ENTORNO DE ENSEÑANZA UBICUO Y MOBILE CLOUD-COMPUTING

Mariana I. Brachetta & Oscar A. León

Ingeniería en Sistemas de Información/FR.Mendoza -UTN

mariana.brachetta@docentes.frm.utn.edu.ar; oleon@frm.utn.edu.ar

Resumen

En el trabajo se describe el abordaje adoptado para implementar el acceso a los recursos requeridos para el funcionamiento de un entorno de enseñanza ubicuo, el cual es soportado mediante servicios de cloud-computing. En el ambiente propuesto se integran tecnologías de computación móvil, realidad aumentada, geo-referencia, bases de datos y entornos virtuales de enseñanza. Dado que parte de los recursos se encuentran alojados en la nube, para el acceso a los mismos, se requiere del uso de web-services a fin de integrarlos con el resto de los componentes del entorno. En particular, en el artículo se comenta la arquitectura de software propuesta, así como las características del front-end y el back-end para la integración de servicios, también se trata la descripción funcional del entorno y finalmente se comentan algunos de los beneficios que el uso de MCC (Mobile Cloud-computing) aportó para la implementación del ambiente ubicuo propuesto. El trabajo se enmarca en el proyecto (PID 4741) "Desarrollo de un entorno basado en cloud-computing para aprendizaje ubicuo" que se lleva adelante en la Universidad Tecnológica Nacional, Facultad Regional Mendoza.

Palabras clave: cloud-computing, aprendizaje ubicuo, entornos virtuales, web-services

Introducción

El proyecto desarrollado utiliza servicios de la "nube" para implementar el siguiente modelo (Ilustración 1) para crear un ambiente ubicuo de aprendizaje.

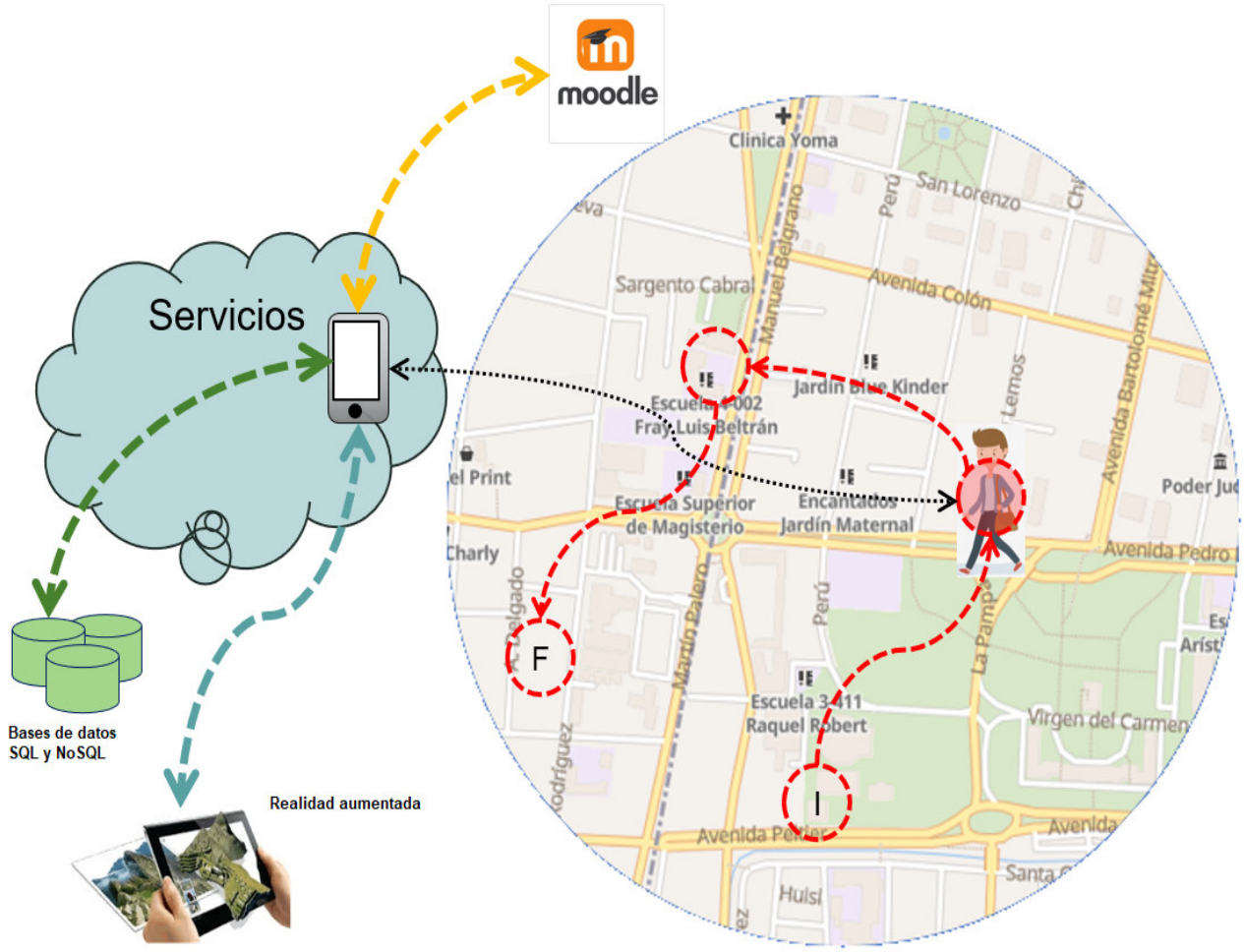


Ilustración 1: Esquema del entorno ubicuo

Los principales componentes son:

- Dispositivo móvil del estudiante donde debe instalar la App, la que mediante el uso de servicios de la nube, identifica su localización en el ámbito geográfico donde desarrolla sus actividades cotidianas.
- Base de datos que 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 activa un mensaje de alerta notificando que tiene una tarea para resolver. También se almacenan los datos vinculados al desarrollo de las actividades.
- Moodle donde se encuentran definidas las actividades de enseñanza, desde la cual la App las recupera para presentarlas al estudiante, que una vez finalizada, 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.
- Objetos de realidad aumentada, que se activan en el caso de no superar la evaluación, para ofrecer un "pista" que oriente en la resolución de la tarea asignada. Para esto último se indica

que se debe buscar en el ámbito donde se desenvuelve el estudiante, un objeto o imagen y enfocarlo con la cámara del dispositivo móvil, entonces la imagen que visualiza "se enriquece" con información adicional, la cual le sirve de orientación para resolver la tarea. Los objetos son desarrollados con Unity/Vuforia.

La amplia difusión de la tecnología de "computación en la nube" (cloud-computing), la cual ha tenido un rápido desarrollo y logrado un alto grado de confiablidad [1], hace que la misma se haya convertido en una tecnología viable de ser aplicada en diversos tipos de proyectos, entre ellos los educativos. La tecnología provee facilidades para prácticamente todas las actividades humanas que requieren servicios de computación.

Una de las áreas que se puede beneficiar de "la nube", es la educación, ya que rompe con algunas de las limitaciones presentes en las aplicaciones de e-learning [4, 5], superando aspectos como la capacidad de almacenamiento y de procesamiento. Las aplicaciones pueden ofrecer a los estudiantes servicios más potentes, sin consumir recursos propios del dispositivo móvil.

También existen antecedentes respecto de los beneficios de combinar m-learning y cloud-computing [6], para mejorar la comunicación entre estudiantes y profesores, como por ejemplo mediante el uso de software para dispositivos móviles con Google Apps Engine, o la incorporación de prestaciones de "Realidad Aumentada" aplicada al ambiente donde se mueve el estudiante en su vida cotidiana. También potencian la aplicación de tecnologías móviles, las iniciativas que ofrecen espacios de movilidad para alumnos y docentes, como EDUROAM.

La infraestructura de servicios disponibles en "la nube", ha evolucionado hacia un contexto donde se ejecutan aplicaciones de forma confiable y segura, con capacidad de respuesta elástica para atender los cambios en la demanda. Lo anterior conforma una capa de tecnologías y servicios [2], sobre la cual se monta aquella que se ocupa de integrarlos para poner a disposición de los usuarios finales las aplicaciones de computación.

Así Mobile Cloud Computing se refiere más bien a una forma de trabajo, donde las aplicaciones móviles al no almacenar datos en el dispositivo y descargar parte del procesamiento en "la nube", se ven potenciadas. De esta forma se ha simplificado el trabajo de desarrollo de aplicaciones, su utilización y atención, trasladando el problema a la integración de los servicios disponibles [3].

La arquitectura general de Mobile Cloud Computing (MCC) se puede dividir en tres partes, en un extremo los dispositivos móviles, en el otro los proveedores de servicios, ambos mediados por Internet. Los dispositivos móviles se conectan a las redes móviles a través de estaciones base que establecen y controlan las conexiones mediante interfaces entre las redes y los dispositivos. Las solicitudes y datos de los usuarios móviles se transmiten a servidores que proporcionan los servicios de red móvil (autenticación, autorización, contabilidad de datos, persistencia, geo-referencia, etc.).

Lo anterior requiere del desarrollo de web-services para integrar los distintos elementos que intervienen.

Arquitectura propuesta

El entorno se basa en una arquitectura cliente-servidor, cuya estructura general se muestra en la Ilustración 2, en la que el back-end se implementa como una plataforma de software como servicios (SAAS) y ofrece un conjunto de servicios web, del tipo RESTful, que son consumidos por una aplicación cliente o front-end[1] implementada mediante una App nativa Android, en adelante la App.

[1] En contraposición con las Apps móviles autocontenidas, donde todo es resuelto en el cliente.

