UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

Facultad de Ingeniería

Carrera de Ingeniería de Software

**ANALISIS COMPARATIVO DE MECANISMOS DE
COMUNICACION CROSS PLATFORM**

TESIS

PARA OPTAR POR EL TITULO DE INGENIERO DE SOFTWARE

AUTORES

Atencio Cordova, Luis Antonio (0000-0001-6587-329X)

Aybar Guerrero, Miguel Bruno (0000-0002-0168-7487)

ASESOR DE TESIS

Barrientos Padilla, Alfredo (0000-0002-0029-4913)

Lima, 10 de abril de 2018

Esta tesis dedicada a nuestras familias, cuyo apoyo se mantuvo incondicional durante nuestra vida universitaria y crecimiento profesional. Gracias por todo

Resumen ejecutivo

El análisis Comparativo de Mecanismos de Comunicación Cross Platform es una investigación que busca encontrar diferentes soluciones tecnológicas que permitan la comunicación entre las plataformas predominantes, Android y IOS. Esta determina su funcionalidad según el entorno para el cual han sido desarrolladas, mostrando información competente a sus capacidades con respecto a los medios de comunicación que utiliza y la velocidad de comunicación medida en milisegundos. Para tal objetivo, utilizamos una aplicación común que mide la comunicación en tiempo real, un chat, que envía información restando el tiempo en milisegundos del dispositivo de envío al dispositivo receptor. Para un mejor análisis, durante el periodo de pruebas, tomaremos diversas mediciones para mostrar un promedio que será el resultado en tiempo de comunicación para cada tecnología. Para estas pruebas, además, se tuvo un entorno fijo, donde se midió con respecto al uso de internet (WIFI, 3G/4G), distancia (corta, media y larga) y comunicación entre plataformas (Android - Android, iOS - Android, iOS - iOS).

Palabras Clave:

Cross-Platform, Cross Device Interaccion, Comunicación en Tiempo Real, Android, iOS, Sockets, Cloud, Bluetooth, Wi-Fi, Performance.

ABSTRACT

The present research aims to compare the different technologies that allow real-time communication between Android and iOS devices. We conducted a measurement-driven experiment to test the performance of each technology under different scenarios. We determine their functionality according to the environment for which they have been developed, showing competent information about their capabilities based on the communication channel they use, and their performance measured in milliseconds. The results allow mobile application developers to determine which technologies suits them the best, based on the specific context of their projects.

Keywords: *Cross-Platform, Cross Device Interaction, Real-time communication, Android, iOS, Sockets, Cloud, Bluetooth, Wi-Fi, Performance.*

Lista de Contenidos

CAPÍTULO 1: DESCRIPCIÓN DEL PROYECTO	10
1.1 OBJETO DE ESTUDIO	10
1.2 DOMINIO DEL PROBLEMA	10
1.3 PLANTEAMIENTO DE LA SOLUCIÓN.....	11
1.4 OBJETIVOS DEL PROYECTO	11
1.4.1 OBJETIVO GENERAL	11
1.4.2 OBJETIVOS ESPECÍFICOS	11
1.5 INDICADORES DE ÉXITO.....	12
1.6 PLANIFICACIÓN DEL PROYECTO.....	13
1.6.1 ALCANCE	13
1.6.2 PLAN DE GESTIÓN DEL TIEMPO.....	13
1.6.3 PLAN DE GESTIÓN DE RECURSOS HUMANOS	13
1.6.4 Plan de comunicaciones.....	16
CAPÍTULO 2: STUDENT OUTCOMES	20
2.1. Competencias Específicas	20
2.1.1. Student Outcome A	20
2.1.2. Student Outcome B.....	21
2.1.3. Student Outcome C.....	22
2.1.4. Student Outcome D	23
2.1.5. Student Outcome E.....	23
2.1.6. Student Outcome F	24
2.1.7. Student Outcome G	25
2.1.8. Student Outcome H	25
2.1.9. Student Outcome I.....	26
2.1.10. Student Outcome J.....	27
2.1.11. Student Outcome K	28
2.2. Competencias Generales.....	29
2.2.1. Comunicación escrita	29
2.2.2. Comunicación oral.....	29
2.2.3. Ciudadanía.....	30
2.2.4. Pensamiento innovador	30
2.2.5. Pensamiento crítico	31
2.2.6. Pensamiento cuantitativo.....	31

