Solución cloud para entorno de aprendizaje ubicuo

Mariana I. Brachetta [1[0000-0003-3388-0326]] and Oscar León[1[0000-0002-5977-9528]]

[1] Universidad Tecnológica Nacional - Facultad Regional Mendoza, Rodríguez 273, M5502AJE, Mendoza, Argentina

Abstract. Se presenta una solución de aprendizaje ubicuo, donde se integran tres componentes de software: una aplicación nativa Android que permite geolocalizar al alumno y ofrecerle experiencias de aprendizaje en el lugar en que se encuentra; la plataforma donde se encuentra el material de aprendizaje en línea (plataforma Moodle); y una aplicación en la nube encargada de resolver la lógica de negocio del entorno, administrar el modelo y persistencia de datos, gestionar la integración con Moodle y ofrecer los servicios del entorno a ser usados por la App nativa Android. El artículo aborda la construcción del tercer componente de software en el entorno -la aplicación cloud, montada como una solución SaaS (software como servicio) en la nube (back-end del entorno). Inicialmente se describe la especificación establecida para el entorno, luego se comentan los principales problemas que debieron resolverse durante la construcción de la solución y por ultimo las alternativas de abordaje que se contemplaron para su desarrollo. Finalmente se comenta brevemente la estructura del proyecto.

Keywords: cloud computing, aprendizaje ubicuo, SasS, REST.

1 Introducción

La aplicación desarrollada utiliza servicios de la “nube” para implementar el siguiente modelo (Fig. 1) de un ambiente ubicuo para el aprendizaje, al tiempo que se intenta recrear un entorno lúdico semejante a un juego de búsqueda del tesoro, en el cual se suman puntos a medida que se avanza por un mapa. Los principales componentes del desarrollo son:

- Dispositivo móvil del estudiante donde debe instalar la App móvil, la que mediante el uso de servicios de la nube, identifica su localización en el ámbito geográfico donde desarrolla sus actividades cotidianas.
- En la base de datos se almacena la estructura de un grafo (en rojo) que representa un mapa virtual de puntos geográficos y las actividades vinculadas al mismo. Cuando el estudiante durante el desarrollo de sus actividades diarias pasa por uno de esos puntos, se le activa un mensaje de alerta que le notifica que tiene una tarea para resolver. También se almacenan los datos vinculados al desarrollo de las actividades.
- Las tareas se encuentran definidas en la plataforma Moodle, desde la cual la App las recupera para presentarlas al estudiante, el que una vez finalizada la actividad

da un aviso. Entonces se le activa desde Moodle una evaluación, para determinar el nivel de aprendizaje alcanzado.

- En caso de superar la instancia de evaluación, el punto se deshabilita y se activa el próximo en el grafo. De no superar la evaluación, se le activa un "pista" que lo oriente para resolver la tarea asignada. Para esto último se le indica que debe buscar en el ámbito donde se desenvuelve, un determinado objeto o imagen y enfocarlo con la cámara del dispositivo móvil, entonces la imagen que visualiza "se enriquece" con información adicional, la cual le sirva de orientación para resolver la tarea.

