

Universidad Nacional de La Plata

Facultad de Informática



Mi Universidad: Una aplicación móvil
para mejorar la experiencia de usuario
de los estudiantes de la Universidad
Nacional de La Plata

Tesina de Licenciatura en Informática

Luciano Agustín Coggiola

Director: Lic. Francisco Javier Díaz

Codirectora: Lic. Ana Paola Amadeo

Asesor Profesional: Lic. María Alejandra Osorio

29 de agosto de 2017

Índice

Introducción	1
Objetivo	2
1. Capítulo I: Contexto	3
1.1. Análisis sobre los Sistemas Operativos Móviles	3
1.2. Características en aplicaciones para el ámbito universitario . .	4
1.2.1. Novedades	4
1.2.2. Planificación estudiantil	5
1.2.3. Geolocalización	6
1.2.4. Presencia móvil de la Universidad	6
1.3. Aplicaciones universitarias existentes	7
1.3.1. Informática UNLP	7
1.3.2. Jursoc UNLP	8
1.3.3. UNLP: ART Salud	9
1.3.4. Kurogo (Modo Labs)	10
1.3.5. Universidad de Harvard	11
1.3.6. Universidad de Oxford	12
2. Capítulo II: Análisis y marco teórico	13
2.1. Introducción	13
2.2. Software libre y de código abierto	13
2.2.1. GNU <i>General Public License</i> versión 3	15
2.3. Infraestructura del <i>backend</i>	16
2.3.1. Sistemas distribuidos	16
2.3.1.1. Servicios Web	17
2.3.1.2. SOAP	17
2.3.1.3. REST	19
2.3.1.4. SOAP o REST: Conclusión	20
2.3.1.5. Diseño de una API RESTful	20
2.3.2. Autorización y autenticación	23
2.3.2.1. OAuth2	23
2.3.2.2. API Key y API Secret	24
2.3.3. Lenguaje y Framework	25
2.3.3.1. PHP	26
2.3.3.2. Lumen	27
2.4. Infraestructura del <i>frontend</i>	30
2.4.1. Interfaz de usuario	30
2.4.1.1. Patrones de navegación primaria	31

2.4.1.2.	Navegación persistente	31
2.4.1.3.	Navegación transitoria	34
2.4.1.4.	Patrones de navegación secundaria	37
2.4.1.5.	Elección de patrones de navegación	38
2.4.2.	Aplicación móvil	38
2.4.2.1.	Aplicaciones nativas	39
2.4.2.2.	Aplicaciones híbridas	40
2.4.2.3.	Apache Cordova	42
2.4.2.4.	Ionic	43
2.5.	Análisis de algunos servicios de la UNLP	43
2.5.1.	SIU Guaraní	44
2.5.2.	Moodle	46
2.5.3.	Otros servicios	47
3.	Capítulo III: Desarrollo	49
3.1.	Funcionalidad	49
3.1.1.	Noticias	49
4.	Capítulo IV: Conclusión	50
4.0.2.	Trabajos futuros	50
Glosario		56

Introducción

El avance tecnológico de los dispositivos móviles ha ido creciendo rápidamente en los últimos años. Cada año nace una nueva generación de smartphones y tabletas incorporando nuevas tecnologías (GPS, lector de huellas, notificaciones, acelerómetro, cámara, etcétera). Estas permiten obtener mayor información para adicionar nuevas funcionalidades, mejorar la experiencia de usuario y potenciar la comunicación. De este modo, las aplicaciones móviles (apps) permiten establecer un buen canal de transmisión entre una entidad y el usuario o entre los distintos usuarios (que comparten un interés en común).

Los dispositivos móviles son formadores de hábito en sus usuarios: existe una tendencia a revisar repetitivamente por períodos cortos, contenido dinámico, fácilmente accesible desde el celular[35]. Estos hábitos motivan al usuario a realizar otras tareas con el teléfono, incrementando el tiempo total de su uso.

Para este trabajo, referiré como “entidad” a la Universidad Nacional de La Plata (UNLP), sus dependencias y subdivisiones, dentro de las cuales en sus sistemas guardan una gran cantidad de información sobre las personas que allí desarrollan sus actividades. Dentro de ese conjunto de datos figuran los pertenecientes a los alumnos, población universitaria a la que apuntaré y voy a considerar “usuarios” .

Según un estudio realizado por Google sobre el uso de *smartphones* en Argentina, [20] “la penetración de los teléfonos inteligentes actualmente alcanza al 24 % de la población, y sus propietarios dependen cada vez más de sus dispositivos. El 71 % de estos usuarios accede a Internet todos los días desde su teléfono inteligente, y casi nunca sale de su casa sin llevarlo.”. Se hace evidente el hecho de que los dispositivos inteligentes se han convertido en un accesorio indispensable para la vida cotidiana, y su aceptación es socialmente masiva, por lo que considero que el desarrollo de una aplicación que comunique a la “entidad” con los “usuarios” será útil para mejorar la experiencia de usuario, por parte de los alumnos, el acceso a la información de los servicios de la Universidad, así como también fomentará la presencia de la misma.

Además, al utilizarse sistemas abiertos e implementados por todas las Universidades de la Argentina, esta solución podrá expandirse hacia ellas.

Por lo expuesto se hace evidente la necesidad de uso de las tecnologías

arriba mencionadas, para el desarrollo de una aplicación que mejore la comunicación entre usuario-usuario y usuario-entidad para mejorar su experiencia.

Objetivo

Esta tesina se centra en el desarrollo de una aplicación móvil que permita integrar múltiples servicios de la Universidad Nacional de La Plata (en particular para esta Tesina: SIU Guaraní y Moodle), con el objetivo de mejorar la experiencia de usuario de los estudiantes, potenciando las posibilidades de comunicación y colaboración entre ellos, la Universidad y sus dependencias, la sociabilización de contenidos y la presencia de la Universidad.

De ello se desprenden los siguientes objetivos específicos:

- Analizar aplicaciones universitarias existentes en Argentina y el mundo.
- Analizar las herramientas existentes para llevar a cabo el desarrollo de una aplicación multiplataforma sobre dispositivos móviles, aprovechando las posibilidades tecnológicas que estos poseen.
- Analizar la integración de la aplicación móvil con los distintos servicios Web que brinda la Universidad.
- Desarrollar una aplicación móvil que permita comunicar y representar la información obtenida de los distintos servicios.
- Desarrollar la integración de la aplicación móvil con SIU Guaraní, contribuyendo a la comunidad de desarrolladores que implementan este sistema en todas las Universidades Nacionales.
- Desarrollar la integración con Moodle aportando la posibilidad de conexión para quienes utilicen este sistema *open source*.

1. Capítulo I: Contexto

En este capítulo se buscará analizar el contexto sobre el uso de los dispositivos móviles, las aplicaciones universitarias existentes en la actualidad y cuales son sus características principales.

1.1. Análisis sobre los Sistemas Operativos Móviles

En el mercado actual existe una amplia variedad de empresas que venden teléfonos móviles inteligentes.

Desde el punto de vista del desarrollo, el 80 % de aquellos que desarrollan aplicaciones móviles profesionalmente, apuntan a Android como su plataforma primaria. Con el 53 % iOS y 30 % el navegador del móvil[32].

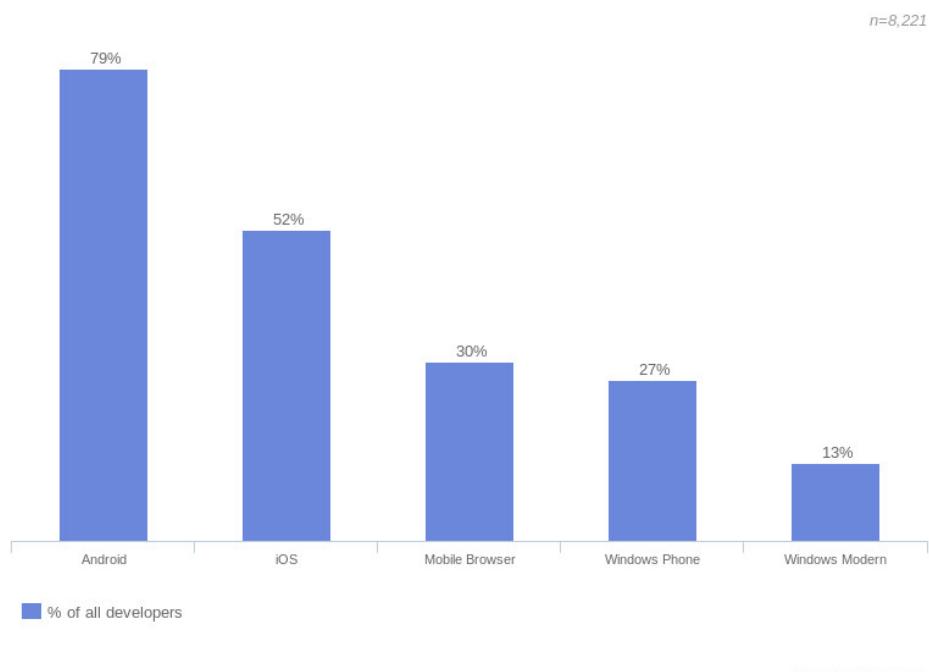


Figura 1: Porcentaje de desarrolladores que eligen como plataforma primaria el sistema operativo.

Particularmente en sudamérica, el porcentaje de desarrolladores móviles

priorizando la plataforma Android es del 38 % de manera profesional profesional, más 18 % como hobby o proyecto secundario. En segundo lugar está iOS con el 15 % profesional y 3 % como hobby o proyecto secundario[32].

Desde el punto de vista del uso, la venta de teléfonos celulares en todo el mundo con Android como sistema operativo representó el 86,2 % del total de las ventas, frente al 12,9 % de iOS y 0,9 % entre Windows, BlackBerry y otros. En Argentina Android representa el 79 % de los sistemas operativos del mercado, Windows 12 % e iOS, 3,5 %[27].

De estos hechos se desprende la necesidad de considerar un desarrollo que permita ser implementado en múltiples plataformas móviles. Se tendrá en cuenta para el alcance de esta tesis, Android, puesto que es el principal sistema operativo móvil del mercado en Argentina[27], además de su importante adopción en el mundo del desarrollo[32].

1.2. Características en aplicaciones para el ámbito universitario

En la actualidad existen muchas aplicaciones para ayudar en cada aspecto de nuestra vida cotidiana, y el ámbito universitario no escapa a ello. En este sentido se revisarán los aspectos característicos de las más útiles[44] para poder analizar funcionalidades interesantes que puedan mejorar la experiencia de los estudiantes y fortalecer la presencia de la Universidad.

Los aspectos a considerar serán tratados en las siguientes secciones.

1.2.1. Novedades

Es importante para el estudiante estar al tanto de las novedades sobre las distintas entidades que intervienen en el transcurso de su carrera universitaria: materias, Facultad, Universidad, biblioteca, etc. Muchas veces, al ser estas tan diversas, sus canales de comunicación también lo son. Es por ello que surge la necesidad de facilitar la unificación del punto de acceso a esta información. Por otra parte, en los usuarios de *smartphones* existe una tendencia a revisar repetitivamente por períodos cortos, contenido dinámico, fácilmente accesible desde el celular[35]. Por estas razones es que se considerará a las “novedades” como uno de los aspectos más relevantes.

Existen múltiples aplicaciones para el manejo de novedades. De hecho, en muchas de ellas es su principal característica y es por ello que en ocasiones

son adoptadas como la herramienta de publicación de noticias.

Tanto para Android como para iOS existen Facebook, Twitter y WhatsApp.



Figura 2: Captura de pantalla: ejemplo de uso de Facebook para la publicación de noticias.

Como principales características podemos destacar la visualización en un lugar común de todas las noticias de nuestro interés (y su actualización con un gesto) y el recibimiento de notificaciones, junto con un indicador visual de la cantidad de novedades.

1.2.2. Planificación estudiantil

La planificación es clave para la vida universitaria de los estudiantes. En este sentido resulta útil una herramienta que ayude a organizar fechas y horarios de cursada, parciales, finales y entregas en un cronograma. Existen aplicaciones para manejar esta información.

En Android está disponible *Timetable*¹: que permite (de manera intuitiva) administrar las tareas, exámenes, permitiendo la sincronización con otros

¹Timetable disponible en Google Play: <https://play.google.com/store/apps/details?id=com.gabrielittner.timetable>

dispositivos. Como característica novedosa, silencia el teléfono en los horarios de clase. Para iOS existe *Class Timetable*² similar a la anterior, pero permitiendo el envío de notificaciones de tareas o vencimientos y mejores visualizaciones para las actividades de la semana.