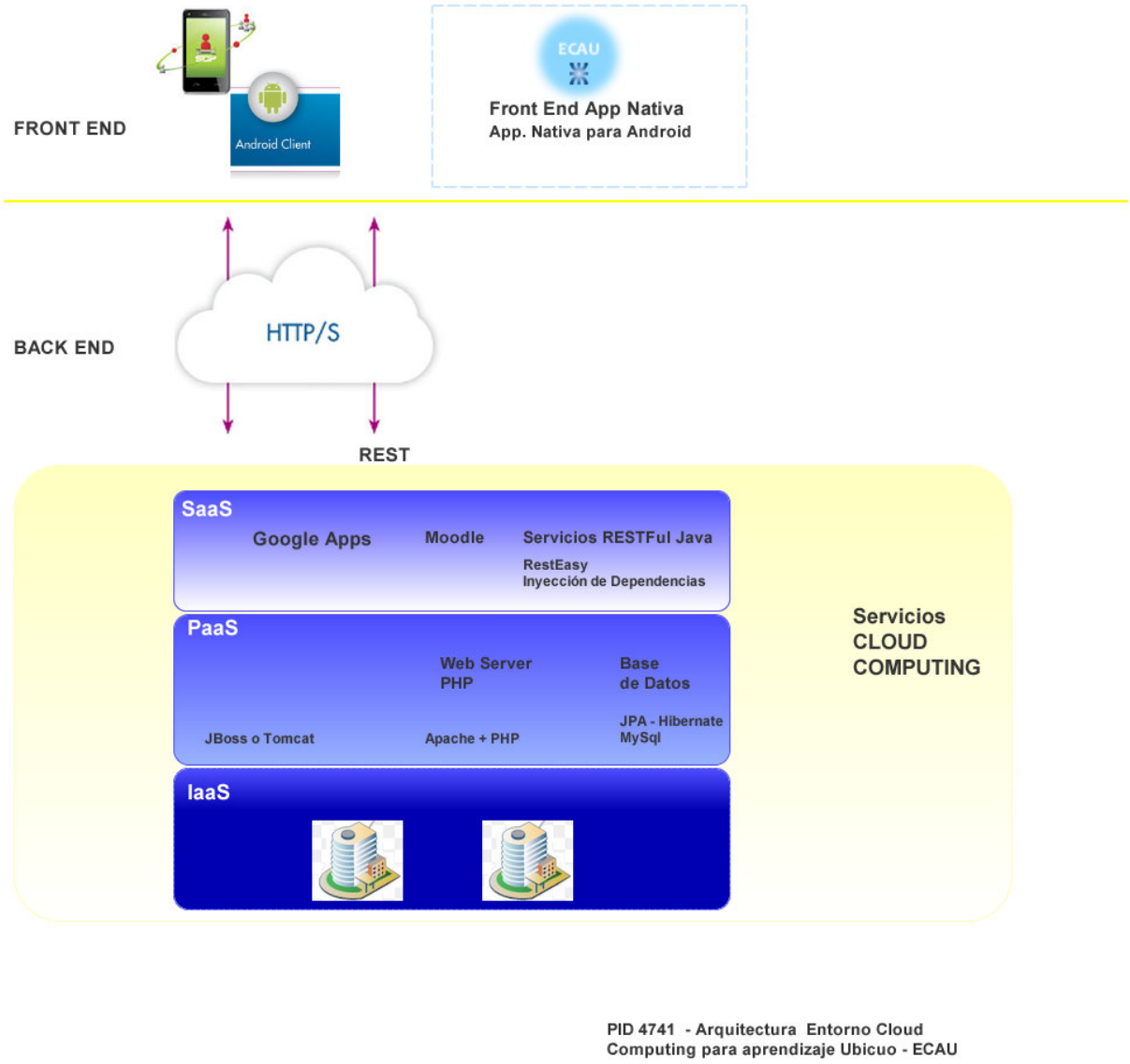


Ilustración 2: Arquitectura general

Arquitectura del front-end

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) 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 siguiente estructura.

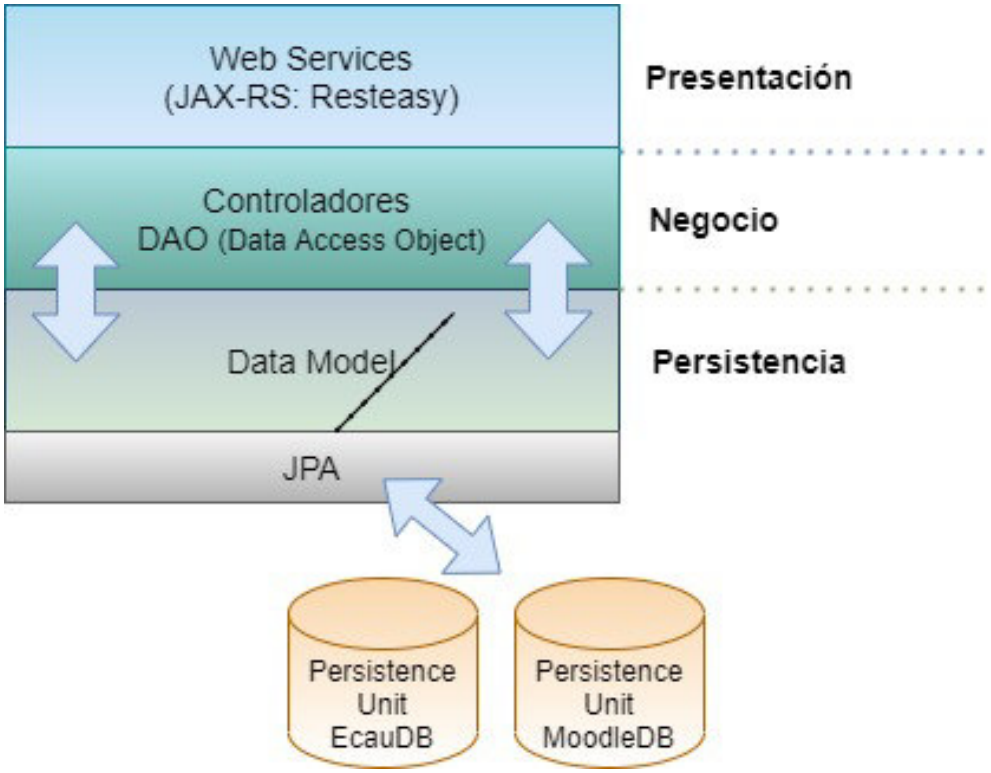


Fig. Arquitectura back end

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. 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) 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. *(Ver Anexo I).*

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. (ver Anexo I).

Si bien la especificación JAX-RS solo se ocupa de los servicios en el servidor (server-side). No obstante, Resteasy provee también un JAX-RS Client Framework que permite implementar clientes Java para los servicios web implementados.

Nótese que 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: 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 demuestra 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 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).