2.2.7. Manejo de la información.....	32
CAPÍTULO 3: MARCO TEÓRICO	33
3.1 CROSS DEVICE INTERACTION	33
3.2 CROSS PLATFORM DEVELOPMENT.....	33
3.3 COMUNICACIÓN EN TIEMPO REAL (RTC)	34
3.4 ANDROID.....	35
3.5 iOS.....	37
3.6 LIBRERÍAS	38
CAPÍTULO 4: ESTADO DEL ARTE	39
4.1 REVISION DE LA LITERATURA.....	39
4.2 MANIPULACION DE DIVERSOS DISPOSITIVOS	39
4.3 COMPLEJIDAD DEL DESARROLLO CROSS-DEVICE	43
4.4 COMPARACION DE TECNICAS DE MOVIMIENTO CROSS-DEVICE	46
CAPÍTULO 5: DESARROLLO DEL PROYECTO	51
5.1 TECNOLOGIAS DE COMUNICACIÓN	51
5.1.1 DESARROLLO DE PROYECTO BASE	51
5.1.3 IMPLEMENTACION DE TECNOLOGIAS INVESTIGADAS	56
5.2 DESARROLLO DE MEDICIONES.....	90
5.2.1 ENTORNO DE DESARROLLO DE PRUEBAS	90
5.2.2 SINCRONIZACION DE TIEMPO REAL Y TIEMPOS DE ENVIO...	93
6. CAPÍTULO 6: RESULTADOS DEL PROYECTO	95
6.1. Resultados de las mediciones	95
6.2. Validación de los resultados	96
6.2.1. Desarrollo de la encuesta.....	97
6.2.2. Encuestados	97
6.2.3. Resultados.....	98
7. CAPÍTULO 7: GESTIÓN DEL PROYECTO	100
7.1. Gestión del Tiempo	100
7.2. Gestión de los Recursos Humanos	101
7.3 Gestión de las Comunicaciones	103
CONCLUSIONES.....	105
BIBLIOGRAFIA.....	106

LISTA DE TABLAS

Tabla 1 Tabla de Recursos.....	15
Tabla 2 Tabla de Riesgos.....	19
Tabla 3 Características de Ably.....	56
Tabla 4 Características de Firebase	61
Tabla 5 Características de Pubnub.....	66
Tabla 6 Características de RabbitMQ.....	72
Tabla 7 Características de Lighstreamer	79
Tabla 8 Características de Xamarin.....	85
Tabla 9 Dispositivos utilizados durante las pruebas.....	90
Tabla 10 Aplicaciones requeridas por tecnología.....	91
Tabla 11 Escenarios de prueba	92
Tabla 12 Descripción de Roles.....	101

LISTA DE FIGURAS

Figura 1 Organigrama del Proyecto.....	14
Figura 2 Arquitectura de Sistemas Android	35
Figura 3 Capas de iOS	37
Figura 4 DemoScript StoryBoard	44
Figura 5 Promedio de completitud de tiempo para cada sub-tarea de DemoScript	45
Figura 6 La Técnica Pinch.....	46
Figura 7 La técnica swipe.....	46
Figura 8 La Técnica throw.....	47
Figura 9 La Técnica Tilt	47
Figura 10 Efectividad: Promedio exitoso de intentos por usuario por técnica, para blancos grandes y pequeños.....	48
Figura 11 Eficiencia: Promedio de tiempo toma por blanco por cada técnica para blancos pequeños y grandes.....	49
Figura 12 Margen de Error: Distancia promedio en pixeles por técnica por blancos pequeños y grandes.....	49
Figura 13 Diagrama de Componentes - Ably.....	57
Figura 14 Diagrama de Despliegue - Ably.....	57
Figura 15 Diagrama de Secuencia - Ably.....	57
Figura 16 Diagrama de Componentes - Firebase	62
Figura 17 Diagrama de Despliegue - Firebase	62
Figura 18 Diagrama de Secuencia - Firebase	62
Figura 19 Diagrama de Componentes - Pubnub.....	67
Figura 20 Diagrama de Despliegue - Pubnub.....	67
Figura 21 Diagrama de Secuencia - Pubnub	67
Figura 22 Diagrama de Componentes - RabbitMQ.....	73
Figura 23 Diagrama de Secuencia - RabbitMQ.....	73
Figura 24 Diagrama de Componentes - Lighstreamer.....	80
Figura 25 Diagrama de Despliegue - Lighstreamer.....	80
Figura 26 Diagrama de Secuencia - Lighstreamer	80
Figura 27 Diagrama de Componentes - Xamarin.....	86
Figura 28 Diagrama de Despliegue - Xamarin.....	86
Figura 29 Diagrama de Secuencia - Xamarin.....	86