1.2.3. Geolocalización

La ubicación en cualquier lugar del planeta puede definirse de manera simple a través de coordenadas (latitud, longitud y altitud). El auge de los *smartphones* y tabletas, el costo cada vez menor del acceso a la red y la popularidad de Internet, hace que haya una “mayor aplicación de la tecnología de geolocalización en el ámbito educativo”[14].

Los datos de geolocalización permiten relacionar una información con una posición en el mapa. Esto facilita la visualización y la identificación de patrones sobre zonas geográficas.

Por otra parte, también permite que los “usuarios encuentren a personas con intereses similares situadas en un entorno cercano y entren en contacto con ellas a través de servicios basados en la localización”[14].

1.2.4. Presencia móvil de la Universidad

El término *presencia móvil* refiere al hecho de *estar presente* en los dispositivos móviles (*smartphones* y *tablets*) estableciendo un canal de difusión de contenidos[38].

Uno de los aspectos importantes de tener una *app* (además de tener mejor experiencia de uso que una aplicación Web) es mejorar esta presencia de la Universidad en el ámbito de las aplicaciones y tiendas virtuales de las distintas plataformas móviles. En este sentido, uno de sus principales beneficios es: fortalecer la imagen. La existencia de una aplicación fortalece el conocimiento de la institución y la comunicación con sus usuarios. De esta manera, mejora el compromiso de ellos para con la Universidad[45].

²*Class Timetable* disponible en <https://itunes.apple.com/gb/app/class-timetable/id425121147?mt=8>

1.3. Aplicaciones universitarias existentes

Actualmente existen varias soluciones para dispositivos móviles en el contexto universitario. Tanto aplicaciones que abarcan aspectos puntuales en el proceso de educación, como plataformas para el desarrollo de herramientas universitarias. Se analizarán los aspectos interesantes de las aplicaciones más importantes en el ámbito de la Universidad Nacional de La Plata, Argentina y el mundo.

1.3.1. Informática UNLP



Figura 3: Capturas de pantalla de la aplicación de la Facultad de Informática: *Informática UNLP*.

Desde Mayo de 2016 está disponible la aplicación³ de la Facultad de Informática. Entre sus características principales, esta posee:

- **Cartelera de novedades:** con la posibilidad de recibir notificaciones al momento de una publicación.
- **Aulas y horarios:** le permite al estudiante conocer qué materias están siendo dictadas en ese momento junto con la indicación del aula. Informa también sobre cuales son los días y horarios de las asignaturas

³*Informática UNLP* disponible en <https://play.google.com/store/apps/details?id=ar.edu.unlp.info.infoUNLP>

de su interés, y además posee una característica muy interesante: un escáner de código QR para conocer qué materia se está dictando en el aula escaneada.

- **Información académica e institucional:** comunica al usuario acerca de las carreras y los planes de estudio, el calendario académico y las fechas de los exámenes finales. También posee enlaces a la Facultad en las redes sociales e información de contacto.

1.3.2. Jursoc UNLP

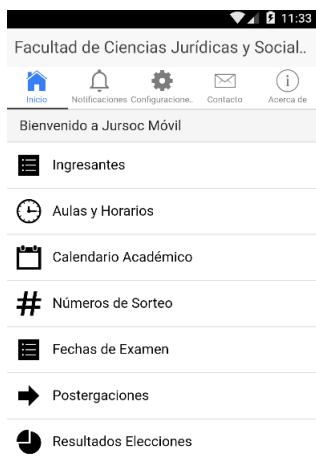


Figura 4: Captura de pantalla de la aplicación de la Facultad de Ciencias Jurídicas y sociales: *JursocUNLP*.

El área de informática de la Facultad de Ciencias Jurídicas y Sociales desarrolló una aplicación móvil⁴ para sus estudiantes. En ella se destacan: la consulta de aulas y horarios de las cátedras, notificaciones sobre cambios en las mesas de examen, los números de sorteo utilizados para la inscripción a cursadas y la geolocalización del edificio para se ubicado mediante el GPS.

⁴*JursocUNLP* disponible en <https://play.google.com/store/apps/details?id=com.jursocunlp.app&hl=es-419>

1.3.3. UNLP: ART Salud



Figura 5: Captura de pantalla de la aplicación *UNLP: ART Salud*.

Se trata de una aplicación⁵ con información útil acerca de la aseguradora de riesgos de trabajo (ART) correspondiente a la Universidad Nacional de La Plata.

Contiene mapas para ubicar las oficinas de la ART, teléfonos útiles y una guía de trámites

⁵ *UNLP: ART Salud* disponible en https://play.google.com/store/apps/details?id=com.mobincube.tramites_art.sc_HS2EQP

1.3.4. Kurogo (Modo Labs)



Figura 6: Capturas de pantalla del menú principal de aplicaciones realizadas con Kurogo.

Kurogo es una plataforma desarrollada por Modo Labs para crear aplicaciones móviles (con sistema operativo Android o iOS) orientadas a la favorecer la comunicación y dar a conocer información acerca de entidades, en especial, universitarias, de manera sencilla. Dicha plataforma está organizada en una serie de módulos (expresados a través íconos con texto en la pantalla principal como se muestran en la figura 6) que representan datos y actividades de interés para el estudiante: servicios que ofrecen, enlaces, eventos, horarios, teléfonos y mapas. Se destacan algunas de sus funciones:

- **Calendario:** Agrupados por las categorías a las que pertenecen, se muestra información de los eventos relacionados a la universidad, con la posibilidad de buscarlos, compartirlos y agregarlos al calendario del dispositivo.
- **Mensajería:** Le permite a los estudiantes recibir notificaciones push y mensajes en forma de avisos en tira⁶.
- **Bibliotecas:** Permite realizar búsquedas de libros y artículos y consultar su información y disponibilidad Mapa: ofrece un mapa completo de

⁶Los avisos en tira son mensajes breves que aparecen en la parte superior de la pantalla.

los edificios internos del campus universitario (de interiores y exteriores). Existe también la posibilidad de realizar búsquedas.

- **Emergencias:** Permite recibir noticias críticas de emergencias y acceder al listado de teléfono útiles.
- **Comedor:** Da a conocer los menús y platos que ofrece el comedor, junto con sus horarios.
- **Estacionamiento:** Este módulo permite recibir información en vivo de los lugares libres para estacionar el automóvil. Requiere de software y hardware extra para permitir esta funcionalidad.

Kurogo es utilizada por Universidades como Colgate University, Harvard y CSUN, entre otras.

1.3.5. Universidad de Harvard

La Universidad de Harvard[24] posee varias aplicaciones móviles para sus estudiantes. La principal, Harvard Mobile, está desarrollada utilizando Kurogo (antes mencionada) y brinda la información básica que provee esta plataforma. Está disponible para iOS, Android y web móvil⁷. También disponen de tres aplicaciones orientadas tours y recorridos virtuales (con diferentes temáticas históricas, culturales y botánicas), y otra relacionada con eventos y noticias de la escuela de salud pública.

⁷Sitio web optimizado para móviles: <https://m.harvard.edu/>

1.3.6. Universidad de Oxford

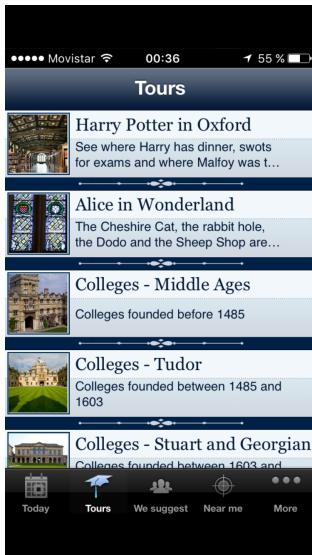


Figura 7: Captura de pantalla de la aplicación oficial de la universidad de Oxford.

Para la Universidad de Oxford[36] existe una aplicación principal (*Oxford University: The Official Guide app*, disponible solo para iOS) que está centrada en mostrar las novedades y los recorridos turísticos focalizados en distintas temáticas como esculturas, jardines y ganadores del premio Nobel, entre otras. Permite también, conocer qué hay en los alrededores y mostrar sugerencias de qué hacer en esa zona.

Existen además otras *apps* secundarias que permiten ver la revista de la institución (*Oxford Today*) y recorrer sus museos (*Explore Oxford University Museums*) y un sitio web adaptado con estilos móviles en el que sus principales funciones son: conocer las tareas diarias, verificar el estado del transporte en colectivo, encontrar un libro de la biblioteca, etcétera.

2. Capítulo II: Análisis y marco teórico

2.1. Introducción

Este capítulo presenta el marco teórico y el análisis de las tecnologías existentes, para fundamentar el desarrollo propuesto y su implementación.

Es importante destacar que para el marco teórico del proyecto propuesto en esta tesina, surge la necesidad de considerar los mecanismos para:

- Brindar un mecanismo genérico para integrar información útil para el estudiante, entre distintos sistemas implementados en la Universidad Nacional de La Plata.
- Proveer nuevas operaciones y que estas estén disponibles para su uso externo, a través de otros sistemas.
- Comunicar datos de interés entre el *backend* y los dispositivos móviles.
- Representar estos datos en una aplicación móvil, de manera sencilla y que el usuario encuentre cómodo al momento de su utilización.
- Expresar la libertad de uso, copia y modificación del desarrollo.

En primera instancia se revisan las razones del uso de software libre. En segundo lugar, puesto que el modelo de aplicación distribuida elegido es el de cliente-servidor, este apartado continúa con dos secciones. La primera, revisa las características técnicas para proveer los recursos (*backend*, servidor) y la segunda, detalla sobre el consumidor (*frontend*, cliente).

Por otra parte, considerará algunos de los servicios implementados actualmente por la Universidad Nacional de La Plata (y sus dependencias) para lograr su integración. Además, se considerará la potencialidad de incorporación de otros servicios pre-existentes a este desarrollo.

2.2. Software libre y de código abierto

El software libre y de código abierto (denominado *Free and open-source software* (FOSS)) es aquel que reúne ambas características: la de ser *Software libre* y de *código abierto*. Esto significa que cualquiera es libre de usar, copiar, estudiar y cambiar el software, y además que su código es abiertamente compartido para motivar a las personas a que mejoren su diseño[19].

Aunque ambos términos parezcan similares y estén motivados por cuestiones en común, existen diferencias: el *código abierto* se basa en las ventajas que posee este modelo de desarrollo, mientras que el *software libre* es un concepto más filosófico que trata de las libertades de los usuarios respecto de los programas.

El software libre se basa en que los programas deben respetar cuatro libertades. A continuación son enumeradas, citando textualmente a la *Free Software Foundation* en [19]:

La libertad de ejecutar el programa como se desea, con cualquier propósito (libertad 0).

La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.

La libertad de redistribuir copias para ayudar a su próximo (libertad 2).

La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello. [19]

Los beneficios estos ambos principios[34] son:

- Seguridad: Cuantas más personas vean el código, es más probable que detecten errores y los corrijan. Esto tiene un impacto directo en el marco de la seguridad.
- Calidad: En relación con el inciso anterior, la cantidad de usuarios de un desarrollo, también influye, ya que permite que estos incorporen nuevas funcionalidades o las mejoren.
- Personalización: Al permitir modificaciones, habilita a que estas se realicen para adaptarse a las necesidades del usuario u organismo.
- Libertad: La utilización de software de *código abierto* libera el hecho de “estar atado” a una tecnología propietaria.
- Interoperabilidad: Suele adhiere más a los estándares libres que el software privativo, lo que evita estar limitado al uso de formatos cerrados.

- Auditabilidad: La visibilidad del código permite a los usuarios ver las acciones que este ejecuta.
- Opciones de soporte: El soporte es gratis a través de la asistencia de la comunidad de usuarios y desarrolladores. También existe el soporte pago, cuando es requerido asegurarse un mantenimiento.
- Sin costo: por definición es gratis.
- Prueba de producto: Ayuda a evaluar un software antes de utilizarlo.

Estos principios motivan su adopción para el desarrollo de software realizado para esta tesina y como una característica requerida en los componentes que esta utilice.

2.2.1. GNU *General Public License* versión 3

La licencia pública general GNU (*GNU General Public License* (GNU GPL)) fue creada por Richard Stallman para el proyecto GNU. Esta brinda garantías al usuario final para utilizar, compartir, estudiar y cambiar el software. Su objetivo es declarar que los desarrollos que estén bajo esta garantía sean libres y estén protegidos por *copyleft*, evitando que futuras modificaciones por terceros restrinjan las libertades que brinda esta licencia.

En al año 2007 se publicó la versión 3. En líneas generales esta nueva versión incorporó los siguientes cambios[18]:

- Neutralizar las leyes que prohíben el software libre.
- Reforzar la protección contra amenazas de patentes.
- Aclarar la compatibilidad de la licencia con otras.
- Agregar licencias compatibles.
- Habilitar nuevas maneras para compartir el código fuente.
- Distribuir menos código fuente (agregando excepciones para librerías comunes).
- Ajustes para hacer mas global a la licencia.

