Objetos (OO) que gestiona los procesos del entorno. Los servicios web en la capa de presentación hacen uso de los objetos definidos en este nivel. A su vez, a través de los objetos de acceso a datos DAO (Data Access Object) definidos en esta capa, se vincula a la capa de persistencia mediante JPA (Java Persistence API). El uso de JPA para acceder a la base de datos relacional automatiza las conversiones de datos necesarias para realizar transacciones, y muestra ser una tecnología que se escala bien conforme una base de datos crece y se vuelve más compleja

Capa de Persistencia: El modelo de datos que soporta la aplicación se almacena en una BD relacional. La conversión del modelo orientado a objetos implementado en la capa de negocios, a un modelo de datos relacional es resuelto por JPA/Hibernate (Hibernate podría reemplazarse por otro en caso de ser necesario, tal como Eclipselink, ObjectDB, Toplink, OpenJPA, etc.). Sin embargo, un limitante de trabajar con JPA como capa superior a Hibernate, es que la especificación cubre solo la conexión con BD relacionales, mientras que Hibernate es más amplio y podría usarse contra bases de datos NoSQL. No obstante, en el contexto del desarrollo del entorno, se decidió priorizar la mayor independencia entre capas.

El motor de BD utilizado puede reemplazarse por otros (PostgreSQL, Oracle Databases). Se configuran dos data sources (fuentes de datos) en el servidor Wildfly: una vincula a la BD del proyecto (EcauDB), la segunda vincula a la BD de Moodle. Luego la aplicación usa estas

fuentes de datos para conectar a la BD mediante Hibernate.

Estructura del proyecto

El código fuente se estructura en cuatro paquetes Java:

service: contiene las clases donde se implementan los servicios web del entorno, que ponen a disposición de la App móvil las funcionalidades de procesamiento de información resueltas en el back-end.

controller: contiene la interfaces y clases que implementan los objetos de acceso a datos DAO. Estos son los beans de sesión que definen:

- A nivel de interfase (capa superior): las operaciones que la lógica de negocios del entorno requiere sobre sus datos, esto permite concentrarse exclusivamente en las operaciones requeridas, con independencia de la tecnología de persistencia subyacente.
- A nivel de clases (capa intermedia): la implementación mediante JPA de las operaciones anteriormente definidas para cada objeto de acceso a datos - DAO. JPA, resuelve de manera transparente la implementación concreta del modelo de persistencia para una tecnología de base puntual (en este caso Hibernate sobre MariaDB). Los querys implementados en los beans de sesión o DAO se realizan en lenguaje JPQL.
- Model: contiene las clases de entidad que modelan los objetos persistentes del modelo y sus relaciones. Estas clases definen las entidades que serán luego persistidas a nivel de BD, llamadas beans de entidad. El back-end del entorno, es independiente del motor de persistencia y de la BD con que se implemente el almacenamiento en las capas subyacentes. En este paquete, se incluyen aquellas clases que modelan los objetos JSON que sirven al intercambio de información mediante web services entre el front-end y el back-end.
- Util: contiene clases utilitarias que cooperan con otras del modelo, tales como la clase "Configuración.java" que define el acceso al archivo XML de propiedades y parámetros del entorno, el acceso al log de transacciones, etc.

A lo anterior se agregan los archivos de configuración:

- config.properties, que contiene pares de propiedades y valores utilizadas para parametrizar la compilación del proyecto.
- persistence.xml, que define las unidades de persistencia utilizadas en el modelo (persistence units), que establecen la vinculación entre un conjunto de clases de entidad en el modelo OO y una fuente u origen de datos en particular (datasource). Un cambio en la arquitectura de base (por ej.: cambiar el motor MariaDB por PostgreSQL) solo requiere redefinir la fuente de datos en el servidor de aplicaciones Java y modificar su vinculación en este documento. También se puede escalar agregando nuevas bases de datos, con solo agregar unidades de persistencia en el proyecto y datasources en el applicaton server.

Librerías no incluidas de forma nativa en Java EE, usadas para el desarrollo del back-end:

- Framework Resteasy para el desarrollo de web services.
- Google GSon utilizada para serializar y deserializar cadenas JSON.

- Función de hashing Bcrypt, se utiliza en la integración con Moodle para verificar que la clave provista por el usuario coincida con la clave cifrada en la BD de Moodle.
- Cliente Http para realizar http request, que se usa para la integración con Moodle.

Servicios desarrollados

WS01

Validación y registro de estudiantes en la aplicación. Cuando un estudiante descarga, instala y ejecuta por primera vez la aplicación móvil, debe registrarse con el mismo usuario y contraseña con los que previamente se registra en la plataforma Moodle. El WS01, tiene por objeto validar que el usuario esté previamente registrado en Moodle y persistir sus datos en la BD central del entorno.

WS02

Registro de coordenadas GPS (Tin Megali, 2016). Cuando el estudiante se registra por primera vez en el entorno, selecciona en un mapa un conjunto de coordenadas GPS. Este servicio, es el encargado de registrar en el entorno las coordenadas seleccionadas por el estudiante, asignando un orden y estado a cada una, así luego puede tener la trazabilidad de las actividades realizadas.