Figura 30 Tiempo de Envío en Milisegundos de Tecnologías basadas en comunicación por la nube	95
Figura 31 Tiempo de Envío en Milisegundos de Tecnologías basadas en comunicación en una red local.....	96
Figura 32 Tiempo de Envío en Milisegundos de Tecnologías basadas en Bluetooth	96
Figura 33 Encuestados: Rol en su compañía	97
Figura 34 Encuestados: Años de experiencia desarrollando aplicaciones	98
Figura 35 Tecnologías que los desarrolladores conocían antes de leer nuestra investigación.....	98
Figura 36 Encuesta: Percepción de la utilidad de la investigación	99
Figura 37 Encuesta: Tiempo destinado a la elección de una tecnología, luego de ver la investigación.....	99
Figura 38 Encuesta: Recomendación de la investigación	99

INTRODUCCIÓN

Hoy en día, estamos en una era en la que la tecnología juega un papel fundamental: los dispositivos móviles han desplazado a los computadores de escritorio como los dispositivos más utilizados, y el “Amanecer del Internet de las Cosas” es prácticamente una realidad. Cada vez más y más objetos que utilizamos en el día a día se ven interconectados y podemos interactuar con ellos de una forma completamente nueva. Si bien existen soluciones que permiten la comunicación de forma casi instantánea, aún persiste un problema que limita mucho las posibilidades de desarrollo: cada sistema operativo tiene sus propias restricciones de seguridad y de comunicación. Es incluso difícil o poco probable realizar ciertos casos de uso de interconectividad, que, si bien antes eran poco comunes, ahora son indispensables para garantizar la calidad de las soluciones. Esto tiene tres grandes consecuencias. Primero, no se están enfocando los esfuerzos en la dirección correcta, pues siguiendo ese rumbo no se obtendrá la interconectividad que los usuarios desean, eficiente y eficaz. En segundo lugar, esta fragmentación no contribuye con el avance de la interconectividad que se busca alcanzar, sino que va imponiendo más y más barreras. Existe una tercera consecuencia, enfocada a las restricciones impuestas por cada plataforma al momento del desarrollo, y tratan de implementar sus soluciones con las alternativas de comunicación que tienen a la mano.

El objetivo de nuestra investigación es encontrar diferentes soluciones tecnológicas que permitan la comunicación entre las plataformas predominantes, Android y IOS, y determinar su funcionalidad según el entorno para el cual han sido desarrolladas, mostrando información competente de sus capacidades con respecto a los medios de comunicación que utiliza y la velocidad de comunicación medida en milisegundos. Para tal objetivo, utilizamos una aplicación común que mide la comunicación en tiempo real, un chat, que envía información restando el tiempo en milisegundos del celular de envío al celular receptor. Para un mejor análisis, durante el periodo de pruebas, tomaremos diversas mediciones para mostrar un promedio que será el resultado en tiempo de comunicación para cada tecnología. Para estas pruebas, además, se tendrá un entorno fijo, se medirá con respecto a uso de internet (wifi, 3G/4G), distancia (corta, media y larga) y comunicación entre plataformas (Android - Android, iOS - Android, iOS - iOS).

CAPÍTULO 1: DESCRIPCIÓN DEL PROYECTO

El capítulo actual contiene 6 secciones, las cuales explicarán de manera general el proyecto. Se presenta la necesidad (problema) encontrada, el objetivo general y los objetivos específicos del proyecto, a los indicadores de éxito y a la planificación del proyecto en base a la guía del PMBOK.

1.1 OBJETO DE ESTUDIO

El presente proyecto tiene como objeto de estudio la comunicación orientada al desarrollo de aplicaciones móviles para sistemas operativos Android y IOS. En este sentido, este está dirigido a comparar metódicamente las diferentes tecnologías de comunicación entre estas plataformas. Los medios centrales que utilizaremos serán el uso de internet, comunicación P2P y comunicación mediante bluetooth.

1.2 DOMINIO DEL PROBLEMA

El desarrollo y uso de aplicaciones de dispositivos móviles ha crecido en a un ritmo alto en los últimos años. Durante la última década la compra y el uso de dispositivos móviles ha crecido de una manera significativa, según Gartner el índice de compras del segundo trimestre del 2016 llega a las 344 mil unidades de equipos, 44 mil adicionales comparados al trimestre pasado¹. Además, según el IDC, los dos sistemas operativos líderes son Android y IOS, con un 86.8% y 12.5% de participación respectivamente para el tercer semestre del 2016². Debido a esto, el desarrollo de aplicaciones para estas plataformas ha aumentado considerablemente.