Para terminar, todo el código fuente escrito para el desarrollo de esta tesina adhiere a la licencia GNU GPL versión 3.

2.3. Infraestructura del *backend*

En la arquitectura del *software* existen muchas capas entre el usuario y el procesador físico que ejecuta las instrucciones. Una de ellas, es la denominada como *backend*. Esta capa es la que se encarga de manejar la lógica de negocios y el almacenamiento, y su ejecución se hará del lado del servidor.

En las siguientes secciones analizaremos las características técnicas requeridas por las necesidades planteadas en la introducción de ese capítulo, que justifiquen las decisiones para este desarrollo (en relación con el *backend*).

2.3.1. Sistemas distribuidos

Al comienzo de la era de las computadoras modernas, entre 1945 y 1985, estas eran muy costosas y de gran tamaño. Además operaban independientemente unas de otras.

A mediados de los años 80, ocurrieron dos avances tecnológicos claves que resultaron en el comienzo de una nueva era para favorecer el desarrollo de los sistemas distribuidos[7].

El primero es el avance en la potencia de los microprocesadores: se produjo un gran aumento del poder de cómputo y a su vez una considerable reducción en su precio. Esto fue tan vertiginoso que, como indica Tanembaum, “si los autos hubieran mejorado a este ritmo en el mismo período de tiempo, un *Rolls Royce* hoy hubiera costado 1 dólar y obtendríamos un billón de millas por galón. Desafortunadamente, también sería probable que tuviera un manual de 200 páginas explicando cómo abrir la puerta”[42].

El segundo avance se trata de la aparición de las redes de alta velocidad: desde las redes de área local, hasta las redes de área amplia, permitieron que miles de computadoras se conecten entre sí. Con velocidades variantes, han ido evolucionando desde unos pocos Kilobits hasta Gigabits por segundo.

En conclusión, estos dos factores hacen que hoy sea posible desarrollar fácilmente sistemas que integren a múltiples computadoras que interactúan a través de redes de alta velocidad. Estos sistemas y su interacción definen un sistema distribuido.

Tanembaum y Van Steen definen a los sistemas distribuidos como “una colección independiente de computadoras que se muestran a sus usuarios como un único sistema coherente”[42, p. 2]. De esta manera especifican que sus principales objetivos son:

- Hacer que los recursos remotos estén disponibles de manera controlada y eficiente.
- Ocultar que procesos y recursos están físicamente dispersos entre computadoras distintas.
- Ser abiertos. Esto significa que el acceso a sus servicios esté establecido por ciertas reglas estándar que definan su sintaxis y su semántica.
- Ser escalables. Esto es que tenga la posibilidad de crecer sin perder calidad en el servicio.

Estos objetivos serán requerimientos primordiales para llevar a cabo la integración de los servicios de la Universidad con la aplicación móvil.

2.3.1.1 Servicios Web

Según la definición de la *Universal Description, Discovery and Integration* (UDDI), los Servicios Web son “aplicaciones modulares, auto-contenidas que tienen interfaces abiertas, orientadas a Internet y basadas en estándares”[11].

En términos generales, son la manera de exponer información y funcionalidad de un sistema a través de tecnologías Web, respetando sus estándares. El uso de estos es clave, ya que reduce la heterogeneidad existente entre sistemas facilitando su integración[2]. Esto nos permite utilizar los estándares Web como medio para comunicar sistemas.

Por estas razones se considera a los servicios Web como interfaz para realizar la comunicación de las aplicaciones de la Universidad. En las siguientes secciones se revisan los distintos tipos de servicios web, describiendo sus principales características.

2.3.1.2 SOAP

Simple Object Access Protocol (SOAP) define un protocolo de envío de información estructurada, con un tipo asociado y de manera descentralizada utilizando como lenguaje XML[6]. Estos datos pueden viajar sobre protocolos de transporte existentes como *HyperText Transfer Protocol* (HTTP) o *Simple Mail Transfer Protocol* (SMTP).

Un mensaje en SOAP consta de un elemento “sobre” (en inglés *Envelope*) que dentro contiene otros dos elementos: un encabezado (*header*) y un cuerpo(*body*). Con esta estructura básica, ya es posible la comunicación.