La App se encarga de geolocalizar al alumno y proveerle, en un ambiente integrado, una secuencia de actividades que se van activando conforme al alumno avanza en un recorrido geográfico. Dichas actividades son administradas desde el back-end. La App se conecta a los servicios de información provistos por el back-end a través de clientes de servicios web RESTful. En particular se utilizó la implementación de la comunidad de JBoss, Resteasy.

Trabajar con RESTful simplifica la implementación de los servicios en el back-end ya que no es necesario guardar el estado de las sesiones cliente en el servidor (statelessness)[2]. Los servicios resultantes, además, mantienen una interfase muy estable y portable, que facilita la conexión desde clientes construidos con diversas tecnologías (Java, Javascript, Php, Kotlin, etc.).

Se decidió implementar el cliente como una aplicación nativa Android escrita en Kotlin. Tal elección se fundamenta en:

- las funcionalidades requeridas, las cuales resultaban más difíciles de implementar en una interfase web
- una mayor facilidad de acceso a los recursos del móvil (Cámara, GPS, almacenamiento local, etc.)
- mayor velocidad de ejecución
- independencia de conexión para los aspectos que la aplicación puede resolver localmente

Arquitectura del back-end

La App se construyó utilizando Google JetPack y su arquitectura por componentes para aplicaciones Android, ajustada al patrón MVVM (Model - View - View Model).

La elección de la arquitectura, componentes y herramientas involucradas en el desarrollo se adecua a las recomendaciones y buenas prácticas establecidas por Google para el desarrollo de aplicaciones móviles nativas de Android, según publicaciones del año 2020 *(Ver Guía de Arquitectura).*

[2] https://restfulapi.net/statelessness/

El patrón de diseño MVVM (Modelo-Vista – Vista-Modelo), propuesto por Microsoft en 2005, propone una arquitectura basada en capas y un conjunto de principios que aplicados adecuadamente conducen a la construcción de aplicaciones robustas, escalables, fáciles de testear y mantener.

La arquitectura por componentes de Google reúne una colección de librerías que proveen una plantilla adecuada para implementar el patrón MVVM en el ámbito de las aplicaciones móviles; y JetPack provee el conjunto de componentes necesarios.

Si bien existe un amplio número de publicaciones que documentan el patrón de diseño MVVM y el conjunto de componentes JetPack, presentamos a continuación una breve descripción de ellos para favorecer luego la comprensión de la arquitectura del proyecto y la introducción de los componentes utilizados en su construcción.

Patrón MVVM (Model - View – View-Model)

MVVM es un patrón de diseño arquitectónico[3] cuyo principio básico orienta a separar la aplicación en tres componentes o capas: el modelo, la vista y el modelo de vista.

- **Modelo**: es la capa de datos, encargada de gestionar la persistencia de datos, ya sea en un repositorio local o remoto (a través de API). Su funcionamiento debe ser independiente del resto de los componentes. El ViewModel consulta al Modelo para obtener los estados en que se encuentra la aplicación.
- **Vista:** resuelve todo lo relacionado a la interfase de usuario. En Android está implementada por las actividades (Activities) o fragmentos (Fragments). En ella se define cómo se visualizan los datos del modelo que son proporcionados por el ViewModel y a su vez cuándo se deben reenviar datos al modelo, como respuesta a acciones del usuario, mediante el ViewModel. Debe ser una capa liviana, independiente de la lógica de la aplicación. La vista se comunica con el modelo mediante variables observables y acciones implementadas en el ViewModel.
- **ViewModel:** comunica la Vista con el Modelo a través de una capa intermedia que mantiene la independencia[4] entre ambos. Es el componente central de la arquitectura propuesta por MVVM, puesto que reúne toda la lógica de negocios y la lógica de presentación. El ViewModel es el responsable de acceder al Modelo y preparar los datos observables que deben ser expuestos en la vista. En sentido inverso, también reacciona a los eventos de entrada en la UI (Interfase de Usuario) para obtener los datos que debe enviar al Modelo para que sean almacenados. Así, el ViewModel implementa la lógica de presentación y los métodos que definen las funcionalidades de la aplicación en respuesta a acciones del usuario en la vista o a cambios en el Modelo. Los cambios en el ViewModel cambian automáticamente la vista y viceversa, pero existe una total independencia entre ambas capas.

En el mundo de las aplicaciones móviles, destacan fundamentalmente tres patrones de diseño arquitectónico: MVC (Model View Controller), MVP (Model View Presenter) y MVVM (Model-View–ViewModel).

El patrón MVVM se propone como una evolución al patrón MVC (Modelo Vista Controlador) en el ámbito de las aplicaciones móviles. La mayor parte de los autores destacan también algunas ventajas de este patrón sobre el patrón MVP (Model View Presenter). Sin perjuicio de lo anterior, la elección de uno u otro dependerá del contexto y complejidad particular de la App a desarrollar. En nuestro caso, decidimos seguir este patrón por ser lo más recomendado en la actualidad y por la facilidad que proveen los componentes de JetPack para implementarlo.

Entre las ventajas del patrón MVVM, la mayor parte de las fuentes enumeran:

---

[3] Un patrón de diseño arquitectónico es aquel que propone a los desarrolladores el estilo al que debe ajustarse la arquitectura de una aplicación.

[4] Por independencia se entiende que un cambio en una capa no debe afectar o requerir cambios en otra capa en la arquitectura.