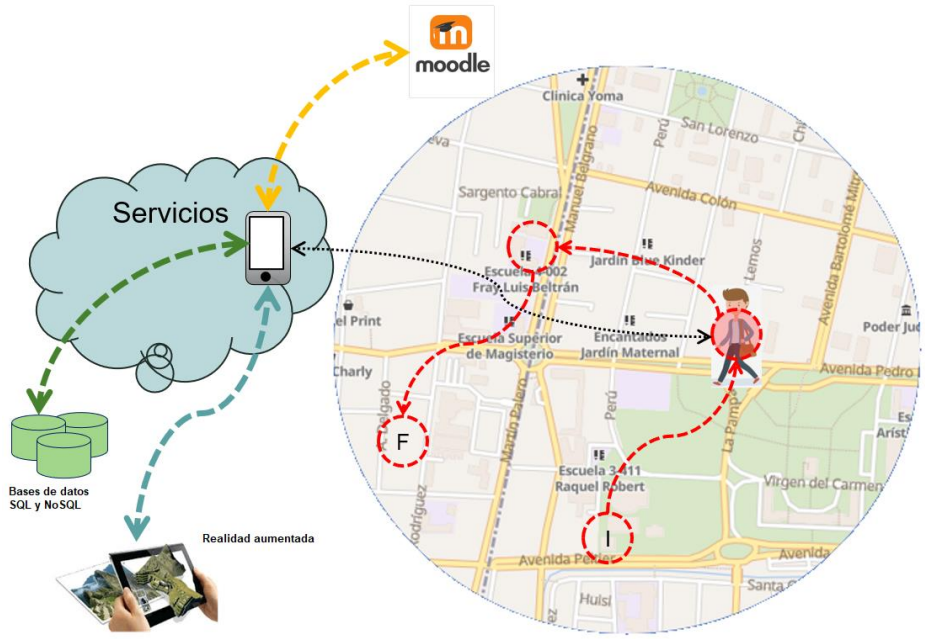

Fig. 1. Esquema de operación del entorno ubicuo.

1.1 Requerimientos

En función de la especificación establecida, los requisitos a resolver en la implementación del entorno son:

1. Cada estudiante debe interactuar con el entorno a través de la App móvil, sin salir de la misma, donde el login del usuario debe identificarlo durante todo su trabajo con el entorno, con la plataforma Moodle y con los componentes de realidad aumentada (RA).
2. La solución debe ser flexible, adaptable a diversos recorridos de aprendizaje y tipos de contenido, descargando la complejidad en los servicios cloud.
3. La App debe ser intuitiva y fácil de utilizar, orientar al alumno respecto del uso y brindar retroalimentación que le permita conocer el estado de su recorrido.

4. En un mismo dispositivo móvil debe poder loguearse más de un usuario, aunque sólo un alumno debe estar logueado al mismo tiempo en un dispositivo.
5. Un usuario puede loguearse en diversos dispositivos móviles.
6. Todos los valores que constituyen constantes para el entorno tales como número de zonas de geovallado a utilizar, direcciones url, mensajes al usuario, códigos de error, etc. deben ser parametrizables.
7. La App deberá vincular de forma dinámica cada unidad temática respetando la secuencia de aprendizaje a una zona de interés geográfica. La vinculación es secuencial en acuerdo a cómo se vayan transitando las áreas de geovallado.
8. La App debe integrar un browser web que permita navegar los recursos hipermedia provistos a los alumnos. El browser debe integrarse con la App para posibilitar que el entorno tenga retroalimentación de las acciones realizadas por el usuario.
9. El entorno debe integrarse de forma transparente para el usuario con el Aula Virtual de Moodle donde están implementados los test de cada unidad de aprendizaje.
10. Los test deben habilitarse de forma personalizada para cada alumno, en el lugar y momento en que el alumno confirma haber leído los recursos y/o pistas previas a cada test.
11. Los test no deben estar habilitados para un alumno en el curso Moodle hasta que el mismo no confirme en la App haber completado la actividad previa al test.
12. Con el objeto de filtrar cuáles alumnos pueden realizar un test, se añade una restricción al Test, para que sólo esté disponible para un determinado Grupo.
13. El entorno implementa en el back-end los servicios que conectan a la base de datos de Moodle para obtener, cuando se requieren, los resultados obtenidos por cada alumno en cada test resuelto.
14. El alumno debe obtener del entorno 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 de geovallado en donde la misma fue resuelta.

1.2 Principales aspectos a resolver

- Transparencia en la integración de la solución con Moodle, que permita desde el entorno y bajo su propia interfase, hacer uso de algunas de las funcionalidades de Moodle.
- Acceso y control de los recursos del dispositivo móvil como la cámara, gps, acelerómetro, etc.
- Garantizar profundidad en la inteligencia de la aplicación (capacidad de respuesta a requisitos funcionales complejos), la elasticidad a la demanda (que pueda soportar múltiples secuencias de aprendizaje, diversos contenidos y un crecimiento en el número de usuarios), la portabilidad y la escalabilidad de la aplicación.
- Agilidad en el desarrollo, entrega y despliegue continuo de nuevas funcionalidades y mejoras. Favoreciendo la aplicación de mejores prácticas de desarrollo, testing, mantenimiento y operación de la solución.