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP:Header>
        <!-- Contenido del encabezado -->
    </SOAP:Header>
    <SOAP:Body>
        <!-- Contenido del cuerpo -->
    </SOAP:Body>
</SOAP:Envelope>
```

Bloque de código 1: Ejemplo de estructura básica de mensaje XML en SOAP. *Envelope* tiene dos hijos: *header* y *body*.

Por otra parte, SOAP define un modelo que le permite indicar a los receptores del mensaje cómo deben procesarlo. Modela el concepto de *actor* que es quien sabe qué parte del mensaje le corresponde y cual descartar para enviar al siguiente.

Sus servicios permiten especificar un documento WSDL (en formato XML) utilizado para detallar la interfaz de conexión, brindando información al usuario sobre los parámetros y sus tipos de dato.

Las fortalezas de SOAP[37] son:

- Transparencia e independencia respecto del protocolo de transporte.
- El uso de WSDL para describir la interfaz del servicio ayuda a abstraer detalles del protocolo de comunicación y serialización, así como también cuestiones de la plataforma sobre el que está implementado (y el lenguaje utilizado).

Sus debilidades:

- Permite la existencia de problemas de interoperabilidad cuando se filtran tipos de datos nativos o construcciones del lenguaje en la interfaz, atravesando las capas de abstracción.
- Produce un *desajuste de impedancia* que resulta costoso al traducir los datos en formato XML a datos utilizados en lenguajes orientados a objetos.

2.3.1.3 REST

REpresentational State Transfer (REST)[16] es una arquitectura de servicios (cliente-servidor) que se basa en los estándares de la Web. En ella, los datos y las funciones son considerados recursos y estos son accedidos mediante *Uniform Resource Identifier* (URI). Las acciones sobre estos recursos son las definidas por los verbos del protocolo HTTP: GET, POST, PUT, DELETE (entre otros). A las *Application Programming Interface* (API) que implementan la arquitectura REST, se les dicen que son RESTful.

De esta forma, su diseño fomenta que los servicios sean simples, livianos y performantes.

Sus fortalezas son[37]:

- Su sencillez: al utilizar estándares web (HTTP, URI, *Multipurpose Internet Mail Extensions* (MIME), *JavaScript Object Notation* (JSON), XML) definidos por la *World Wide Web Consortium* (W3C) e *Internet Engineering Task Force* (IETF), la infraestructura necesaria para su implementación, es de uso generalizado.
- Servidores y clientes HTTP están disponibles para la mayoría de los lenguajes de programación, sistemas operativos y plataformas. Además el puerto 80 generalmente se deja abierto en cualquier configuración de *firewall*.
- Infraestructura liviana y económica.
- Fácilmente escalable gracias a que soporta caché, balance de carga y Clustering.
- Permite formatos de mensajes livianos como JSON o inclusive texto plano para tipo de datos simples.

Sus debilidades son:

- Su implementación, al ser abierta, a veces no se adapta al correcto uso de los verbos HTTP.
- Para solicitudes idempotentes (utilizando el verbo *GET*), existe una limitación en el tamaño de la URI de 4KB.

2.3.1.4 SOAP o REST: Conclusión

En base a las ventajas y desventajas de ambas tecnologías, se realizó un análisis[37, p. 809] y se concluyó que es conveniente utilizar REST para integrar servicios a medida a través de la Web y preferir SOAP en la integración de aplicaciones de negocio y que posean como requisito la calidad de servicio.

En base a lo analizado se concluye que la utilización de servicios REST es conveniente para este desarrollo puesto que: provee mayor flexibilidad, es más liviana, sus tecnologías son de uso generalizado y se adapta correctamente a las necesidades de conexión de los dispositivos móviles y de otros servicios externos que consuman la información provista. Además facilita la posibilidad de escalar horizontalmente y permite el uso de caché.

Por otra parte, para la comunicación con aplicaciones móviles, REST supera ampliamente a SOAP. Una evaluación hecha[22] demostró que REST tuvo una mejor *performance* debido a mensajes de menor tamaño y tiempos de respuesta más cortos, además de su alta flexibilidad y bajo *overhead*. La latencia es importante en el ámbito de las aplicaciones móviles ya que existe una pérdida de *performance* y mayor consumo de red al sobrecargar los mensajes con datos extra.

En conclusión, se recomienda la utilización de servicios RESTful[22] para la conexión entre el *backend* y el dispositivo.

2.3.1.5 Diseño de una API RESTful

Para la comunicación con otras aplicaciones utilizando REST es necesario definir una API RESTful que cumpla con sus características[16]. Estas son:

- **Sin estado:** No debe almacenar información de contexto del cliente. Todos los datos que sean necesarios para la comunicación, se envían en cada pedido HTTP, inclusive los datos de autenticación (para el caso de acceso a recursos restringidos). Esto es así para favorecer la escalabilidad y por ende la *performance* general de los servicios: la ausencia de estados en el servidor, elimina la necesidad de sincronizar los datos de sesión entre los distintos nodos[39].
- **Orientada a recursos:** La *arquitectura orientada a recursos* se basa sobre el concepto del *recurso*. Esto significa que cada uno de ellos es un componente distribuido que permite ser accedido directamente y es

manejado a través de una interfaz común estándar[30]. Cómo acceder a ellos y qué acciones realizar, se tratarán en los dos siguientes puntos.

- **Acceso mediante URIs:** En HTTP y en particular para los servicios web REST, la URI es la principal interfaz de manipulación de los datos. Por lo tanto, para ser correcta, debe ser auto-descriptiva: que de manera intuitiva se pueda predecir, o al menos saber dónde buscar, el acceso al recurso. Una manera para lograr esto es definiendo URIs en base a una estructura jerárquica (de forma similar a la organización de los directorios). De esta forma, existe una rama principal y de ella heredan sub-ramas que van exponiendo cada uno de los servicios. Se utiliza como carácter de separación la “barra inclinada” .
- **Uso de métodos HTTP:** Las acciones que se realicen sobre los recursos deben ser representadas a través de los “verbos HTTP” (definidos en el estándar como *métodos HTTP*). Existe una asociación directa entre CRUD y las operaciones HTTP[16]. Además, cada una de ellas tiene una semántica asociada que implica un comportamiento implícito en la API (pero bien definido en el estándar HTTP). Por ejemplo el método GET es idempotente, lo que significa que no importa cuantas veces sea invocado, siempre retornará los mismos resultados y no producirá una modificación explícita (esperada por el usuario) en el sistema.
- **Representación de los recursos en XML o JSON:** la representación de un recurso refleja el estado de este y sus atributos al momento de su solicitud. La manera en que esta información es codificada es lo que llamamos formato. Comúnmente se utilizan dos formatos XML y JSON. Ninguno es superior al otro y su elección depende de las necesidades del sistema.

En particular, para la presente solución, que no requiere la existencia de datos complejos, se hace el foco en JSON. Este es más simple (legible) y compacto (menos *overhead*) y por ende su transferencia en la red es más rápida. Su uso en soluciones que utilizan JavaScript y AJAX es más eficiente y flexible[31].
- **Estado de respuesta utilizando códigos HTTP:** estos códigos son devueltos siempre en una solicitud HTTP y están definidos dentro del estándar[15]. Si bien no indican demasiado en dentro del dominio de aplicación, se vuelven útiles cuando son utilizados por un destinatario o intermediario genérico (como cachés, proxies o librerías), que comprende el protocolo y sabe cómo actuar ante ciertas respuestas.

- **Versionable:** el versionado no está dentro de las recomendaciones o de la especificación, de hecho va en contra del “purismo de REST” y es muy discutida la forma de llevarlo a cabo. Sin embargo, existe una realidad: como el *software* cambia, también lo hacen sus APIs, por lo tanto es un aspecto importante a destacar en su diseño.

A medida que el software evoluciona surge la necesidad de incorporar estos cambios en los servicios. Si estos se hicieran y no se tomase ningún recaudo, se estarían generando incompatibilidades en sistemas que dependen de estos servicios. Para evitar estos problemas, surge la necesidad del versionado de APIs.

Como indica Hunt en su artículo[26], existen tres maneras “incorrectas” de versionar APIs:

- Por *Uniform Resource Locator* (URL): Incorporando el número de versión en la URI.
- Encabezado personalizado: Se crea un encabezado HTTP personalizado indicando la versión de la API del *request*.
- Encabezado *Accept*: Similar al punto anterior, pero este está definido en el estándar sólo que se le incorpora el dato de la versión.



Figura 8: *Meme* que bromea acerca de las discusiones en Internet sobre el versionado las interfaces REST

2.3.2. Autorización y autenticación

Al definirse la utilización de servicios REST para la comunicación, el siguiente paso es determinar qué mecanismos de seguridad existen, que sean compatibles y utilizados por la industria, para restringir el acceso a los recursos.

Se denomina *autenticación* al proceso de determinar que algo o alguien es quien dice ser y *autorización* a la verificación de los permisos necesarios para acceder a un recurso[12].

El desarrollo propuesto requerirá que se verifique el acceso a la información de:

- los usuarios a través de la aplicación móvil y
- de las aplicaciones y servicios externos que quieran integrar su información.

Para ello se revisarán los mecanismos que más se adecuen para cada caso.

En el siguiente apartado, se investigan las tecnologías para realizar la autorización y autenticación de las diferentes partes intervenientes.

Para la interacción entre:

- la aplicación móvil y la API: se utiliza el protocolo estándar denominado OAuth, en su segunda versión (OAuth 2.0).
- aplicaciones externas y la API: se adopta el mecanismo de clave de API y clave secreta.

A continuación se detallan las características de cada una.

2.3.2.1 OAuth2

El protocolo OAuth2 permite verificar que las aplicaciones tengan los permisos necesarios para el acceso a los datos y operaciones en nombre del usuario, sin estas conocer sus credenciales (nombre de usuario y contraseña, por ejemplo)[23].

Este mecanismo además, posibilita la integración entre las aplicaciones sin compartir o guardar los datos de acceso. Para ello se vale del concepto

de *tokens* de acceso de manera que el usuario no tenga que ingresar la clave en aplicaciones de terceros. Este método sirve para autenticar y autorizar, por un tiempo limitado, cada solicitud. Desde el punto de vista de la seguridad, esto es útil, ya que si un atacante se hace del *token* de acceso, este expirará reduciendo el potencial de daño.

El tipo de acceso elegido para la interacción “aplicación móvil-API” es el denominado *Resource Owner Password Credentials Grant* ya que existe confianza entre la aplicación y el “dueño del recurso”, que en este caso es la API. Este tipo de otorgamiento es apropiado para clientes que son capaces de obtener las credenciales del dueño de los recursos[23]. En la figura 10 se puede ver el flujo de los datos desde el cliente hasta la obtención del token para este tipo de acceso.

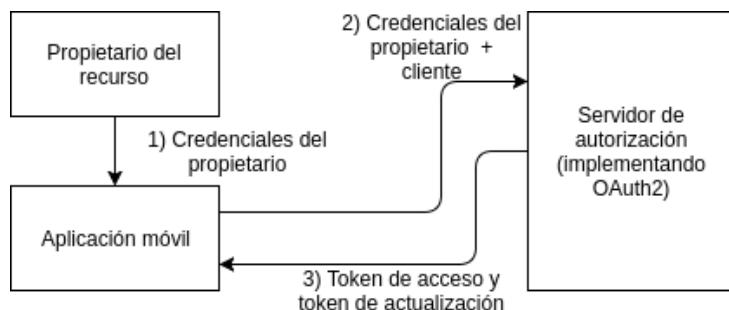


Figura 9: Flujo de información en el proceso de autenticación por *Resource Owner Password Credentials Grant* de OAuth2

Oauth2 es ideal para asegurar APIs ya que permite autenticar usuarios desde dispositivos sobre los cuales no se tiene confianza[13].

Para el caso de esta tesina, se utiliza OAuth2 para la interacción de la aplicación móvil con la API RESTful, manteniendo los requisitos que esta impone.

2.3.2.2 API Key y API Secret

Cuando no se requiere acceder en nombre del usuario, o realizar acciones no directamente relacionadas con él, OAuth no es necesario. Sin embargo se desea controlar la identidad del sistema que está intentando acceder a la información.