- Permite desacoplar mejor la Vista del Modelo. Es decir, la introducción del ViewModel promueve una mejor arquitectura al separar la UI de los datos, facilitando el manejo del ciclo de vida de la aplicación y mejorando las posibilidades de testing. El ViewModel no sabe nada sobre la Vista o vistas susbcriptas a los observadores. Esto se logra gracias al Data Binding (enlace de datos) que permite entregar los cambios en el Modelo a la Vista y viceversa. Esto permite desarrollar y probar la Vista de forma aislada y sin acoplamiento con el ViewModel. En nuestra experiencia la promesa no se concretó en la práctica de una forma tan transparente como se proponía en las fuentes consultadas.
- MVVM es muy similar a MVP en lo que respecta a la estructura de proyecto y organización del código, pero con la significativa diferencia que introduce el enlace de datos bidireccional (Bidirectional Data Binding). Esto implica que cualquier cambio en el Modelo se refleja inmediatamente en la Vista y viceversa; lo que contribuye al desarrollo de aplicaciones altamente reactivas sin requerir un amplio esfuerzo de codificación. El enlace de datos bidireccional es una implementación del patrón Observable.
- En las aplicaciones móviles, el ViewModel se ajusta al ciclo de vida del componente visual al que está vinculado (Activity o Fragment), por lo que es resistente a cambios de configuración como una rotación de pantalla o cuando la App sale del primer plano, manteniendo el estado de los datos.
- Es un patrón muy maduro, ya que desde su presentación en 2005 ha sido incorporado en la mayor parte de los Frameworks orientados a UI; inicialmente en el mundo de las aplicaciones web y más adelante para las aplicaciones móviles.
- Conduce a aplicaciones robustas con una codificación limpia y legible. La organización del código en capas favorece la escalabilidad y mantenimiento posterior de los desarrollos.

Arquitectura por Componentes

La siguiente figura ilustra la arquitectura por componentes sugerida por Google en su "Guía de arquitectura de Apps". Como se mencionó anteriormente, nuestro proyecto respeta esta arquitectura y sus principios, bajo al patrón MVVM.

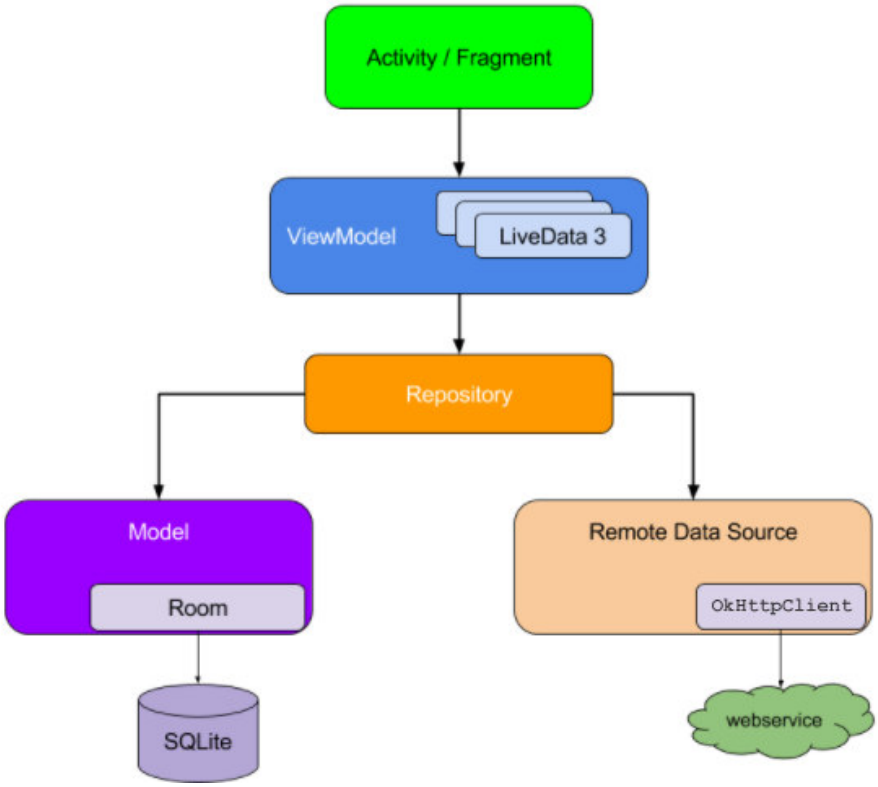


Fig. Arquitectura propuesta para la App - Fuente Guía de arquitectura de Apps Android https://developer.android.com/jetpack/guide?hl=es-419

Siguiendo el principio de separación de problemas:

- La capa de UI, implementada en las actividades y fragmentos de la aplicación, sólo incluye la lógica que resuelve las interacciones con el SO y los eventos de usuario sobre los componentes gráficos. Se ha intentado mantener esta capa lo más limpia posible, aunque en lo concreto este postulado puede resultar dificultoso para un equipo de desarrollo no experto en el tema. No obstante, el uso de componentes JetPack contribuyo a conducir el desarrollo en el sentido correcto. En nuestro proyecto, tal como se detalla más adelante se optó por implementar esta capa bajo el modelo "Una actividad – múltiples fragmentos" para simplificar aún más los problemas que típicamente suceden en las aplicaciones móviles con la gestión del ciclo de vida y la interacción con el SO.
- Siguiendo las sugerencias de la Guía, el Modelo que gestiona los datos de la aplicación, se implementó siguiendo un patrón Repository que usa el ORM Room Persistence, incluido como componente de JetPack, para los datos que se persistan localmente en el móvil. La API REST permitió la integración con el repositorio remoto a través de web services disponibles en el back end. Room utiliza Objetos de Acceso a Datos (DAO – Data Access Object) para proporcionar operaciones CRUD sobre el repositorio local (en nuestro caso en la BD SQL Lite). Según la documentación, el modelo de persistencia es recomendado por cuanto los usuarios no perderán datos cuando el SO cierra la App para liberar recursos. Así mismo, se sugiere mantener en el repositorio local aquellos datos que permitan a la aplicación seguir funcionando cuando la red sea inestable o no esté disponible. Más adelante se describen las consideraciones relativas a la decisión de mantener coordinadamente dos repositorios de datos para el entorno: uno local a la App móvil y otro remoto en el back end, detallando los pro y contras experimentados con relación a esta decisión.
- El Modelo de Vista (ViewModel) vincula la UI con el Modelo manteniendo la independencia entre estas capas. El ViewModel utiliza componentes del tipo LiveData y MutableLiveData, que se detallan más adelante, para implementar el Data Binding entre la capa visual y el modelo.