1.3 Alternativas de solución evaluadas

- Construir el front-end como una aplicación Nativa (Android o IOs), una aplicación Web o una aplicación híbrida. Este análisis excede el alcance del artículo, pero es claro que la alternativa elegida fue la App nativa por cuanto permite un mejor control de los recursos del móvil, una mejor eficiencia de procesamiento y una mayor flexibilidad en el diseño de interfase.
- En cuanto al back-end, puesto el objetivo de aprovechar los beneficios de la computación en la nube, alivianando la carga de procesamiento en el móvil y garantizando agilidad de desarrollo (al utilizar un conjunto de servicios previamente disponibles), facilidad en el despliegue, elasticidad a la demanda, portabilidad, escalabilidad y reducción del esfuerzo de operación, se evalúan alternativas de abordaje en cuanto a:
  - la tecnología a utilizar en la construcción de la API que permite integrar el front-end con el back-end, valorándose entre SOAP y REST. Se elige REST porque proporciona mayor versatilidad en el formato de mensajes (soporta JSON además de XML, posibilitando el intercambio de mensajes más simples), es más sencillo de implementar vía peticiones GET, PUT, POST, etc., tiene mejor performance en cuanto al peso de la información transferida y el uso de cache, soporta el procesamiento de transacciones sin estado (stateless) y proporciona escalabilidad simple y directa.
  - el modelo de nube a utilizar (pública, privada o híbrida) [2, 6]. Se decide utilizar nube pública, dado que las conexiones desde dispositivos móviles de los alumnos utilizan redes públicas (Internet). Además, la aplicación consume servicios de nubes públicas (por ejemplo, en la capa de SaaS con Google Apps). Si a futuro se agregaran componentes al interior de alguna red privada en la Universidad (por ejemplo, determinados componentes que sólo estén accesibles dentro de la UTN a través de una conexión a su Intranet), entonces se caerá en un modelo de nube híbrida, pero en principio esto no es necesario.
  - la plataforma cloud a utilizar, para lo que se evaluaron [3, 4, 5]: App Engine de Google, RedHat Openshift de IBM Cloud, Kubernetes Services de IBM Cloud y Azure de Microsoft. También la factibilidad de implementar todos los servicios del back-end en servidores propios o en máquinas virtuales. Se optó por una solución con independencia de la infraestructura y arquitectura de despliegue subyacente. De este modo, la solución construida se puede desplegar de forma contenerizada, sobre servidores físicos reales o virtual machines siempre y cuando se parta de imágenes que contengan un servidor de aplicaciones Java y un motor de base de datos relacional.

Se eligió una arquitectura cliente-servidor, en la que el back-end se implementa como una plataforma SaaS y ofrece un conjunto de servicios web, del tipo RESTful, que son consumidos por una aplicación cliente o front-end, implementada mediante una App nativa Android, en adelante la App como se muestra en al Fig. 2.