Una forma de realizar esto es a través de claves de API. Este método consta de dos de ellas: una podría ser pública e identifica a la aplicación

del tercero (*API key*). La otra es secreta y no debe exponerse (*API secret*). Ambas funcionan como una suerte de “usuario y clave” que son enviadas en cada solicitud HTTP. Sin embargo el valor secreto nunca es viaja en ninguna comunicación realizada.

La forma de autenticación en cada pedido es la siguiente:

- La *API key* identifica a una aplicación y puede ser conocida por otras entidades.
- La clave secreta sólo es conocida por la entidad emisora y la aplicación externa.
- Se procesa la solicitud a enviar, y se adjunta en el encabezado la identificación de la aplicación (*API key*) y la firma. Esta se genera en base al resultado de procesar a través de una función de *hash* (denominada Código de autenticación de mensajes en clave/*hash* (HMAC)) el contenido del pedido y la clave secreta. El resultado de esta función asegura que no se pueda manipular el mensaje, y permite determinar que este es originado por quien dice ser.
- La solicitud es procesada por el destinatario realizando el mismo proceso: aplicando la función de *hash* sobre el contenido y la clave secreta... si el resultado (firma) es el mismo, significa que la solicitud fue originada por el remitente esperado y no fue modificada.

Este método (HMAC) permite identificar al remitente que origina la solicitud y su autenticidad[28] y por ende se pueden controlar (a nivel de aplicación) sus permisos asociados. Este método es utilizado por *Amazon Web Services*[3], *Facebook*, *Telegram* entre otros.

En particular se utiliza el encabezado HTTP *Authorization* para transmitir ambos valores (*API key* y firma).

2.3.3. Lenguaje y Framework

Para el presente desarrollo, en lo que respecta al *backend*, queda por definir el lenguaje sobre el cual se construyen los servicios. Además, para no “reinventar la rueda” se elige un Framework que cumpla con las necesidades de REST.

2.3.3.1 PHP

PHP: *Hypertext Preprocessor* (PHP) es un lenguaje de programación, Open Source, que se ejecuta del lado del servidor y es utilizado generalmente para el desarrollo Web.

En su uso tradicional, al recibirse una solicitud HTTP a un recurso manejado por un *script* PHP, el servidor Web interpreta este código y genera una respuesta. Posee además una interfaz por línea de comandos que permite ejecutar *scripts* desde la consola sin la necesidad de un servidor Web o un cliente HTTP. Este tipo de ejecución se utiliza habitualmente para la programación de tareas de rutina, procesamiento de texto y para ejecutar comandos en Frameworks[1].

Algunas de sus características destacables son:

- Es multiparadigma ya que soporta programación imperativa, funcional, orientada a objetos y procedural.
- Funciona en los principales sistemas operativos: Linux, variantes de Unix, Windows, Mac OS X, RISC OS, entre otros.
- Es compatible con un amplio espectro de bases de datos y soporta su abstracción mediante librerías nativas como PDO.

Es uno de los lenguajes para la Web, del lado del servidor, más utilizados[46], con una gran comunidad de desarrolladores y es adoptado por grandes sitios como: Facebook, Wikipedia, Flickr, Yahoo, Tumblr y Wordpress.com, entre otros.

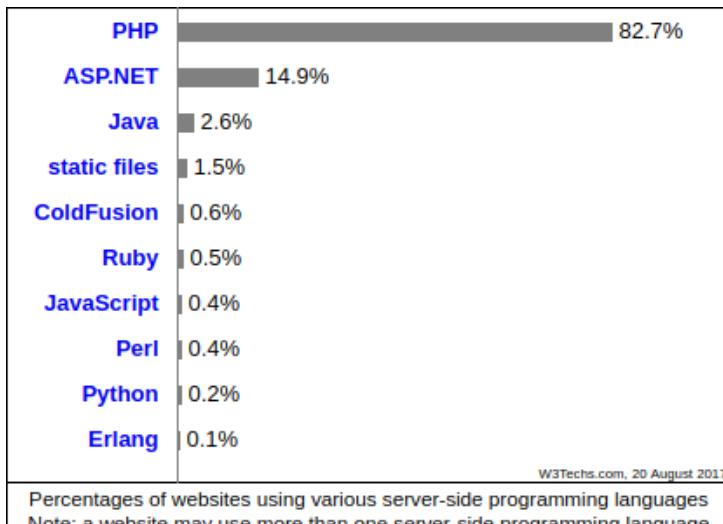


Figura 10: Porcentaje de uso de lenguajes del lado del servidor. Fuente: w3techs.com [46]

Por estas razones y su amplio uso en aplicaciones Web (manejando solicitudes HTTP) lo convierten en una alternativa útil para la implementación de una API RESTful. Además, se elige este lenguaje por mi experiencia en su uso. La versión requerida es 5.6 o superior.

No obstante, hace falta el uso de un Framework PHP que provea las herramientas adecuadas para implementar REST. Analizaremos este aspecto en la siguiente sección.

2.3.3.2 Lumen

Lumen es un Microframework ultra rápido que deriva de Laravel. Está orientado al desarrollo de APIs RESTful.

En sí es una versión de Laravel con funcionalidad removida para incrementar su *performance* y velocidad. Se eliminan aspectos que no se utilizan (o no deberían) en APIs, como por ejemplo el manejo de sesiones o plantillas *HyperText Markup Language* (HTML). Permite en cualquier momento incorporar funcionalidad y pasarse a un proyecto Laravel.

Entre sus principales características se destacan:

- Es de código abierto.

- Su sintaxis es intuitiva.
- Posee una extensa comunidad de desarrolladores dispuestos a resolver los problemas que surjan.
- Bien documentado⁸.
- Completo en recursos: tiene una amplia variedad de extensiones.
- Los validadores.

Cabe destacar que la utilización de Laravel como Framework PHP ha ido incrementando en los últimos años. Las tendencias de búsqueda en Google muestran el interés en la comunidad[43] respecto de otros principales entornos (ver figura 11).

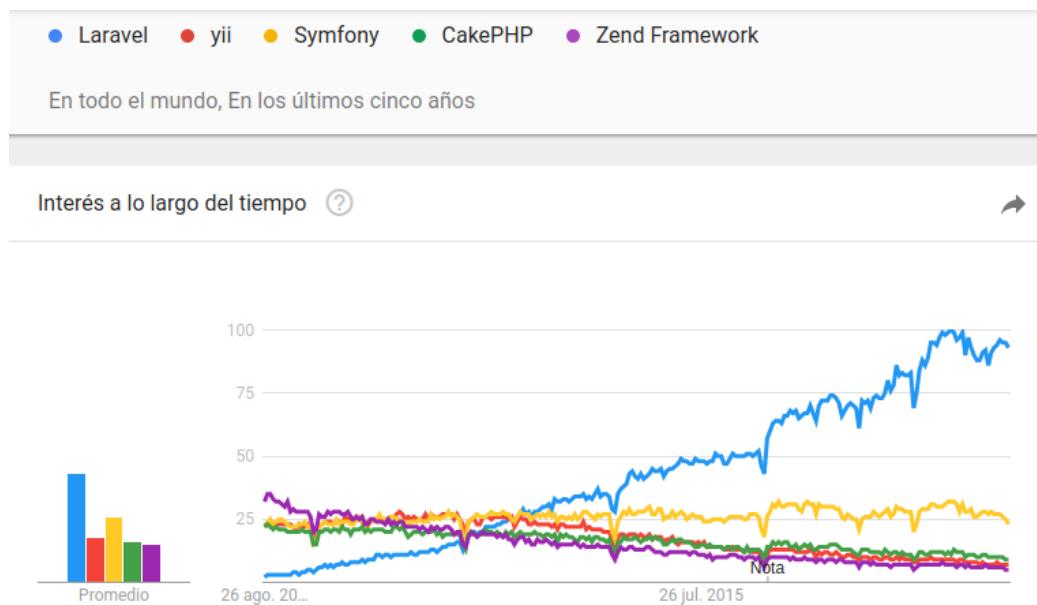


Figura 11: Tendencia en búsquedas sobre los principales Frameworks PHP en los últimos 5 años. Fuente: Google Trends[43]

De acuerdo a un análisis realizado en ambientes productivos por el autor del artículo[5], los tiempos de respuesta y el margen de escalabilidad del

⁸Uno de sus principales sitios es <https://laracasts.com/>

Framework, fueron muy satisfactorios. Las pruebas se realizaron sobre una cantidad aproximada de 50.000 solicitudes por minuto, y se logró tener respuestas de 6 milisegundos para configuraciones de PHP con HHVM y de 25 milisegundos utilizando PHP-FPM.

La infraestructura utilizada constó de 3 nodos con servidores web Nginx y PHP-FPM (cada uno con seis núcleos de 2.6 GHz, 32GB de memoria RAM, y 2 discos de 128GB de estado sólido), balanceados con un HAProxy utilizando *round-robin*. Los resultados indicados se pueden ver en la figura 12.

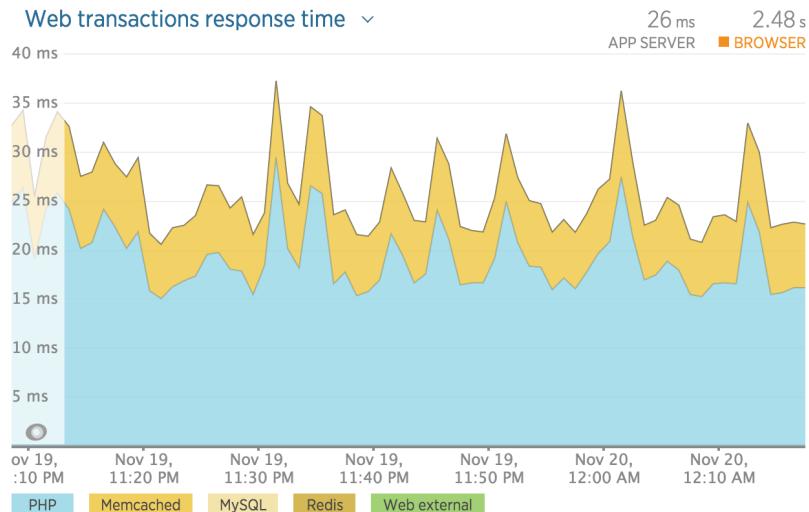


Figura 12: Tiempo de respuesta de las transacciones web con Lumen para aproximadamente 50k solicitudes por minuto. Fuente: darwinbiler.com [5]

Respecto a la escalabilidad, de acuerdo al gráfico de la figura 13, el autor concluyó que como “en un sistema no escalable, la línea tiende a curvarse hacia arriba”, entonces “inclusive si se incrementen las solicitudes, el tiempo de respuesta no crece exponencialmente, de esta manera no afecta la experiencia del usuario final”[5].

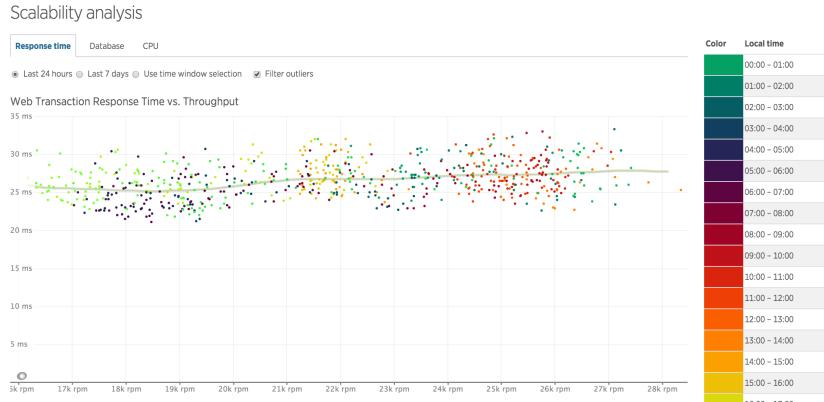


Figura 13: Análisis de escalabilidad de Lumen: tiempo de respuesta vs solicitudes por minuto. Fuente: darwinbiler.com [5]

Finalmente, es importante destacar que *Lumen* soporta (a través de PHP) múltiples motores de bases de datos y el desarrollo se abstrae de las características particulares de ellos. Esto significa que se puede implementar por cualquier motor soportado por *Lumen*⁹, siendo el cambio, totalmente transparente.

Por todas estas razones se elige a *Lumen* como Framework de desarrollo para el proyecto *Mi Universidad*. La versión que se utiliza es la 5.4.

2.4. Infraestructura del *frontend*

Habiendo visto las características requeridas por el *backend*, en la siguiente sección se revisan las relacionadas con el *frontend*. Esto implica estudiar cuestiones relevantes para el cliente (en el modelo *cliente-servidor*), desde decisiones de interfaz de usuario hasta tecnologías para implementar la aplicación móvil.

2.4.1. Interfaz de usuario

Los patrones de diseño para interfaces móviles han ido evolucionando a través del tiempo. La autora Theresa Neil destaca este aspecto comparando

⁹Actualmente soporta: MySQL, Postgres, SQLite y SQL Server. <https://lumen.laravel.com/docs/5.4/database>

los cambios en las ediciones de sus libros [33]: si bien no han habido nuevos patrones de diseño, estos han evolucionado para estar más centrados en dispositivos móviles. Esto significa que dejan de utilizar metáforas relacionadas con aplicaciones Web o de escritorio, para elaborar soluciones centradas en móviles.

En las siguientes secciones se estudian los patrones de diseño para interfaces móviles en base a las necesidades de la aplicación móvil (*frontend*).

2.4.1.1 Patrones de navegación primaria

La navegación organiza el contenido de manera que sea lo más simple posible encontrar un dato buscado. En particular se llama navegación primaria a la orientada a moverse entre las categorías más importantes de información (similar a los menús principales en las aplicaciones de escritorio).

La navegación primaria se divide en dos tipos[33]:

- **Persistente:** las opciones de navegación están siempre presentes: son inmediatamente visibles y claras al abrir la aplicación. Forman una estructura simple como un menú con listado o pestañas.
- **Transitoria:** las opciones de navegación están disponibles durante un breve período de tiempo y deben ser explícitamente mostradas con un *toque* o *gesto*. Surgen de las restricciones en el tamaño de las pantallas de los dispositivos, y la necesidad de mostrar información que no cabe en ellas.

2.4.1.2 Navegación persistente

Los patrones actuales de navegación persistente[33] son:

- **Springboard:** Este patrón consiste en un pantalla dividida por una grilla invisible en donde cada celda posee una opción que actúa como punto de partida para las operaciones principales. Este patrón fue popular en el año 2011 debido a las limitaciones existentes (en iOS y Android) para la visualización de pestañas: solamente se permitían de 3 a 5 elementos. Es por ello que este patrón encontraba la forma de ampliar esta restricción, añadiendo inclusive paginado para incluir más opciones.

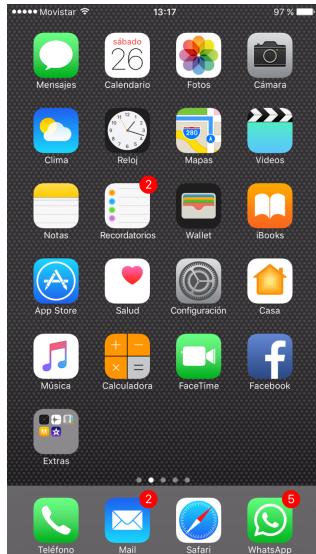


Figura 14: Ejemplo de patrón de navegación *springboard* a nivel de Sistema Operativo en iOS

Su desventaja es que aplana a las opciones, igualándolas en nivel de importancia y no pudiendo reflejar información de herencia entre ellas. Aunque *springboard* no es muy utilizado actualmente en aplicaciones, siguen existiendo implementaciones con algunas variantes que mejoran la interfaz y agregan herencia, como por ejemplo las aplicaciones: EasyJet, Kurogo (ver figura 6), Informática UNLP, entre otras.

- **Tarjetas:** Este patrón es una metáfora de una pila de tarjetas. Permite mostrar la información como si fuera una tarjeta y sus operaciones relacionadas: apilar, mezclar, descartar y ver reverso. Existen variantes como las utilizadas por Instagram y Facebook, en donde algunas de sus operaciones con gestos son removidas en pos de mejorar el estilo.
- **Menú lista:** Similar a *springboard* pero los ítems son ubicados en forma de lista, uno debajo del otro y para cambiar de módulo, es necesario volver a la lista principal.

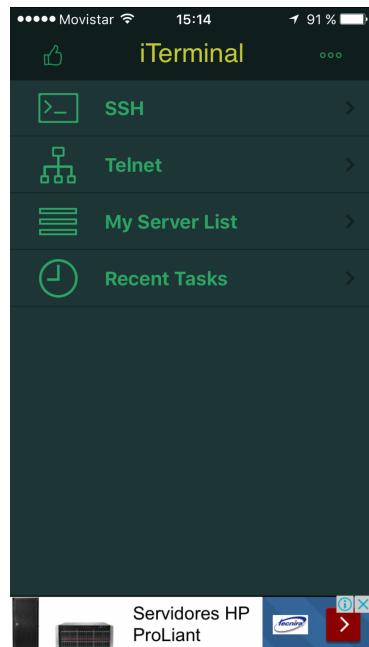


Figura 15: Ejemplo de patrón de navegación de “Menú lista” en iTerminal

- **Tablero:** Similar y combinable con los anteriores, pero brindando (en un vistazo) la información más relevante para el usuario, antes de navegar a la opción deseada.

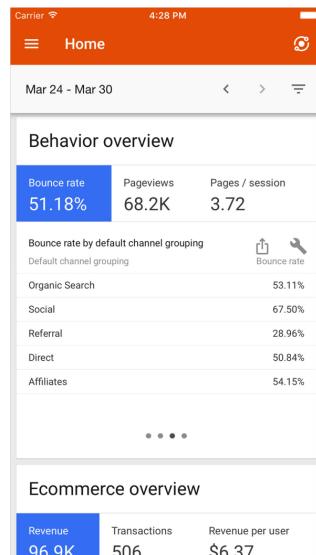


Figura 16: Ejemplo de patrón de navegación *dashboard* en Google Analytics

- **Galería:** Muestra el contenido¹⁰ dentro de una grilla, carrusel o diapositivas.
- **Pestañas:** Recomendadas para una aplicación con una estructura plana, el esquema de pestañas, al estar siempre visible en la pantalla principal, permite navegar entre categorías con tan solo un toque o gesto. Android recomienda[4] el uso de pestañas si:
 - Se espera que el usuario cambie de vistas frecuentemente.
 - Se tiene un número limitado de vistas principales (hasta tres).
 - Se desea que el usuario sea consciente de las vistas alternativas.
- **Skeumórfica:** Este patrón se caracteriza por ser una interfaz que emula a un objeto real y su uso. Aunque no es muy común, algunas aplicaciones se valen de este recurso para mejorar la usabilidad cuando tienen que representar objetos reales. Es muy utilizada en videojuegos, pero no tanto en aplicaciones. Actualmente está en desuso[21].

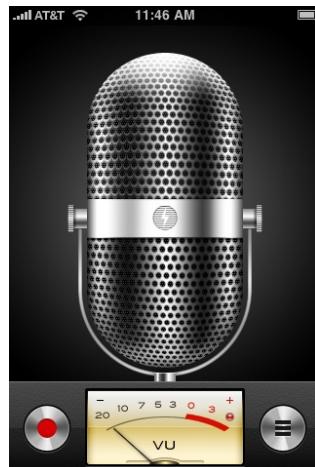


Figura 17: Ejemplo de interfaz con patrón skeumórfico en aplicación de grabadora para iOS

2.4.1.3 Navegación transitoria

Los patrones de navegación transitoria[33] son tres:

¹⁰Notar que esta vez se habla de contenido y no de opciones del menú u operaciones

- **Side drawer** (en español “cajón lateral”): Se trata del componente contenedor del menú que es desplegado desde el lateral de la pantalla hacia el centro. De ahí provine su nombre, ya que es similar al comportamiento de un cajón: oculto, se despliega, utiliza su contenido y es cerrado. Existen dos variantes: el que se muestra superpuesto sobre la pantalla principal y el que desplaza a esta.



Figura 18: *Side Drawer* con menú en aplicación de Netflix (desplaza pantalla principal)

- **Toggle Menu** (en español menú desplegable o desplegado): Es similar al anterior, solo que se despliega desde el borde superior de la pantalla. Una característica clave es que se debe permitir que la misma acción que lo activó, lo pueda desactivar. El menú no debe cubrir toda la pantalla, dejando visible una porción de la vista principal, permitiendo desactivarlo al hacer un toque en esta sección.

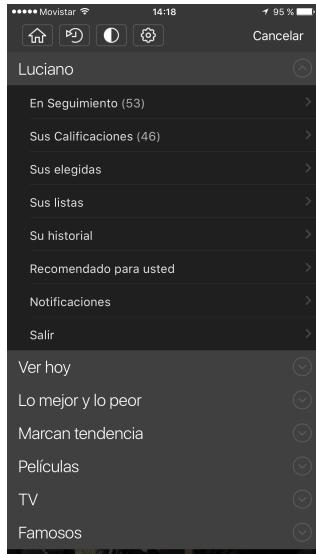


Figura 19: *Toggle menu* con menú en aplicación de IMDB

- **Pie Menu** (en español “menú de torta”, también conocido como “rueda” o “menú radial”): se trata de un menú contextual circular en donde la selección de las opciones depende de la dirección hacia donde se mueve el puntero (o el toque con el dedo). Está dividido en “porciones” (alrededor de un centro) que representan las distintas opciones. Su ventaja se basa en el uso de la memoria muscular para agilizar la interacción[25].

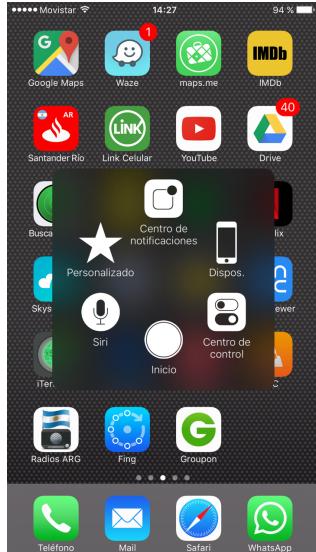


Figura 20: *Pie menu en assistive touch de iPhone*

No obstante, el uso de técnicas de interacción que no son basadas en un puntero (como por ejemplo toques o gestos con el dedo) son problemáticas[29].

2.4.1.4 Patrones de navegación secundaria

La navegación secundaria se centra en el movimiento dentro del módulo seleccionado (por la navegación primaria). Todos los patrones primarios también se pueden utilizar como secundarios, inclusive combinarse, por ejemplo, *pestañas* con *textitgalería*.

Además existen otros tres patrones secundarios más[33]:

- ***Page swiping*** (en español, “páginas deslizables”): se utiliza para navegar el contenido (agrupado en una página) deslizándolo hacia los costados. Para indicar este mecanismo, se recomienda mostrar un indicador visual de las páginas disponibles, por ejemplo una barra con puntos, en donde cada uno representa una página con contenido.
- ***Scrolling tabs*** (en español, “pestañas desplazables”): Es útil para mostrar múltiples categorías o vistas del contenido dentro de un mismo módulo. Pueden ser activadas con un toque o deslizadas en el sentido de la pestaña que se desea acceder.

- **Accordion** (en español, “acordeón”): Consiste en mostrar un resumen o título de cierta información, y al seleccionarlo, se abre, desplazando el resto del contenido contiguo y mostrando el detalle. Le permite al usuario ver más información, manteniéndose dentro de la misma pantalla. Es eficiente en el sentido que, para ver un contenido, evita que el usuario tenga que navegar a otra pantalla siendo probable que luego regrese a ella.

2.4.1.5 Elección de patrones de navegación

En base a lo expuesto anteriormente, se utiliza el patrón de *Pestañas* (persistente) en la navegación primaria para agrupar contenido con las mismas características. Se considera adecuado ya que en la aplicación no hay mas de 3 grupos de contenidos y el usuario se espera que frecuentemente visite todas las vistas. También se utiliza *Side drawer* (transitorio) para mostrar el acceso a otras operaciones de la aplicación y a contenidos específicos de cada servicio asociado. Debido a que estos ítems no son de consulta constante, se considera apropiado este patrón, ya que no es necesario que el menú esté visible todo el tiempo y además le permite a este crecer verticalmente a medida que se suma el contenido de nuevos servicios.

Para la navegación secundaria, dependiendo de la vista, se utilizan los patrones *tarjeta* y *menú lista*.

Debido a que todavía no se han visto los aspectos de funcionalidad de la aplicación, estas elecciones se detallan puntualmente en el capítulo 3.

2.4.2. Aplicación móvil

Al momento de comenzar el desarrollo hay que decidir qué tecnología utilizar para poder crear una aplicación móvil. En la actualidad existen varias, y su diferencia radica en base a las necesidades del desarrollo. En general, ninguna es mejor que la otra, pero en particular, de acuerdo a los requerimientos y necesidades, quizás sea más conveniente una por sobre la otra.

En las siguientes secciones se estudian los distintos métodos, tecnologías y lenguajes para desarrollar aplicaciones móviles y se analiza cuales son los más convenientes para *Mi Universidad*. Este análisis se basa en cumplir con los siguientes requerimientos:

- Que permita la utilización de características del dispositivo como co-

nexión a internet, Sistema de Posicionamiento Global (GPS), notificaciones, calendario, cámara, etc.

- Que sea capaz de brindar una experiencia de usuario de calidad y posibilite la implementación de los patrones de navegación anteriormente vistos.
- Que la aplicación sea potencialmente ejecutada en la mayor cantidad de dispositivos móviles, independientemente de su Sistema Operativo¹¹.
- Que el costo de desarrollo sea bajo: para favorecer el mantenimiento, implementación y adopción de la aplicación por la UNLP y otras Universidades.

Existen dos tipos de técnicas y tecnologías de desarrollo para aplicaciones móviles: nativas e híbridas. Estas se discuten en las siguientes secciones.

2.4.2.1 Aplicaciones nativas

Las aplicaciones nativas son aquellas que para su programación se utiliza el lenguaje específico del sistema operativo del dispositivo y su desarrollo está optimizado para este. Esto significa que la aplicación creada será dependiente de la plataforma, no pudiendo ser directamente portada a otras[10].

Las ventajas de utilizar aplicaciones nativas son:

- Logran un mejor rendimiento: al eliminar una capa de abstracción, su código se compilado y utilizar llamadas nativas, permiten una mayor optimización.
- Mejor *look and feel*: ya que se utilizan los controles desarrollados para la plataforma específica, lo que conlleva a una visualización e interacción pulida.
- Mayor capacidad de procesamiento gráfico y acceso a capacidades del dispositivo: si se desean crear nuevas interfaces, saliendo de lo preestablecido, o implementar juegos, en donde el apartado gráfico es más

¹¹Para el alcance de esta tesis y por cuestiones de costos de licencia, se utiliza Android como principal plataforma, pero igualmente debe existir la posibilidad que la aplicación funcione para otros Sistemas Operativos.

Cuadro 1: Conocimientos de lenguajes necesarios por plataforma[10]

Plataforma	Lenguajes
Apple iOS	C, Objective C
Google Android	Java (Harmony flavored, Dalvik VM)
RIM BlackBerry	Java (J2ME flavored)
Symbian	C, C++, Python, HTML/CSS/JS
Windows Mobile	.NET
Window 7 Phone	.NET
HP Palm webOS	HTML/CSS/JS
MeeGo	C, C++, HTML/CSS/JS
Samsung bada	C++

importante. También tienen una mayor flexibilidad en la utilización de las capacidades del dispositivo.

Como desventajas:

- Dependencia a una plataforma debido a diferencias en los lenguajes, *Software Development Kit* (SDK), herramientas y APIs de cada una de ellas.
- Mayor costo: debido al hecho que portar la Aplicación móvil (App) a otro sistema operativo significa una reimplementación desde cero. Además se multiplican (por cada plataforma) los costos de mantenimiento, al corregir y agregar nuevas funcionalidades. Por ejemplo para las interfaces de usuario, no hay ninguna plataforma implemente los mismos paradigmas.
- Mayor curva de aprendizaje: ya que requiere el conocimiento de todas las características de cada plataforma (lenguaje, APIs, arquitectura, ver cuadro 1.

2.4.2.2 Aplicaciones híbridas

Se trata de aquellas aplicaciones que son desarrolladas utilizando las tecnologías Web (HTML, JavaScript, *Cascading Style Sheets* (CSS)). Esto implica que corran dentro de una contenedor nativo (Apache Cordova/PhoneGap) brindando una capa de abstracción y un acceso homogéneo a las diferentes capacidades del dispositivo (neutralizando las diferencias entre los sistemas operativos).

Las ventajas que tienen las aplicaciones híbridas son:

- Multiplataforma: una misma aplicación permite ser compilada para múltiples sistemas operativos móviles, lo que permite reutilizar el código.
- Menor costo: al realizarse un solo desarrollo y mantenimiento multiplataforma de una sola versión de la aplicación, los costos se reducen.
- Menor curva de aprendizaje: ya que al tratarse de lenguajes y tecnologías Web, se utiliza el mismo paradigma y solo resta conocer cuestiones del Framework de desarrollo. Además estas tecnologías son ampliamente conocidas, por lo que existe una mayor cantidad de recursos humanos.

Como desventajas podemos destacar:

- Menor diseño: ya que a veces no logra simular completamente el *look and feel* de los componentes nativos, además de unificar comportamientos y estilos de interfaz que son diferentes para cada plataforma.
- Menor *performance*: debido a que corren dentro de un contenedor lo que produce tiempos extra de parseo, interpretación y ejecución de scripts en JavaScript.

De acuerdo a los requerimientos establecidos al inicio de esta sección (apartado 2.4.2), se determina que la aplicación a desarrollar sea híbrida. Las razones de esta elección son:

- La aplicación desarrollada permite ser ejecutada en las principales plataformas móviles (multiplataforma).
- Un menor costo que favorece al futuro mantenimiento. En relación con el punto anterior, como el código desarrollado es uno sólo, se simplifica la tarea de corrección de errores e incorporación de nuevas funcionalidades. Por otro lado es más sencillo conseguir recursos humanos capacitados en tecnologías Web que en soluciones nativas móviles. Estas ventajas son clave ya que se ajustan a la realidad la Universidad Nacional de La Plata(y otras universidades).
- En la actualidad, existen Frameworks que posibilitan el uso de todas las capacidades del dispositivo (detallados en la siguiente sección).

Como característica adversa, aunque las soluciones híbridas no sean tan performantes y sus componentes visuales no estén tan pulidos como en las versiones nativas, la funcionalidad a implementar no requiere de una alta *performance*. Por otro lado el Framework elegido simula los componentes muy satisfactoriamente. A continuación, se ven las características de este.

2.4.2.3 Apache Cordova

En el año 2009, Nitobi crea PhoneGap, un Framework que intenta cerrar la brecha entre las plataformas móviles para el desarrollo de aplicaciones. Puesto que cada Sistema Operativo implementa sus APIs y su SDK de manera distinta y en lenguajes distintos, no existe una compatibilidad directa entre ellos. La solución que encuentra Nitobi para cerrar estas diferencias, es el uso de estándares. En particular los que mejor se adaptan para esta solución, son los estándares Web (como HTML5, JavaScript y CSS3). De esta forma, Nitobi desarrolla un contenedor que dota de una capa JavaScript capaz de utilizar funcionalidades nativas en los dispositivos móviles.

La empresa decide que para que el proyecto funcione mejor se necesita una mayor participación de colaboradores. Además esto también produciría un incremento en su uso. Es por ello que en 2011 dona el código fuente del Framework a la fundación Apache, pasando a ser un producto Open Source.

Más tarde ese mismo año, Adobe adquiere Nitobi, adueñándose de la marca PhoneGap. No obstante, esta empresa está de acuerdo con la donación, por lo que el proyecto Open Source sigue adelante.

A principios de 2012, Apache decide renombrar el proyecto para diferenciarlo de PhoneGap y así evitar inconvenientes legales (debido a que se trata de una marca registrada). El nuevo nombre es *Apache Callback* y luego *Cordova*. Este es elegido debido a que las oficinas de Nitobi estaban sobre la calle *Cordova* en Vancouver, Canadá.