Debido a la preferencia entre ambos sistemas operativos, la comunicación entre ambos es muy importante, valiéndose de diferentes tecnologías se ha logrado realizar este propósito. Sin embargo, cada tecnología responde de manera diferente según capacidad de envío, velocidad de transmisión, distancia, entorno y conectividad.

¹ (Gartner, 2016)

² (IDC, 2016)

Por ello, el desarrollo del análisis comparativo tiene como finalidad apoyar a la elección de tecnologías para desarrollo de la interacción entre aplicaciones de manera indistinta a la plataforma a la cual están implementadas, permitiendo a un equipo de desarrollo poder sustentar con objetividad el uso de cierta tecnología para la comunicación entre dispositivos entre estas dos plataformas.

1.3 PLANTEAMIENTO DE LA SOLUCIÓN

Como solución, se ha planteado la investigación de las diferentes tecnologías de comunicación entre dispositivos IOS y Android traducidas en un cuadro comparativo, analizando sus principales características con respecto a la comunicación.

1.4 OBJETIVOS DEL PROYECTO

1.4.1 OBJETIVO GENERAL

Realizar una investigación comparativa sobre diferentes tecnologías de comunicación Cross-Platform entre dispositivos móviles iOS y Android, analizando su arquitectura, protocolos de comunicación y midiendo el tiempo de respuesta entre interacciones.

1.4.2 OBJETIVOS ESPECÍFICOS

Para el proyecto se plantearon los siguientes objetivos específicos relacionados al objetivo general:

- **OE1:** Analizar las distintas implementaciones existentes de Cross Device Interactions para Android y iOS
- **OE2:** Diseñar la arquitectura física y lógica de cada tecnología, definiendo un mecanismo de interacción entre las diferentes plataformas.
- **OE3:** Desarrollar pruebas de concepto con las tecnologías investigadas por medio de aplicaciones orientadas a la comunicación entre dispositivos Android y iOS.
- **OE4:** Implementar un análisis comparativo entre las tecnologías Cross-Platform investigadas, según el enfoque que usan, su arquitectura y forma de comunicación.

1.5 INDICADORES DE ÉXITO

Para el proyecto se plantearon los siguientes indicadores de éxito relacionados a cada objetivo específico:

Para el Objetivo Específico 1:

- **IE01:** Acta de aprobación del cliente, respecto al análisis de las distintas tecnologías Cross Device Interactions para Android y iOS.
- **IE02:** Acta de aprobación del cliente, respecto al análisis de investigaciones de desarrollo en proyectos similares.

Para el Objetivo Específico 2:

- **IE03:** Acta de aprobación del cliente, donde se acepta la arquitectura física y lógica que define el mecanismo de cada tecnología.
- **IE04:** Acta de aprobación del cliente, respecto al diseño de la arquitectura para las pruebas de concepto.

Para el Objetivo Específico 3:

- **IE05:** Acta de aprobación del cliente, respecto al desarrollo de demos las pruebas de concepto de cada tecnología.
- **IE06:** Acta de aprobación del cliente, respecto a la validación del análisis de tiempo de descubrimiento, tiempo de conexión y tipo de información que se puede enviar para cada demo

Para el Objetivo Específico 4:

- **IE07:** Acta de aprobación del cliente, respecto a la implementación del análisis comparativo que define la investigación de los diferentes mecanismos de comunicación sobre Cross Device Interactions.

1.6 PLANIFICACIÓN DEL PROYECTO

1.6.1 ALCANCE

El proyecto se va a desarrollar en 2 ciclos académicos los cuales serán el 2017-01 y 2017-02. El alcance del proyecto consta de la documentación necesaria para el desarrollo del mismo, ésta será entregada de manera progresiva durante el transcurso del proyecto. Ésta compuesta principalmente por los diagramas arquitectónicos de las tecnologías (diagrama de despliegue, diagrama de componentes y diagrama de secuencia), prototipos de la aplicación, actas de conformidad del cliente y la memoria del proyecto.

1.6.2 PLAN DE GESTIÓN DEL TIEMPO

El plan de gestión de tiempo define el cronograma del proyecto, indicando las fases y entregables con sus respectivas fechas, las cuales se deberán de cumplir para la correcta realización del proyecto. Adicionalmente, también se utilizará el Plan de Trabajo y de manera semanal se realizarán actas con el control de valor ganado que corresponderán a las reuniones con el cliente.

La periodicidad de actualización del avance y control del plan de trabajo es semanal y está a cargo del jefe de proyecto.