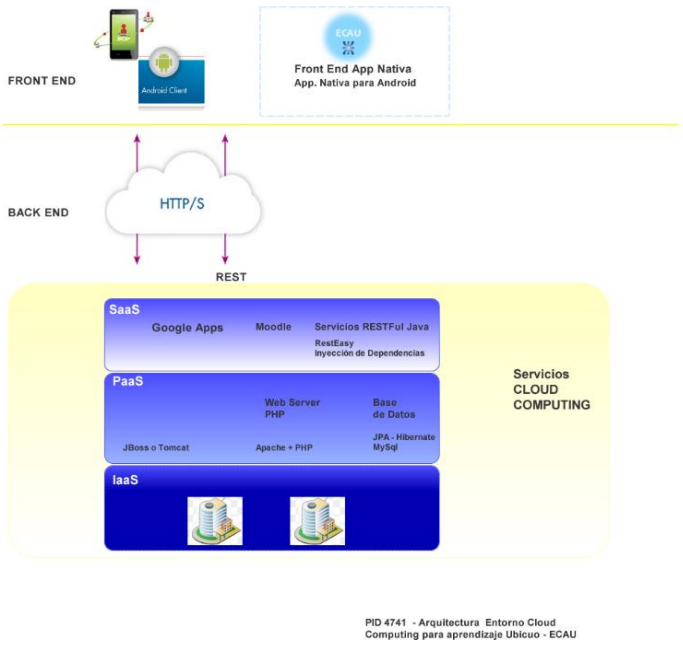

Fig. 2. Arquitectura general.

2 Implementación

Siguiendo un modelo Mobile Cloud Computing (MCC), las capacidades de almacenamiento y procesamiento de información son resueltas del lado del servidor utilizando servicios de computación en la nube. En particular se implementa una arquitectura de tres capas para el back-end:

- Capa SaaS (Software como servicios): es la capa más alta e involucra al software (aplicaciones) ofrecidas como servicios que ejecutan en servidores en la Nube. En este punto ubicamos como desarrollo propio, los servicios web Restful (Resteasy) que soportan la lógica de negocios del entorno junto su modelo de datos; y como utilización de servicios de terceros, la API de Google Apps que ofrece servicios básicos de geo-referencia y ubicación en mapas (Google Maps API). El entorno, integra además con la plataforma de aprendizaje en línea Moodle. Tal integración se resuelve a nivel de modelo de datos, accediendo de manera directa a las tablas correspondientes de la BD de Moodle.
- Capa PaaS (Plataforma como servicios): provee una pila de recursos de base para el desarrollo, testing y puesta en producción de aplicaciones como servicios (utilizando Servidor Linux Ubuntu 14.04. Servidor web Nginx + servidor de aplicaciones Java EE (Wildfly 10) + PHP 7.3. Base de Datos MariaDB v10.3.25)

- Capa IaaS (Infraestructura como servicios): capa más baja que disponibiliza hardware con capacidades de almacenamiento y cómputo como servicios estandarizados en la red. Servidores, sistemas de almacenamiento, routers, balanceadores de carga, dispositivos de seguridad, etc. En esta capa utilizamos, para la construcción y prueba del prototipo un servidor virtual (droplet) provisto por DigitalOcean que escala vertical y horizontalmente según la demanda de servicios. Los servicios de networking son provistos por la misma empresa DigitalOcean.

Como ya se mencionó, la solución utiliza una Nube Pública, dado que las conexiones desde dispositivos móviles utilizan redes públicas (Internet). Así mismo, se consumen recursos de nubes públicas, tales como la capa de SaaS de Google Apps. Si a futuro, el proyecto escala y se implementa una interfase de diseño, administración y carga del Objetos de Aprendizaje en el entorno, para que sea utilizado sólo por docentes desde la red privada de la UTN, entonces caeremos en un modelo de nube Híbrida.

Respecto al intercambio de información sensible para el usuario entre la APP y los servicios en el back-end, cabe mencionar en primer lugar que el acceso a los recursos del móvil, como GPS o cámara, queda bajo el absoluto control del usuario y media su expreso consentimiento. Así mismo, las coordenadas de ubicación del alumno son comparadas en la propia APP con las áreas de geovallado establecidas y administradas por la API Google Geofences, valiéndose incluso de las últimas lecturas de ubicación realizadas por otras aplicaciones como WhatsApp, Google Maps, etc. No obstante lo anterior, destaca que todo el intercambio de información entre el front-end y los servicios en la nube se realiza mediante protocolo HTTPS, garantizando así la confidencialidad de la información mediante cifrado extremo a extremo.

2.1 Arquitectura

La aplicación que presta servicios del lado del servidor (SaaS), se despliega sobre un servidor de aplicaciones Java implementado en la capa de PaaS (Plataforma como servicios) en nuestro contexto de desarrollo cloud. En particular se ha usado Wildfly 10.0, no obstante, el desarrollo es portable y puede desplegarse en otros applications servers tales como JBoss, Tomcat, GlassFish, WebLogic, etc. La aplicación en el back-end respeta una arquitectura por capas con la estructura que muestra la Fig. 3.