En la actualidad, Cordova se define como un Framework móvil Open Source que permite utilizar las tecnologías estándar de la Web como HTML5, JavaScript y CSS3 para el desarrollo multiplataforma, evitando el lenguaje nativo de cada plataforma móvil. Las aplicaciones se ejecutan dentro de contenedores especiales orientados a cada plataforma y dependen de APIs estándares para acceder a las capacidades de cada dispositivo como sensores, datos, estado de la red, etcétera.[17]

Como se indica en la figura ??, la arquitectura de Cordova, consta de

tres componentes:

-
-
-

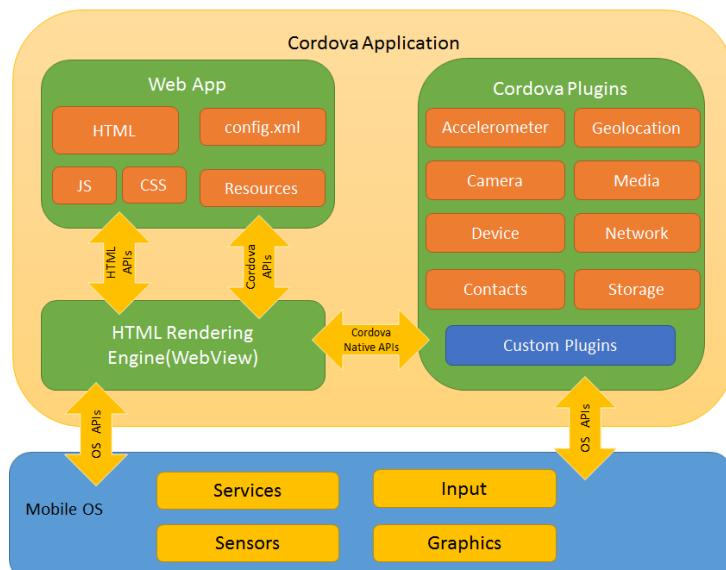


Figura 21: Arquitectura de Cordova. Fuente: Apache Cordova[17]

2.4.2.4 Ionic

2.5. Análisis de algunos servicios de la UNLP

Las siguientes secciones examinan algunos de los servicios que actualmente se implementan en la Universidad Nacional de La Plata y potencialmente permiten su integración con la aplicación desarrollada en le marco de esta tesis.

En la actualidad existen muchos sistemas funcionando en la Universidad y sus dependencias, entre los cuales varios son implementados por el Centro Superior para el Procesamiento de la Información (CeSPI). El criterio para la elección de los sistemas que en esta primer instancia se integran con la aplicación móvil, es debido al alcance y marco de mi trabajo en esa dependencia. Esto no significa que se limite solo a ellos, por el contrario, el objetivo

es establecer una interfaz común y mostrar la posible integración con otros servicios.

2.5.1. SIU Guaraní

El Sistema de Información Universitaria (SIU) tiene sus orígenes en el año 1996 con un préstamo del Banco Mundial. Su objetivo fue la construcción de sistemas que mejoren la calidad y disponibilidad de los datos para mejorar el estado de las instituciones[40]. En 2003 finalizó el financiamiento, y a través del Consejo Interuniversitario Nacional y la Secretaría de Políticas Universitarias se analizaron medidas para mantener y financiar su funcionamiento.

En el año 2007 se conforma un consorcio de Universidades Nacionales al que adhieren casi todas ellas, conformando un grupo colaborativo y favoreciendo con sus aportes al desarrollo grupal.

En 2013 pasa a formar parte del Consejo Interuniversitario Nacional.

El SIU desarrolla el sistema *SIU Guaraní* para todas las Universidades Nacionales que integren el consorcio. Este sistema se encarga de gestionar la actividad académica de los alumnos desde que ingresan a la Universidad hasta que egresan de alguna de sus carreras. Es implementado por 62 instituciones educativas¹² (entre Universidades Nacionales y otras dependencias) y tiene más de 460 instalaciones.

Actualmente el sistema consta de dos partes (denominadas por el SIU como *interfaces*):

- Una interfaz de gestión (a la que referiremos como *gestión*) que es una aplicación de escritorio, utilizada por el personal de la dirección de enseñanza. Esta es capaz de definir Carreras, planes, calendario, mesas de examen, comisiones, cursadas, etc. Está desarrollada con *Power Builder* y funciona para *Windows*¹³.
- Una Interfaz Web (a la que llamaremos *el Web*): es utilizada por Alumnos, Docentes y Directivos (además de un perfil administrador).

¹²De acuerdo a los datos relevados en Agosto 2017. <http://www.siu.edu.ar/listado-de-implementaciones/>

¹³Algunas Universidades la utilizan en Linux con Wine.

Está desarrollada en PHP.

Otro componente importante (en sus versiones 2.x) es su base de datos ya que hace de nexo entre *gestión* y *el Web*. Utiliza Informix y gran parte del funcionamiento (común a ambas interfaces) está implementado a través de *Stored Procedures*.

Su amplia utilización en Universidades de Argentina hace que exista la necesidad de diseñar el sistema considerando cuestiones de funcionamiento específico de cada institución educativa. Por ello a partir del año 2014, en su versión 2.8, se reimplementa su interfaz Web y se incorpora el Framework *SIU Chulupí*.

Esta herramienta es desarrollada por el SIU y sus principales objetivos son mejorar la *performance*, flexibilidad y la capacidad de crear personalizaciones que soporten los cambios de versión. Además se tuvieron en cuenta otros aspectos para mejorar la herramienta y la Usabilidad para alumnos y docentes.

Chulupí está desarrollado en PHP, es basado en el paradigma *Model View Controller* (MVC) y utiliza un conjunto de librerías de código abierto, algunas provenientes de Symfony. Este brinda la posibilidad de realizar personalizaciones (similares a *plugins*), sin la necesidad de modificar código del núcleo, y soporta un esquema de versionado que facilita el mantenimiento del código al publicar nuevas actualizaciones[41].

Actualmente conviven dos versiones de Guaraní: la 2.9.x y la 3.x. En esta última hay un cambio radical respecto de *gestión* y la base de datos. Por un lado se deja de utilizar la aplicación de escritorio para implementarse su funcionalidad desde la Web con el Framework SIU Toba. Por otro, se cambia el motor de base de datos a PostgreSQL. Además se replantea el modelo para incorporar nuevas funcionalidades y que el sistema sea más genérico.

Para la interfaz de Alumnos, Docentes y Directivos, se sigue utilizando Chulupí, y solo cambian cuestiones del modelo, manteniendo la vista y el controlador (en MVC).

En la Universidad Nacional de La Plata, Guaraní se implementa por primera vez en el año 2003 para la Facultad de Ciencias Económicas, llegando al día de hoy, a 23 instalaciones, incluyendo todas las Facultades (para carreras de grado), algunos postgrados y otras dependencias de la Universidad[9]. También existen implementaciones de Guaraní en su versión 3.x.

Para esta tesis, se utiliza el esquema de personalizaciones de Chulupí

para realizar las modificaciones necesarias e integrar aspectos útiles del sistema de alumnos en la aplicación móvil. A su vez, esta personalización se deja disponible a la comunidad de Universidades Nacionales para que también puedan implementar *Mi Universidad*.

2.5.2. Moodle

Moodle es una plataforma de aulas virtuales implementada en PHP que permite a los docentes y estudiantes crear un ambiente de aprendizaje personalizado a sus necesidades. Este sistema es de código abierto, con una gran comunidad de colaboradores que aportan código a la plataforma y creando nuevas extensiones. Su desarrollo es motivado en base a una filosofía que hace hincapié en la “pedagogía construcciónista social”¹⁴.

Moodle es utilizado por más de 100 millones de usuarios en aproximadamente 79 mil instalaciones¹⁵ en múltiples idiomas. Particularmente en Argentina hay alrededor de 1680 sitios registrados de los cuales 9 están relacionados con la Universidad Nacional de La Plata¹⁶. Estos son:

- “Asignaturas de la Facultad Virtual Informática - Ingeniería” en <https://asignaturas.linti.unlp.edu.ar/>
- “AU24 Campus Virtual FCE UNLP” en <http://www.au24.econo.unlp.edu.ar>.
- “AULA VIRTUAL - FCV” de la Facultad de Ciencias Veterinarias en <http://ead.fcv.unlp.edu.ar>
- “Campus virtual de la Facultad de Humanidades y Ciencias de la Educación - Fahce” en <http://campus.fahce.unlp.edu.ar>
- “Campus Virtual Escuela Media 26” en <http://campusem26.fahce.unlp.edu.ar>

¹⁴Para más información sobre su filosofía visitar <https://docs.moodle.org/all/es/Filosof%C3%ADA>

¹⁵Según <https://moodle.net/stats/> en Julio de 2017

¹⁶Sitios registrados y públicos en Argentina, disponibles <https://moodle.net/sites/index.php?country=AR>

- “Cátedras de la Facultad de Informática - UNLP” en <https://catedras.info.unlp.edu.ar>
- “Cursos - LINTI - UNLP” en <https://cursos.linti.unlp.edu.ar>
- “Docentes en línea” de la Facultad de Humanidades y Ciencias de la Educación en <http://intercambioenlinea.fahce.unlp.edu.ar>
- “Entorno virtual de aprendizaje basado en software libre - UNLP” en <https://postgrado.linti.unlp.edu.ar>
- “Facultad de Odontología - Entorno de Enseñanza Virtual” en <http://cursos.folp.unlp.edu.ar/moodle>.

Este sistema permite el desarrollo de extensiones de diferentes tipos de acuerdo a sus funcionalidad (bloques, autenticación, mensajería, etc). Brinda la posibilidad de asociar funciones a eventos definidos, permitiendo de manera flexible extender el comportamiento de su núcleo.

Se utiliza este mecanismo de extensiones para desarrollar la integración entre *Moodle* y *Mi Universidad*. En el capítulo 3 se detallan las características técnicas específicas de su implementación así como también su funcionalidad.

2.5.3. Otros servicios

Existen otros servicios en la Universidad Nacional de La Plata que se pueden integrar y compartir alguno de sus aspectos en *Mi Universidad*. Por ejemplo, el sistema de bibliotecas *Meran*.

Desarrollado por el CeSPI, *Meran* es un “Sistema Integrado de Gestión de Bibliotecas (SIGB) que permite administrar los procesos bibliotecarios y gestionar servicios a los usuarios en forma integrada”[8]. Es de código abierto (licencia GNU GPL v3) y está implementado en varias Facultades, colegios y dependencias de la Universidad Nacional de La Plata¹⁷. Este realiza notificaciones a sus usuarios sobre la disponibilidad en las reservas de libros, o emite avisos que un ejemplar está listo para retirar. Este tipo de información podría integrarse a la aplicación haciendo uso de las notificaciones y novedades personales. También maneja fechas de vencimiento para regresar los

¹⁷Listado de quienes usan *Meran* disponible en http://www.cespi.unlp.edu.ar/articulo/2013/6/14/quienes_usan_meran

libros prestados, pudiendo este dato ser agregado al calendario del dispositivo móvil.

Dando a conocer en la Universidad, la existencia del servicio *Mi Universidad* y su documentación, otros servicios internos a dependencias podrían integrarse y así mejorar la experiencia de los alumnos de la Universidad Nacional de La Plata.

3. Capítulo III: Desarrollo

3.1. Funcionalidad

3.1.1. Noticias

- Globales
 - Con usuarios asociados: Notifica a todos los usuarios que utilizan la aplicación de API (se ignora listado de usuarios específicos ya que la noticia es global).
 - Sin usuarios asociados: Notifica a todos los usuarios que utilizan la aplicación de API.
 - Con contexto asociado: Notifica a todos los usuarios que utilizan la aplicación de API y a los interesados en el contexto.
- No globales
 - Con usuarios asociados: Notifica solamente a los usuarios del listado asociado.
 - Sin usuarios asociados: No notifica a nadie en particular (no debería existir)
 - Con contexto asociado: Notifica a todos los usuarios interesados en el contexto.

4. Capítulo IV: Conclusión

4.0.2. Trabajos futuros

- Mejorar contenidos existentes e incorporar nuevos:
 - Nuevo contenido de tipo “Certificado”: que permita a los servicios asociar certificados, pudiendo ser estos accedidos desde la aplicación, generando un texto y un código QR de validación del mismo. Este código contendría una URL apuntando a un servicio externo que controla su validez.
 - Mejorar el contenido de “Mapas” para que permita mayor interacción de usuario, como puede ser la incorporación de algunos controles o la actualización dinámica de los datos dibujados sobre el mapa.
 - Incorporar más tipos de contenido, siempre teniendo en cuenta que estos sean genéricos y no se involucren en aspectos específicos de la lógica correspondiente a los servicios externos.
 - Enviar avisos (en novedades) al haber cambios en los contenidos.
- Incorporar un motor de búsqueda (como Sphinx, ElasticSearch, Lucene, etc.) para buscar sobre novedades, calendario y todo contenido “buscable”.
- Mejorar la integración con redes sociales, permitiendo la incorporación de novedades desde páginas de *Facebook*, *Twitter* y *Google+*.
- Incorporar la posibilidad de insertar multimedia en las novedades.

Referencias

- [1] Mehdi Achour, Friedhelm Betz, Antony Dovgal, and Nuno Lopes. Php manual, 2017.
- [2] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. Web services. In *Web Services*, pages 123–149. Springer, 2004.
- [3] Amazon. *Signing and Authenticating REST Requests*. <http://docs.aws.amazon.com/AmazonS3/latest/dev/RESTAuthentication.html>, 2017. [Online; accedido 30-01-2017].