1.6.3 PLAN DE GESTIÓN DE RECURSOS HUMANOS

El plan de gestión de recursos humanos define los roles con sus respectivas responsabilidades. Asimismo, quien cubre cada rol descrito y las personas involucradas con la realización y aprobación del proyecto.

- Organigrama del Proyecto

Teniendo como referencia que el proyecto pertenece a la Empresa Virtual Innova TI, el equipo del proyecto se distribuye de acuerdo al siguiente organigrama:

Figura 1 Organigrama del Proyecto³

- **Comité de Proyectos:** Se encuentra conformado por la directora de la Escuela de Ingeniería de Sistemas y Computación, Rosario Villalta; y los coordinadores de las carreras de la Escuela: Jimmy Armas (coordinador de la carrera de Ingeniería de Sistemas de Información), Pedro Shiguihara (coordinador de la carrera de Ciencias de la Computación) y Luis Vives (coordinador de la carrera de Ciencias de la Computación).
- **Profesor de Innova TI:** El profesor de gerente de la empresa virtual Innova TI es Julio Quispe, quien se encarga del seguimiento y aprobación de los proyectos que son de dicha empresa.

³ Fuente: Elaboración propia

- **Cliente:** El cliente del proyecto es Alfredo Barrientos, quien se encarga de dar los requerimientos del proyecto, de dar retroalimentación conforme se vaya desarrollando el proyecto y de aprobar el producto final.
- **Jefes de Proyecto:** Es el alumno que se encargan de la gestión del proyecto a desarrollar. En esta ocasión, el jefe de proyecto del actual proyecto es el estudiante de Ingeniería de Software, Luis Antonio Atencio Córdova
- **Jefe de Desarrollo:** Es el alumno que se encargan del desarrollo del proyecto a desarrollar. En esta ocasión, el jefe de desarrollo del actual proyecto es el estudiante de Ingeniería de Software, Miguel Bruno Aybar Guerrero
- **Desarrolladores:** El equipo de desarrolladores son los encargados de implementar el código necesario para que el proyecto cumpla con los requerimientos especificados. En el actual proyecto, Luis Antonio Atencio Cordova y Miguel Bruno Aybar Guerrero y los recursos asignado por la empresa virtual Software Factory.

- Adquisición del personal

El proyecto requerirá dos colaboradores para las actividades de desarrollo por lo que se solicitará dichos recursos a Software Factory. Se pedirá un recurso cada ciclo académico que dure el desarrollo del proyecto, 2017-01 y 2017-02. Se asume que dichos recursos cumplen con todas las habilidades requeridas para desarrollar las tareas con efectividad.

Tabla 1 Tabla de Recursos

Rol desempeñado	Tipo (Interno / Externo)	Ciclo académico requerido	Fase del proyecto requerido	Cantidad personal requerido
Jefe de desarrollo	Interno	2017-01 2017-02	Todas	2
Desarrollador Móviles 1	Interno	2017-01	Ejecución - Control y Monitoreo	1
Desarrollador Móviles 2	Interno	2017-02	Ejecución - Control y Monitoreo	1

Los horarios obligatorios para los colaboradores del proyecto son los siguientes:

- Martes de 16:00 a 19:00
- Viernes de 16:00 a 19:00
- Sábado de 16:00 a 22:00

1.6.4 Plan de comunicaciones

El plan de comunicaciones tiene como finalidad brindar los procedimientos para la comunicación entre los interesados. A continuación, se indicará las guías para los eventos de comunicación y para la documentación del proyecto:

Guías para las reuniones

Todas las reuniones deben seguir las siguientes pautas:

- En caso de tener un nuevo miembro en cada reunión, se hará la respectiva presentación de este ante los asistentes.
- Cada punto de la agenda contará con un nivel de prioridad a ser tratado.
- Se cuenta con un horario específico para cada reunión, sin embargo, será necesario confirmar/recordar la reunión vía correo electrónico un día antes como máximo.
- En caso se deba postergar una reunión, se deberá justificar y además acordar el nuevo día de esta.
- Al finalizar cada reunión se deberá recopilar lo tratado, resaltando los acuerdos para posteriormente elaborar el acta de reunión.
- Todas las actas de reunión deberán ser firmadas por el jefe de proyecto, asistente de proyecto y gerente general o cliente, respectivamente.