Según la Guía de Google, cada componente debe depender sólo del que está un nivel más abajo en la arquitectura. Así, son las actividades y/o fragmentos quienes se suscriben al ViewModel para conocer u "observar" los cambios en los datos e informar eventos que impliquen cambios al modelo. El ViewModel depende del Repositorio y este a su vez del modelo de persistencia local y del origen de datos en el back end que se vincula mediante la API REST (clientes de web services).

La siguiente figura ilustra el flujo de datos que se produce entre las distintas capas de la arquitectura.

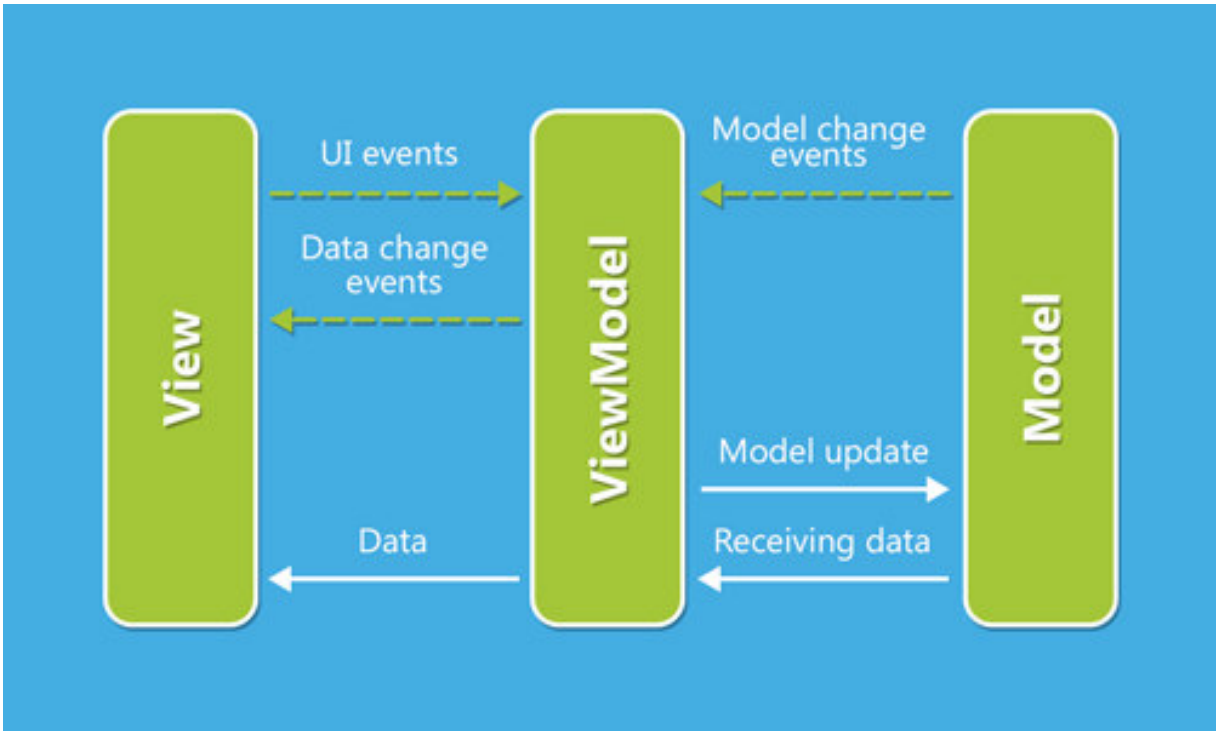


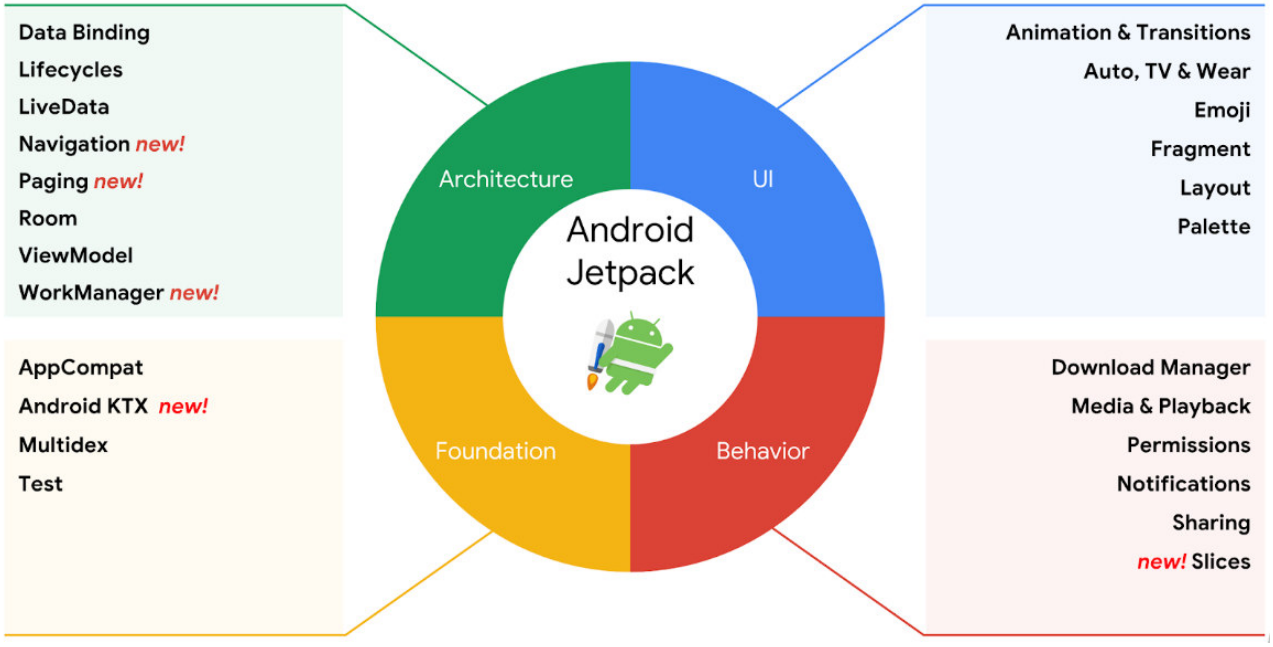
Fig. MVVM Arquitectura – Fte.: https://medium.com/@umang.burman.micro/login-example-with-mvvm-databinding-with-livedata-81319048afb0

Nota: En la sección 9.3. se describe una de las secuencias del patrón MVVM en la App, para ilustrar estos conceptos.

JetPack

Para implementar la arquitectura se utilizó JetPack, una colección de bibliotecas de Android provista por Google, que incorpora prácticas recomendadas y proporciona compatibilidad con Apps orientadas a versiones antiguas de Android.

JetPack proporciona un conjunto de librerías y herramientas en cuatro áreas clave del desarrollo de Android, que permiten implementar los principios propuestos por la Guía sobre la arquitectura de Apps, con el objetivo de crear aplicaciones robustas, reactivas, escalables y testeables que promuevan una experiencia de usuario satisfactoria.



Fuente: https://developer.android.com/jetpack/androidx/explorer?hl=es-419

Cada uno de los componentes o librerías incluidos en JetPack se pueden usar de forma individual, pero su adecuada combinación permite aumentar la potencia de sus funcionalidades, favoreciendo una codificación limpia y ordenada.

Google provee amplia documentación sobre cada una de las cuatro dimensiones de JetPack y los componentes incluidos en cada una. En el contexto de este informe se aborda, más adelante, la descripción de aquellos componentes utilizados en nuestro desarrollo.

Para incorporar al proyecto el uso de componentes de JetPack, se agrega en el archivo de configuración de Gradle (build.gradle), el repositorio google().

Descripción funcional del entorno

El entorno permite a un estudiante transitar un recorrido de aprendizaje a través de una secuencia de unidades temáticas. Cada unidad se compone de una serie de actividades que el alumno debe abordar de forma autónoma. Estas actividades pueden ser de tres tipos:

- Recursos: contenidos hipermedia (documentos, videos, gráficos, páginas web, actividades interactivas, etc.) que se presentan al alumno para su lectura o visualización. El objetivo es que el alumno construya conocimientos en función de la información que se le brinda en estos recursos.
- Test: son actividades de evaluación implementadas en la plataforma Moodle (cuestionarios, selección múltiple de opciones, juegos, etc.).