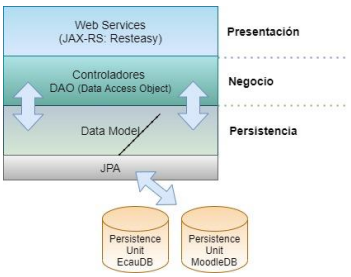

Fig. 3. Arquitectura back-end

Capa de Presentación:

Constituye la interfase del back-end con el front-end. Se ubican aquí los servicios web que el servidor brinda a la App Nativa Android u otros clientes que a futuro quisieran integrarse. Estos servicios constituyen 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, han sido desarrollados usando Resteasy. Resteasy es una implementación portable de la especificación JAX-RS (JSR 311 & JSR 339) [1] que funciona en cualquier contenedor de Servlets. Constituye por tanto una API Java para implementar Restful Web Services. En este caso, la elección de Resteasy radica fundamentalmente en su simplicidad de configuración y programación, y en su integración directa con Wildfly puesto que está embebido en este servidor de aplicaciones.

La App nativa Android intercambia información con los servicios web mediante la serialización de objetos con JSON (JavaScript Object Notation). Se elige JSON porque provee una alternativa sencilla de implementar, estandarizada y estable. Es más liviana que XML y más segura que el intercambio de objetos binarios.

Si bien la especificación JAX-RS solo se ocupa de los servicios en el servidor (server-side), Resteasy provee también un JAX-RS Client Framework que permite desarrollar clientes Java para los servicios web implementados.

Capa de Negocio:

Se incluyen en esta capa las clases de control del modelo Orientado a Objetos (OO) que gestiona los procesos del entorno. Es decir, el conjunto de clases y relaciones entre clases que modelan el registro de un alumno y la gestión de su recorrido por la secuencia de unidades temáticas y actividades conforme a su paso por las zonas de geo-vallado definidas. Los servicios web en la capa de presentación hacen uso de los objetos definidos en este modelo, con sus datos y comportamientos. 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 hecho de usar JPA para acceder a la base de datos relacional automatiza las conversiones de datos necesarias para realizar transacciones, y resulta ser una tecnología que se escala bien conforme una base de datos crece.

Capa de Persistencia:

El modelo de datos que soporta la aplicación se almacena en la BD relacional MariaDB la que constituye un fork de MySQL. La conversión del modelo orientado a objetos implementado en la capa de negocios, a un modelo de datos relacional es resuelto por JPA/Hibernate. En este punto se debe tener claro que JPA es una especificación, parte de EJB 3 (JSR 220), mientras que Hibernate es un framework de persistencia que implementa dicha especificación. En este sentido, el desarrollo también gana independencia respecto del framework de persistencia, ya que Hibernate podría reemplazarse por otro en caso de ser necesario, tal como Eclipselink, ObjectDB, Toplink, OpenJPA, etc. Una limitación que impone 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 BD NoSQL como por ejemplo MongoDB. No obstante, en el contexto del desarrollo del entorno y de los trabajos de laboratorio de software complementarios se decidió priorizar la mayor independencia entre capas.

Destaca que el motor de BD utilizado puede reemplazarse de manera sencilla por otros tales como PostgreSQL u 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. Hibernate es a su vez la capa de implementación de persistencia que la aplicación utiliza a través de la especificación JPA (Java Persistence API).