Guías para correo electrónico

Todas las remisiones de correos electrónicos deben seguir las siguientes pautas:

- Cada correo electrónico debe ser redactado de una manera formal.
- Todo correo electrónico al cliente debe ser copiado a todo el equipo de proyecto; en especial a gerente, jefe de proyecto y de desarrollo.
- Los correos electrónicos entre el gerente de la empresa y el cliente, deberá ser enviado por el jefe de proyecto para establecer un estándar en la conexión.
- En caso de envíos de solicitud de servicios a empresas externas, se debe copiar al correo de la empresa.

Guías para Codificación de Documento

La codificación de los documentos del proyecto será la siguiente:

[Código de Proyecto]- [Nombre de documento]-[Versión]

Ejemplo:

ACMCP– Plan de Gestión de Comunicaciones – 1.0

Guías para Almacenamiento de documento

El almacenamiento de los documentos del proyecto deberá seguir las siguientes pautas:

- Se solicitará al asistente el acceso al repositorio de la empresa.
- Se subirán los archivos según la estructura definida por la empresa.
- Cada nueva versión de un documento debe ser subida en la misma carpeta del documento origen.

Guías para Recuperación y Reparto de Documento

Para la recuperación de documentos se tomarán las siguientes pautas:

- Se solicitará el apoyo del asistente de gerencia, quien cuenta con acceso a los backups de la empresa.

- Se subirán nuevamente los archivos perdidos al repositorio de la empresa.

Para el reparto de documentos se tomarán las siguientes pautas:

- Se solicitará el otorgamiento de permisos de acceso al repositorio al asistente de gerencia.
- Se comprobará el acceso de los involucrados.

Guía para el control de versiones

El control de las versiones de los entregables será de la siguiente forma:

- Todos los entregables previos a una entrega pactada (entrega parcial, entrega final) serán codificados de la forma: 0. #, siendo “#” el número de veces que ha sido realizado.
- Cuando ocurra una entrega pactada, el entregable será codificado de la siguiente manera: 1.0, 2.0, 3.0 y así sucesivamente hasta que el proyecto finalice.

1.6.5 Plan de Gestión de Riesgos

El plan de gestión de riesgos tiene como finalidad prevenir hechos que puedan afectar al proyecto, ya sea en tiempo, alcance o recursos. Los principales riesgos se detallan en la siguiente tabla.

Tabla 2 Tabla de Riesgos

#	Riesgo	Probabilidad	Impacto	Estrategia de mitigación
1	La adquisición de dispositivos iOS y Android.	Medio	Medio	Se plantea usar dispositivos proporcionados por la universidad.
2	Cambio en el alcance del proyecto o producto por parte cliente o comité.	Alto	Alto	Se debe realizar el cambio al proyecto, teniendo en cuenta el impacto en costo y tiempo que ya fue invertido.
3	Falta de recursos en Software Factory.	Medio	Bajo	Tener la documentación en orden para realizar una solicitud de servicio iniciando el periodo académico.
4	Necesidad de capacitación para el recurso.	Medio	Alto	Se plantea el uso tiempos de aprendizaje mediante reuniones online para manejar este riesgo.
5	La estimación de tiempos no fue la correcta.	Bajo	Alto	Se debe ajustar el cronograma de entregables con tiempos realistas.
6	Cambio de versión significativos en alguna de las tecnologías utilizadas.	Medio	Medio	Se debe definir si los cambios en las tecnologías son indispensables para su aplicación.
7	La salida de algún miembro del equipo de trabajo.	Medio	Alto	Se debe aumentar las horas de trabajo por persona, evitando la pérdida de tiempo.
8	Las funcionalidades de las aplicaciones móviles no reflejan lo esperado debido a su desarrollo Cross-platform.	Medio	Alto	Se debe adaptar las funcionalidades para simular una aplicación nativa, mediante la revisión de incidencias en las funcionalidades de la biblioteca ya implementadas.

CAPÍTULO 2: STUDENT OUTCOMES

2.1. Competencias Específicas

2.1.1. Student Outcome A

“Propone soluciones a problemas en Ingeniería de Software aplicando principios de matemáticas, ciencias de la computación e ingeniería”.

Se realizó un cálculo matemático para poder realizar la sincronización en tiempo real de los dispositivos móviles para realizar las mediciones con un margen de error mínimo. Se trabajó con el modelo matemático propio del algoritmo de sincronización de tiempos utilizando NTP, y además se plantaron fórmulas propias para poder realizar el ajuste de tiempos en todos los dispositivos utilizados para medir tiempos. Con nuestra propuesta, es posible hallar el tiempo de envío real entre dispositivos con relojes internos que no necesariamente están sincronizados.