El registro de coordenadas se realiza solo una vez, es decir que si vuelve hacer el "loguin" desde un dispositivo distinto, se descarga desde el back-end las coordenadas previamente registradas para guardarlas en su BD local (Umang Burman, 2018).

WS03

Consulta coordenadas registradas, para lo cual recibe como parámetro el identificador único de cada estudiante y luego requiere a la capa de negocios (mediante el objeto de acceso a datos AlumnoDAO), que devuelva la lista de coordenadas GPS registradas para ese usuario.

WS04

Consulta de próxima actividad en el recorrido. Se encarga de obtener la próxima actividad a desarrollar por el estudiante en su recorrido por la secuencia de aprendizaje. Dicha actividad puede ser del tipo Recurso, Test o Pista según corresponda. Es el servicio más complejo en su implementación y el que más interacciones produce con la capa de negocios por cuánto deben validarse muchas reglas. Por ejemplo consultar el estado del recorrido previo del alumno; determinar la próxima actividad que le corresponde realizar a un estudiante; recuperar los recursos necesarios para entregar al estudiante una actividad, habilitar el acceso a un test (interactuando con la BD de Moodle) o activar un objeto de RA.

WS05

Registro de actividad completada o consumida. Cuando un estudiante completa alguna actividad, este servicio es el encargado de registrar tal situación. Por ejemplo, controla si se completó un test de Moodle o no, y el resultado obtenido.

Transacciones

En el archivo de configuración config.properties se parametriza la ruta que apunta a un archivo de log del back-end. En este archivo se registra toda la información de operaciones en el back-end, con fecha/hora en que se producen. También se almacena la traza de error en caso de que se lancen excepciones.

Conclusiones

La App consume pocos recursos del dispositivo móvil, dado que el procesamiento más complejo, se descarga en el back-end.

La solución MCC (Mobile Cloud Computing) que integra una App móvil delgada, con un conjunto de servicios cloud computing posibilitó crear un ambiente de trabajo ubicuo, en el cual el estudiante puede determinar en qué lugares y horarios realizar las diferentes actividades involucradas en el proceso de aprendizaje de una temática.

El enfoque por capas adoptado para el desarrollo del back-end, en el cual se descargan las tareas más complejas que permiten recrear el ambiente ubicuo, ha posibilitado definir una estructura flexible, fácilmente configurable y sencilla de mantener.

La combinación de un modelo de persistencia local en el front-end (App), junto con el repositorio en el back-end, garantiza que los usuarios no pierdan eventualmente los datos locales en el dispositivo móvil, además de posibilitar que la aplicación continúe funcionando cuando la red es inestable o no está disponible y evitar realizar permanentes accesos al repositorio remoto vía webservices.

Agradecimiento

El artículo es resultado del proyecto PID UTN4741: "Desarrollo de un entorno basado en Cloud Computing para Aprendizaje Ubicuo" el cual es financiado por la FRM - UTN, Mendoza- Argentina.

Bibliografía

Durán, B. Á. (2014). Ontological model-driven architecture for ubiquitous learning applications. In Proceedings of the 7th Euro American Conference on Telematics and Information Systems (pág. 14). ACM.

Google Developers (2021). Guía de arquitectura de apps. Recuperado de https://developer.android.com/jetpack/guide?hl=es-419

Google Developers (2021). Cómo acceder a la ubicación en segundo plano. Recuperado de https://developer.android.com/training/location/background?hl=es

León, O.; Schilardi, A.; Monetti, J.; Brachetta, M. (2019) Entorno ubicuo de enseñanza y formación por competencias. XIV Congreso Nacional de Tecnología en Educación y Educación en Tecnología (TE&ET). Facultad de Ciencias Físico Matemáticas y Naturales (FCFMyN) de la UNSL

León, O.; Brachetta, M.; Monetti, J. (2017). Desarrollo de un Entorno de Aprendizaje basado en Ulearning. TE&ET. Libro de Actas XII Congreso de Tecnología en Educación y Educación en Tecnología. REDUNCI, 1ª. Ed. San Justo: Universidad Nacional de La Matanza

Mohindra, A. (2015). ACM Tech Pack on Cloud Computing: IBM Research Division. Thomas J. Watson Research Center Chair, ACM Tech Pack Committee on Cloud Computing. Recuperado el 7 de octubre de 2017, en https://techpack.acm.org/cloud/cloudcomputing.pdf

Rosado da Cruz, A., Paiva S. (2016). Modern software engineering methodologies for mobile and cloud environments. USA. Information Science Reference (and imprint of IGI Global). ISBN-10: 1466699167. ISBN-13: 978-1466699168.

Tin Megali, (2016). How to work with geofences on Android. Recuperado de https://code.tutsplus.com/tutorials/how-to-work-with-geofences-on-android--cms-26639?ec_unit=translation-info-language

Umang Burman, (2018). Login example with MVVM, DataBinding with LiveData. Recuperado de https://medium.com/@umang.burman.micro/ login-example-with-mvvm-databinding-with-livedata-81319048afb0