3 Estructura de proyecto

En concordancia con la arquitectura de capas propuesta para el back-end, el código fuente se estructura en cuatro paquetes:

- Service: contiene las clases donde se implementan los servicios web del entorno. Estos servicios exponen las funcionalidades de procesamiento de información resueltas en el back-end para que sean consumidas desde la App móvil.
- Controller: contiene todas la interfaces y clases que implementan los objetos de acceso a datos (DAO Data Access Object). 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 (tales como registrar un alumno, actualizar datos de coordenadas, consultar el estado del recorrido de un alumno, etc.) pero con total independencia de la tecnología de persistencia subyacente.
  - A nivel de clases (capa intermedia) la implementación mediante JPA (Java Persistence API) de las operaciones anteriormente definidas para cada objeto de acceso a datos - DAO. JPA resuelve luego de manera transparente para los developers la implementación concreta del modelo de persistencia en una tecnología de base puntual (en nuestro 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. Destaca que el back end del entorno está basado en el Modelo de Programación Orientado a Objetos (POO) implementado en Java, y mantiene total independencia del motor de persistencia y de la BD con que se implementa el almacenamiento en las capas subyacentes. Esto implica que, si bien en esta implementación se ha desplegado el proyecto utilizando como motor de persistencia Hibernate y la base relacional MariaDB, el desarrollo puede también desplegarse de forma transparente sobre otras arquitecturas de almacenamiento. Así, el desarrollador solo se ocupa de implementar el diagrama de clases propuesto sin preocuparse de cómo se persistirán los datos. La persisten-

cia de los objetos en una base de datos relacional se resuelve mediante JPA. Esta independencia de la arquitectura subyacente proporciona como ventaja mayor posibilidad de abstracción en el modelo, mayor portabilidad, y mayor escalabilidad futura de la aplicación.

Se han tenido en cuenta en el uso de anotaciones y codificación de controladores y servicios, las buenas prácticas recomendadas tanto por la especificación REST JAX-RS, como por CDI y JPA para lograr una BD adecuadamente normalizada y portabilidad en el código a otros Applications Servers y Motores de BD.

Se incluyen además en este paquete, aquellas clases que modelan los objetos JSON que sirven al intercambio de información mediante web services entre el front-end y el back-end.

- Útil: contiene clases utilitarias que cooperan con otras clases del modelo, tales como la "Configuración.java" que define el acceso al archivo XML de propiedades y parámetros del entorno, el acceso al log de transacciones, etc.

El código incluye también archivos de configuración y parametrización que permiten:

- Parametrizar valores y mensajes
- Definir las unidades de persistencia utilizadas en el modelo (persistence units).

Cada unidad de persistencia establece la vinculación entre un conjunto de clases de entidad en el modelo OO y una fuente u origen de datos en particular (datasource). Las fuentes de datos se configuran en el servidor de aplicaciones donde se despliega el proyecto.

Las fuentes de datos definen la BD sobre la cual se persistirán datos, los drivers de conexión (en este caso se usa JDBC) y el motor de persistencia que se desea utilizar. En el contexto de este proyecto, se ha configurado dos unidades de persistencia: aquella que permite al modelo integrar con su BD propia ecauDB y aquella que permite integrar con la BD de Moodle.

La definición de las unidades de persistencia constituye la única referencia que existe en el proyecto a la arquitectura de base utilizada para implementar la persistencia de datos. Cualquier cambio en esta arquitectura de base (por ej.: si se cambia 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. Esto implica que con solo modificar unas pocas líneas en documentos XML de configuración se puede cambiar la tecnología de BD de todo el proyecto. Del mismo modo también se puede escalar agregando nuevas bases de datos con solo agregar unidades de persistencia en el proyecto y datasources en el applicaton server.

Destaca que el desarrollador solo debe concentrarse en codificar el diagrama de clases e introducir las anotaciones JPA pertinentes. Al desplegar el proyecto en el servidor de aplicaciones, estando configurados los datasources correspondientes se creará automáticamente la estructura de BD con las tablas requeridas, sus claves primarias, las claves foráneas, etc. Esto da cuenta de las facilidades que el uso de las tecnologías de persistencia utilizadas aporta a la portabilidad, flexibilidad y escalabilidad de este tipo de desarrollos.

4 Conclusiones

Con el desarrollo se ha buscado implementar una solución de software, para apoyar el aprendizaje autónomo, colaborativo, continuo y significativo en un entorno virtual diferente y permanentemente accesible para el alumno, dándole utilidad a un dispositivo que utilizan en su vida cotidiana. Para lograrlo se propone crear una aplicación que, cumpliendo un conjunto de requisitos funcionales y técnicos específicos, posibilite al alumno construir conocimientos de una forma lúdica, versátil y atractiva a sus intereses, proporcionándole actividades de aprendizaje situadas en el contexto.