Se diseñaron diagramas de arquitectura de software de alto nivel en cada una de las 5 tecnologías Cross-device implementadas, utilizando la notación estándar UML. Los diagramas de componentes, despliegue y secuencia realizados representan la estructura y el comportamiento de los diversos componentes que conforman la arquitectura planteada a nivel lógico y físico.

Se utilizaron diversos lenguajes de programación durante el desarrollo del proyecto para implementar las distintas pruebas de concepto realizadas, así como la aplicación y plantillas de proyectos Cross-device. En total se utilizaron 5 lenguajes distintos, siendo estos Java, Kotlin, Swift, Javascript y C#. Esta diversidad en uso de lenguajes es resultante del trabajo en las plataformas de desarrollo Android, iOS, .NET y Unity. Además, durante el trabajo respectivo con cada lenguaje se aplicaron los estándares y fundamentos de programación necesarios para garantizar una correcta construcción del código fuente. Se implementó el paradigma de programación orientada a objetos, programación funcional, y reactiva. Se realizó una capa de abstracción para la transmisión

de la información, para que acoplar cualquier otra tecnología de comunicación en realtime sea sencillo.

Se recolectaron los requerimientos necesarios para el desarrollo de la aplicación de conectividad Cross-platform. El alcance de la aplicación se definió en las distintas reuniones con el cliente, luego de recopilar las ideas y definirlas en base a historias de usuario para su implementación. Estos requerimientos quedaron concretados por medio de un acta de reunión con el cliente. Luego, se procedió a realizar los diseños y el desarrollo de las pruebas de concepto. Finalmente, el cliente realizó las pruebas de la aplicación para validar el correcto funcionamiento de la misma, aceptando su conformidad por medio de un acta.

2.1.2. Student Outcome B

“Diseña y conduce experimentos en base al análisis e interpretación de datos relevantes en la implementación de soluciones de software.”

Se diseñaron una serie de escenarios de medición para conocer el rendimiento de las diferentes tecnologías de comunicación bajo diferentes entornos. Para el diseño de estas pruebas, fue imperativo realizar las mediciones bajo un ambiente controlado y uniforme, pues de lo contrario las mediciones podrían distorsionarse y no reflejar el rendimiento real de cada tecnología, con lo cual el experimento perdería validez.

Además, se realizó un cuestionario en Google Forms para validar la efectividad, utilidad y facilidad de comprensión de la investigación realizada, así como la reducción de tiempo para la selección de una tecnología a implementar, según los casos de usos requeridos. Este cuestionario se aplicó a 10 expertos en el desarrollo móvil con más de 3 años de experiencia en la creación de aplicaciones iOS y Android.

Se implementaron 22 pruebas de conceptos para validar las limitaciones y efectividad de las 11 tecnologías investigadas, según diversos casos de uso planteados, y se desarrollaron 5 servicios que funcionarían como backend de algunas de las pruebas de concepto. Estas pruebas de concepto se enfocaron en 3 escenarios: por medio de por internet, por una red local, y por Bluetooth. Las pruebas de concepto fueron desarrolladas para los sistemas

operativos Android y iOS. Estas pruebas tuvieron como requerimientos relevantes la conexión entre dispositivos y envío y recepción de mensajes.

Las más de 6000 mediciones realizadas nos permiten hallar las tecnologías que tienen mejor rendimiento bajo diferentes escenarios. Además, nos muestran los escenarios en los que todas las tecnologías rinden mayor o menor eficientemente.

La encuesta aplicada en Google Forms permitió la recolección de los datos necesarios para la evaluación del material desarrollado como parte de la investigación.

2.1.3. Student Outcome C

“Diseña sistemas, componentes o procesos para encontrar soluciones a necesidades, teniendo en cuenta restricciones económicas, sociales, políticas, éticas, de salud y seguridad y otras propias del entorno empresarial.”

Se diseñó la arquitectura física y lógica de la aplicación de conectividad Cross-platform. En el caso de la arquitectura lógica se realizó para tener un mejor detalle de los diferentes niveles de componentes que maneja el aplicativo según el sistema operativo. Por otro lado, la arquitectura física describe la interacción entre el hardware, el mecanismo de conexión y los servicios de la aplicación. Además, se crearon plantillas de proyectos para la creación de aplicaciones Cross-device por cada una de las tecnologías trabajadas, estas plantillas implementan las arquitecturas de alto nivel trabajadas en los diagramas de componentes, despliegue y secuencia.