- Pista: son ayudas a las que accede el estudiante en caso que desapruebe un test, con el objeto de ayudarlo a comprender los contenidos cuyo aprendizaje debe validar.

Cada unidad de aprendizaje debe incorporar al menos un recurso, un test, una pista y un test de recuperación, en ese orden. A criterio del equipo docente que diseñe el recorrido podrán incorporarse más actividades si se desea. 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 trabajo en el entorno. Para interactuar con el sistema el estudiante utiliza la App provista como front-end del entorno.

El alumno puede elegir, seleccionando puntos en un mapa, las ubicaciones geográficas donde quiere realizar sus actividades. A posteriori, cada vez que el alumno ingrese o transite dentro de un radio de *m* metros en torno a estos puntos se le ofrecerán contenidos u actividades a realizar de forma ubicua. En adelante, llamaremos a cada uno de estos puntos, zonas de interés o geo-vallados.

En particular, el alumno interactúa con el entorno del siguiente modo:

1) La primera vez que ingresa a la App se identifica en el entorno completando un formulario de login. Debe ingresar el usuario y contraseña que utiliza para loguearse en el aula virtual de la Plataforma Moodle. La App realiza las validaciones correspondientes contra el entorno y la plataforma Moodle para garantizar este proceso de login.
2) Una vez identificado en el entorno, elige sus zonas de interés o áreas de geo-vallado. Debe luego seleccionar tantos puntos en el mapa como unidades temáticas contenga el diseño propuesto para el recorrido. La App realiza las validaciones correspondientes para garantizar que se seleccione el número adecuado de puntos.
3) Una vez registrados los puntos en el entorno, un servicio administrado por la App y permanentemente activo, se encarga de geo-localizar el móvil y enviar una notificación cuando se ingrese a alguna de las áreas de geo-vallado.
4) De forma ubicua en la zona de interés detectada, el entorno habilita el acceso a las actividades de aprendizaje vinculadas a la unidad temática que corresponda. A la primera área de geo-vallado detectada se le asigna dinámicamente la primera unidad temática, a la segunda área la segunda unidad y así sucesivamente.
5) Cuando se habilita una unidad temática, deberá:
    a) Leer los contenidos provistos como recurso y confirmar su lectura.
    b) Completar el test Moodle que se habilita de forma automática en la App. El entorno validará además que cada test se habilite de forma personalizada para cada alumno en el lugar y momento en que el estudiante confirma haber leído los recursos y/o pistas previas que corresponda. El test es evaluado de forma automática por la plataforma Moodle y en función del resultado el entorno resuelve:
6) Si aprueba, se completa la unidad de aprendizaje.