Esta aplicación debe poder instalarse en un dispositivo móvil del alumno sin consumir excesivos recursos -para que sea asequible a una población amplia de estudiantes pero debe tener la capacidad de cómputo necesaria para servir un gran número de recursos en diversos formatos, resolver una lógica de procesos compleja, almacenar un importante volumen de datos y poder escalar a demanda en cantidad de usuarios, en cantidad de contenido y/o en nuevas funcionalidades. Debe además integrarse a la plataforma Moodle, para aprovechar sus módulos de gestión de aulas y grupos de estudiantes, herramientas de evaluación, calificación y gestión de contenido. También es necesario que pueda consumir los servicios de georeferencia y geofencing de Google y pueda integrarse con componentes de RA.

Para dar respuesta a lo anterior, se piensa en una solución MCC (Mobile Cloud Computing) que integre una App Móvil delgada a un conjunto de servicios cloud que otorguen la profundidad y experiencia requerida.

El producto obtenido, plenamente funcional, permite que el alumno perciba la App móvil (el front end del entorno) como una solución integral que le permite resolver todas las actividades de su recorrido de aprendizaje, sin que requiera salir de la aplicación para interactuar con Moodle o visualizar un recurso en otra interfase. El estudiante percibe la aplicación como un entorno accesible y lúdico, que le acerca contenidos y actividades al lugar en que se encuentra.

La solución de cloud computing implementada en el back end, resuelve la complejidad de procesamiento y posibilita que el producto sea flexible y pueda adaptarse a diversos recorridos y tipos de contenido u objetos de aprendizaje que los equipos docentes de distintas disciplinas pueden construir.

Aprovechando los beneficios de la computación en la nube, la solución es elástica a la demanda, sin que el crecimiento en el número de usuarios o la cantidad de contenido almacenado sea una limitante.

La arquitectura propuesta para el back-end, prioriza la independencia entre capas posibilita que los servicios web incluidos en la capa de presentación no deban preocuparse de cómo se implementan las operaciones sobre los objetos persistentes. Así mismo, la capa de negocios ve el modelo de datos como un modelo OO sin preocuparse en cómo luego la capa de persistencia resuelve el almacenamiento de datos. De este modo, el modelo es independiente del motor de persistencia y de la BD con que se implementa el almacenamiento en las capas subyacentes. Esta independencia de la arquitectura subyacente proporciona como ventaja mayor posibilidad de abstracción en el modelo, mayor portabilidad, y mayor escalabilidad futura de la aplicación.

5 Reconocimiento

El artículo es producto 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.

6 Referencias

1. Almanzar Espitia, N., Vanzina Solis, J.D. Comparación de implementaciones JAX-RS. Resumen Analítico en Educación - RAE. Universidad Católica de Colombia, Facultad de Ingeniería en Sistemas. Bogotá (2020), https://repository.ucatolica.edu.co/bitstream/10983/2716/2/RAE.pdf
2. Barry, D., Dick, D. Web Services, Service-Oriented Architectures, and Cloud Computing: The Savvy Manager's Guide. Ed. 2. Amsterdam; Boston: Elsevier; Waltham [Mass.]: Morgan Kaufmann (2013)
3. Raj Amal, W. (2015). Learning Android Google Maps. Packt Publishing. ISBN-10:1849698864. ISBN-13:978-1849698863.
4. 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)
5. Sproviero, F. Geofencing API tutorial for Android. (2020), https://www.raywenderlich.com/7372-geofencing-api-tutorial-for-android.
6. Zirpins, C., Paraskakis, I., Andrikopoulos, V., Kratzke, C., y otros. (2021). Advances in Service-Oriented and Cloud Computing: International Workshops of ESOCC 2020. Heraklion, Crete, Greece, September 28-30, 2020, Revised Selected Papers. Springer Nature.