- [4] Android.com. App Structure. <http://developer.android.com/design/patterns/app-structure.html>. [Online; accedido 13-02-2017].
- [5] Darwin Biler. Laravel Lumen's performance under production use. <http://www.darwinbiler.com/laravel-lumen-performance-production/>. [Online; accedido 22-01-2017].
- [6] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (soap) 1.1, 2000.
- [7] Pier Luigi Capucci. Art, multiplicity and awareness. 2005.
- [8] Cespi. Qué es Meran. http://www.cespi.unlp.edu.ar/articulo/2013/6/14/que_es_meran. [Online; accedido 20-08-2017].
- [9] CeSPI. Sistema de gestión de alumnos. http://www.cespi.unlp.edu.ar/sistemas_academicos_cespi. [Online; accedido 11-02-2017].
- [10] Andre Charland and Brian Leroux. Mobile application development: web vs. native. *Communications of the ACM*, 54(5):49–53, 2011.
- [11] UDDI Consortium et al. Uddi executive white paper, 2001.
- [12] Universidad de Boston. *Understanding Authentication, Authorization, and Encryption*. <https://www.bu.edu/tech/about/security-resources/bestpractice/auth/>, 2016. [Online; accedido 21-05-2017].

- [13] Randall Degges. *The Ultimate Guide to Mobile API Security.* <https://stormpath.com/blog/the-ultimate-guide-to-mobile-api-security>. [Online; accedido 21-01-2017].
- [14] Eva Durall Gazulla, Begoña Gros Salvat, Marcelo Fabián Maina, Larry Johnson, and Samantha Adams. Perspectivas tecnológicas: educación superior en iberoamérica 2012-2017. 2012.
- [15] Roy Fielding and Julian Reschke. Hypertext transfer protocol (http/1.1): Semantics and content. 2014.
- [16] Roy Thomas Fielding. Rest: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California*, 2000.
- [17] Apache Software Foundation. *Architectural overview of Cordova platform.* <https://cordova.apache.org/docs/en/7.x/guide/overview/index.html>. [Online; accedido 20-02-2017].
- [18] Free Software Foundation. *A Quick Guide to GPLv3.* <https://www.gnu.org/licenses/quick-guide-gplv3>. [Online; accedido 17-08-2017].
- [19] Free Software Foundation. ¿Qué es el software libre? <https://www.gnu.org/philosophy/free-sw.html>. [Online; accedido 17-08-2017].
- [20] Google. Apps from the University of Oxford. <http://es.slideshare.net/delgadocristian/estudio-de-google-sobre-smartphones-en-argentina>, 2017. [Online; accedido 17-01-2017].
- [21] Klaus Göttling. *Skeuomorphism is dead, long live skeuomorphism.* <https://www.interaction-design.org/literature/article/skeuomorphism-is-dead-long-live-skeuomorphism>, 2017. [Online; accedido 20-08-2017].
- [22] Hatem Hamad, Motaz Saad, and Ramzi Abed. Performance evaluation of restful web services for mobile devices. *Int. Arab J. e-Technol.*, 1(3):72–78, 2010.
- [23] Dick Hardt. The oauth 2.0 authorization framework. 2012.

- [24] harvard.edu. Harvard Mobile Apps. <http://www.harvard.edu/about-harvard/harvard-mobile-apps>, 2017. [Online; accedido 17-01-2017].
- [25] Don Hopkins. The design and implementation of pie menus. *Dr. Dobb's Journal*, 1991.
- [26] Troy Hunt. Your API versioning is wrong, which is why I decided to do it 3 different wrong ways. <https://www.troyhunt.com/your-api-versioning-is-wrong-which-is/>. [Online; accedido 01-02-2017].
- [27] kantarworldpanel. Smartphones OS sales market share. <http://www.kantarworldpanel.com/global/smartphone-os-market-share/>, 2016. [Online; accedido 22-02-2017].
- [28] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Rfc 2104: Hmac: Keyed-hashing for message authentication. 1997.
- [29] Daniel Leithinger and Michael Haller. Improving menu interaction for cluttered tabletop setups with user-drawn path menus. In *Horizontal Interactive Human-Computer Systems, 2007. TABLETOP'07. Second Annual IEEE International Workshop on*, pages 121–128. IEEE, 2007.
- [30] Roberto Lucchi, Michel Millot, and Christian Elfers. Resource oriented architecture and rest.
- [31] Kazuaki Maeda. Performance evaluation of object serialization libraries in xml, json and binary formats. In *Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on*, pages 177–182. IEEE, 2012.
- [32] Christina Voskoglou Mark Wilcox, Stijn Schuermans. The state of the developer nation q3 2016. *Developer Economics*, Agosto 2016. <https://www.developereconomics.com/reports/developer-economics-state-developer-nation-q3-2016> [Online; accedido 23-11-2016].
- [33] Theresa Neil. *Mobile design pattern gallery: UI patterns for smartphone apps*. .”O'Reilly Media, Inc.”, 2014.
- [34] Katherine Noyes. *10 Reasons Open Source Is Good for Business*. http://www.pcworld.com/article/209891/10_reasons_open_source_is_good_for_business.html. [Online; accedido 17-08-2017].

- [35] Antti Oulasvirta, Tye Rattenbury, Lingyi Ma, and Eeva Raita. Habits make smartphone use more pervasive. *Personal and Ubiquitous Computing*, 16(1):105–114, 2012.
- [36] ox.ac.uk. Apps from the University of Oxford. <http://www.ox.ac.uk/apps>, 2017. [Online; accedido 17-01-2017].
- [37] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big'web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, pages 805–814. ACM, 2008.
- [38] María Ripoll. ¿POR QUÉ NECESITA MI EMPRESA UNA APLICACIÓN MÓVIL? <https://www.emprenderalia.com/por-que-necesita-mi-empresa-una-aplicacion-movil/>, 2016. [Online; accedido 20-01-2017].
- [39] Alex Rodriguez. Restful web services: The basics. *IBM developerWorks*, 2008.
- [40] SIU. Página Web del Sistema de Información Universitaria. <http://www.siu.edu.ar/nosotros/>. [Online; accedido 17-08-2017].
- [41] SIU. SIU-Culupí. <https://repositorio.siu.edu.ar/trac/chulupi>. [Online; accedido 11-02-2017].
- [42] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.
- [43] Google Trends. *Tendencias de google sobre frameworks PHP*. <https://trends.google.com.ar/trends/explore?date=today%205-y&q=%2Fm%2F0jwy148,yii,%2Fm%2F09cjcl,%2Fm%2F09t3sp,%2Fm%2F0cdvjh>, 2017. [Online; accedido 20-08-2017].
- [44] Laura Tucker. Most Helpful Apps for Students. <https://www.topuniversities.com/blog/most-helpful-apps-students>, 2016. [Online; accedido 17-01-2017].
- [45] Jamil Velji. 4 Ways Your Business Can Benefit From Having a Mobile App. <https://buildfire.com/ways-business-benefit-having-mobile-app/>, 2016. [Online; accedido 17-01-2017].

- [46] w3techs.com. *Usage of server-side programming languages for websites*. https://w3techs.com/technologies/overview/programming_language/all, 2017. [Online; accedido 17-08-2017].

Glosario

AJAX

JavaScript asíncrono y XML, del inglés *Asynchronous JavaScript And XML*. 21

Android

Sistema operativo desarrollado por Google para dispositivos como smartphones y tabletas. 31

API

Application Programming Interface. 19–25, 27, 40, 42

App

Aplicación móvil. 40

CeSPI

Centro Superior para el Procesamiento de la Información. 43, 47

Chulupí

Framework desarrollado por el SIU para la implementación de Guaraní Web.. 45

Clustering

Agrupamiento de computadoras conectadas por una red de alta velocidad que figuran como si fuesen una sola computadora. 19

CRUD

Sigla en inglés de *Create Read Update Delete* que significan el listado de operaciones básicas asociadas a una entidad: Crear, leer, actualizar y eliminar.. 21

CSS

Cascading Style Sheets. 40, 42

FOSS

Free and open-source software. 13

Framework

Es un entorno de trabajo: representa un conjunto de herramientas enfocadas a resolver problemáticas comunes evitando reimplementarlas en cada nuevo desarrollo.. 25–30, 41, 42, 45, 56, 58, 61, I

GET

Método HTTP para obtener un recurso. 21

GNU GPL

GNU General Public License. 15, 47

GPS

Sistema de Posicionamiento Global. 39

Guaraní

Sistema desarrollado por el SIU para la gestión académica de alumnos desde su ingreso hasta su egreso.. 44, 45

HHVM

HipHop Virtual Machine es una máquina virtual desarrollada por Facebook, open source, que optimiza la ejecución de código PHP y Hack.. 29

HMAC

Código de autenticación de mensajes en clave/*hash*. 25

HTML

HyperText Markup Language. 27, 40, 42, 58

HTTP

HyperText Transfer Protocol. 17, 19–22, 25–27

IETF

Internet Engineering Task Force. 19

iOS

Sistema operativo desarrollado por Apple Inc. para dispositivos como smartphones y tabletas. 12, 31

JavaScript

JavaScript es un lenguaje interpretado y ligero. Comúnmente utilizado del lado del cliente en aplicaciones web (ejecutado en el browser). Existen también implementaciones *server side*. . 21, 40–42, 58

JSON

JavaScript Object Notation. 19, 21

Laravel

Es un Framework PHP (que corre del lado del servidor) para desarrollar aplicaciones web.. 27, 28

Microframework

Es un Framework con funcionalidades innecesarias removidas para que sea más liviano.. 27

MIME

Multipurpose Internet Mail Extensions. 19

MVC

Model View Controller. 45

Open Source

Software de código abierto. 26, 42

PhoneGap

Framework (propiedad de *Adobe*) para el desarrollo de aplicaciones móviles híbridas, basado en JavaScript y HTML5.. 42

PHP

PHP: *Hypertext Preprocessor.* 26–30, 45, 46, 57–59, 61

PHP-FPM

FastCGI Process Manager de PHP es una implementación de FastCGI optimizada para la ejecución de PHP en sitios de alto tráfico.. 29

REST

REpresentational State Transfer. 19–23, 25, 27, 61

RESTful

Adjetiva a servicios o APIs, indicando que implementa REST. 19, 20, 24, 27

SDK

Software Development Kit. 40, 42

SIU

Sistema de Información Universitaria. 44, 45

smartphone

Nombre en inglés dado a los teléfonos inteligentes. 1

SMTP

Simple Mail Transfer Protocol. 17

SOAP

Simple Object Access Protocol. 17, 18, 20, 62

Symfony

Symfony es un *framework* PHP basado en MVC para el desarrollo de aplicaciones Web.. 45

UDDI

Universal Description, Discovery and Integration. 17

URI

Uniform Resource Identifier. 19, 21, 22

URL

Uniform Resource Locator. 22, 50

W3C

World Wide Web Consortium. 19

WSDL

Web Services Description Language, es un formato del Extensible Markup Language (XML) que se utiliza para describir servicios web (WS). 18

XML

Del inglés *eXtensible Markup Language*, que significa Lenguaje de Marcas Extensible: es un meta-lenguaje que permite definir lenguajes de marcas.. 17–19, 21

Índice de figuras

1.	Porcentaje de desarrolladores que eligen como plataforma primaria el sistema operativo.	3
2.	Captura de pantalla: ejemplo de uso de Facebook para la publicación de noticias.	5
3.	Capturas de pantalla de la aplicación de la Facultad de Informática: <i>Informática UNLP</i>	7
4.	Captura de pantalla de la aplicación de la Facultad de Ciencias Jurídicas y sociales: <i>JursocUNLP</i>	8
5.	Captura de pantalla de la aplicación <i>UNLP: ART Salud</i>	9
6.	Capturas de pantalla del menú principal de aplicaciones realizadas con Kurogo.	10
7.	Captura de pantalla de la aplicación oficial de la universidad de Oxford.	12
8.	<i>Meme</i> que bromea acerca de las discusiones en Internet sobre el versionado las interfaces REST	22
9.	Flujo de información en el proceso de autenticación por <i>Resource Owner Password Credentials Grant</i> de OAuth2	24
10.	Porcentaje de uso de lenguajes del lado del servidor. Fuente: w3techs.com [46]	27
11.	Tendencia en búsquedas sobre los principales Frameworks PHP en los últimos 5 años. Fuente: Google Trends[43]	28
12.	Tiempo de respuesta de las transacciones web con Lumen para aproximadamente 50k solicitudes por minuto. Fuente: darwinbiler.com [5]	29
13.	Ánálisis de escalabilidad de Lumen: tiempo de respuesta vs solicitudes por minuto. Fuente: darwinbiler.com [5]	30
14.	Ejemplo de patrón de navegación <i>springboard</i> a nivel de Sistema Operativo en iOS	32
15.	Ejemplo de patrón de navegación de “Menú lista” en iTerminal	33
16.	Ejemplo de patrón de navegación <i>dashboard</i> en Google Analytics	33

17.	Ejemplo de interfaz con patrón skeumórfico en aplicación de grabadora para iOS	34
18.	<i>Side Drawer</i> con menú en aplicación de Netflix (desplaza pantalla principal)	35
19.	<i>Toggle menu</i> con menú en aplicación de IMDB	36
20.	<i>Pie menu</i> en <i>assistive touch</i> de iPhone	37
21.	Arquitectura de Cordova. Fuente: [17]	43

Listado de bloques de código

1.	Ejemplo de estructura básica de mensaje XML en SOAP. <i>Envelope</i> tiene dos hijos: <i>header</i> y <i>body</i>	18
----	---	----

Índice de cuadros

1.	Conocimientos de lenguajes necesarios por plataforma[10]	40
----	--	----