        (1) ii. Si desaprueba, se pasa a c.

    b) Visualizar la pista que le brinda el entorno y confirmar su lectura.
    c) Completar el test Moodle que se habilita como recuperación de forma automática en la App cuando se confirma la lectura de la pista anterior. El test de recuperación es evaluado de forma automática por la plataforma Moodle y en función del resultado el entorno resuelve:
        i. Si aprueba, se completa la unidad de aprendizaje y vuelve a activarse el servicio de geolocalización para acceder a la siguiente unidad.
        ii. Si desaprueba, se pasa a la siguiente pista y así sucesivamente se repite el ciclo (puntos c., d.), hasta que todas las actividades incluidas en la unidad temática hayan sido completadas.
7) Una vez iniciadas las actividades de una unidad temática, el servicio de geo-vallado se suspende y no vuelve a activarse hasta que la unidad en curso sea completada y se deba avanzar a la siguiente unidad. De este modo, aun cuando el alumno ingrese en un área de geo-vallado, no será detectado ni notificado mientras no haya completado las actividades de la unidad en curso. En el caso que la actividad sea un Test de Moodle, se verificará contra la BD de Moodle que el alumno haya rendido el test antes de permitirle avanzar a la siguiente actividad.
8) Cada unidad temática incluida en la secuencia de aprendizaje se completa, cuando se da alguna de las siguientes condiciones:

a) Unidad aprobada: se aprueba alguno de los test de Moodle correspondiente a la unidad.

b) Unidad desaprobada: se accedió a todos los recursos y pistas de la unidad, y se desaprobaron todas las instancias de evaluación previstas para la unidad.

9) Cuando una unidad temática se completa, ya sea en condición "aprobada" o "desaprobada", vuelve a activarse el servicio de geolocalización para ofrecer la siguiente unidad temática al alumno, de forma ubicua, cuando este sea detectado en la siguiente zona de interés por la que circule.

10) Cuando completa la última unidad temática incluida en la secuencia de aprendizaje, se habrán transitado todas las áreas de geo-vallado y el recorrido llega a su fin.

Conclusiones

Se implementó un entorno de enseñanza ubicuo, el cual es soportado mediante servicios de cloud-computing.

Se han integrado tecnologías de computación móvil, realidad aumentada, geo-referencia, bases de datos y entornos virtuales de enseñanza, utilizando webservices para el acceso a los diferentes componentes del sistema.

Se ha validado satisfactoriamente el funcionamiento general de la herramienta a través de la interacción con Moodle, quedando registrados los resultados de la interacción del alumno con el entorno.

Reconocimiento

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.

Referencias

Chen, Y. K. (2002). A Mobile Scaffolding-Aid-Based Bird. Watching Learning System, Proceedings of IEEE International Workshop on Wireless and Mobile Technologies in Education (WMTE'02), (págs. 15-22).

Gao H, Z. Y. (2010). System design of cloud-computing based on mobile learning. In Proceedings of the 3rdInternational Symposium on Knowledge Acquisition and Modeling (KAM), (págs. 292-293).

Guía de Arquitectura de Apps https://developer.android.com/jetpack/guide?hl=es-419

Li, J. (2010). Study on the development of mobile learning promoted by cloud-computing. In Proceedings of the 2nd International Conference on Information Engineering and Computer Science (ICIECS), (pág. 1).

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, de https://techpack.acm.org/cloud/cloudcomputing.pdf

Nidal M. Turab, A. A. (2013). Cloud-computing Challenges and Solutions. International Journal of Computer Networks & Communications (IJCNC), Vol.5, Nro.5.

Oracle. (2016). Five Ways to Simplify Cloud Integration - Oracle Integration Cloud Service. Recuperado el 15 de noviembre de 2020

World Wide Web Consortium, 3.1.3. (2004). "Web Services Architecture". Relationship to the World Wide Web and REST Architectures, 11 February 2004. Recuperado el 29 setiembre de 2020

[1] 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, de https://techpack.acm.org/cloud/cloudcomputing.pdf

[2] Nidal M. Turab, A. A. (2013). Cloud Computing Challenges and Solutions. International Journal of Computer Networks & Communications (IJCNC), Vol.5, Nro.5.

[3] Oracle. (2016). Five Ways to Simplify Cloud Integration - Oracle Integration Cloud Service. Recuperado el 15 de diciembre de 2018

[4] Gao H, Z. Y. (2010). System design of cloud computing based on mobile learning.In Proceedings of the 3rdInternational Symposium on Knowledge Acquisition and Modeling (KAM), (págs. 292-293).

[5] Li, J. (2010). Study on the development of mobile learning promoted by cloud computing. In Proceedings of the 2nd International Conference on Information Engineering and Computer Science (ICIECS), (pág. 1).

[6] Möller, P. H. (2013). Ubiquitous Learning: Teaching Modeling and. In Proceedings of the 2013 Grand Challenges on Modeling and Simulation Conference (pág. 24). Society International for Modeling & Simulation.

[7] 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.