Se diseñó e implementó un proyecto tanto para Android como para iOS, que permite la comunicación entre dispositivos móviles y realizar mediciones con respecto al tiempo de envío de mensajes de cada tecnología bajo diferentes escenarios. Fue desarrollado considerando que integrar una tecnología adicional sea rápido y sencillo, de tal forma que todas las aplicaciones generadas funcionen bajo un entorno común, el mantenimiento de los cambios este unificado, y la generación de cada ejecutable sea sencilla.

Se realizó un análisis del rendimiento de cada tecnología bajo diferentes escenarios. Estas mediciones, junto con los diagramas y código fuente desarrollado, beneficiará a desarrolladores de aplicaciones móviles a agilizar este proceso de selección de tecnologías Cross-device.

2.1.4. Student Outcome D

“Participa en equipos multidisciplinarios liderando o desarrollando sus tareas eficientemente con profesionales de diferentes especialidades o dominios de aplicación.”

Se lidero un equipo de proyecto multidisciplinario en varias fases del proyecto. Durante el planeamiento inicial se trabajó con dos integrantes de la carrera de Ingeniería de Sistemas de Información para realizar una validación y verificación de los entregables correspondientes al PMBOK. Se trabajó con un recurso de Ingeniería de Software para realizar el landing page del proyecto, el cual evidencia todo lo realizado durante nuestra investigación. Por último, el diseño del poster del proyecto se trabajó con el apoyo de un diseñador gráfico.

Durante el desarrollo de la página web del proyecto, se trabajó con un recurso de Ingeniería de Software con la capacidad de plasmar el diseño y el contenido del proyecto utilizando tecnologías de Frontend como HTML5, CSS3 y Javascript. Cada recurso aportó al desarrollo del proyecto en diferentes dominios de aplicaciones, siendo estos el dominio en aplicaciones móviles y el dominio en desarrollo web.

2.1.5. Student Outcome E

“Identifica, formula y resuelve problemas de Ingeniería de Software.”

Se analizaron artículos científicos en diversos repositorios de investigación como ACM e IEEE relacionados a la conexión de dispositivos, los cuales posteriormente formaron parte del estado del arte del proyecto. En dichos artículos se presentaron proyectos que utilizaron explícitamente los conceptos descritos por Cross-device interaction. Esto es generado por la necesidad de conexión entre múltiples dispositivos para realizar una interacción. Se pudo identificar la dificultad en la creación de estas aplicaciones por todo el proceso de implementación que los autores tuvieron que llevar a cabo, teniendo como causa principal la falta de estandarización entre diversas plataformas y sistemas operativos.

Luego de identificado el problema, se procedió a realizar un correcto planteamiento del mismo, teniendo como resultado la dificultad en el desarrollo de aplicaciones con

conectividad Cross-device. En base a este problema, se realizó el Project Charter del proyecto junto a la formulación del objetivo general, así como los objetivos específicos para conseguir una solución a la problemática planteada. Además, el contenido del documento cubre el alcance del proyecto, sus restricciones y los riesgos planteados en la etapa de planificación.

El problema formulado fue atacado por medio de los objetivos específicos definidos previamente, estos fueron ejecutados mediante el seguimiento de un plan de trabajo con las fechas y entregables a cumplir. Los cuales fueron realizados con proactividad al definir posibles planes de contingencia en los riesgos identificados para conseguir una mitigación de los mismos. Al concluir todas las actividades del plan de trabajo, se entregó el análisis de los resultados encontrados, el cual es detallado tanto en el presente documento como en el paper elaborado.

2.1.6. Student Outcome F

“Propone soluciones a problemas de Ingeniería de Software con responsabilidad profesional y ética.”

Durante la elaboración de los diversos entregables del proyecto como la documentación de las pruebas de concepto, investigaciones, project charter, estado del arte, paper, la memoria final, entre otros, se realizó la citación de todas las fuentes bibliográficas y páginas web utilizadas como referencias o puntos de aprendizaje. Esto se puede apreciar en la sección de bibliografía de cada documento, en donde se detalla el origen de dichas fuentes.

Para el desarrollo de las pruebas de concepto, la aplicación Cross-device y otros entregables del proyecto, se cumplieron los estándares profesionales esperados para el perfil de Ingeniero de Software. Para este cumplimiento se tomó como punto de apoyo el código de ética del Ingeniero de Software impuesto por ACM. A modo de ejemplo, esto se valida mediante la correcta implementación del código fuente de las aplicaciones siguiendo estándares internacionales de programación, el cumplimiento y la conformidad de entregables con el cliente, el aprendizaje de nuevas herramientas y tecnologías para la correcta implementación del proyecto, el trabajo proactivo con equipos multidisciplinarios, entre otros.

2.1.7. Student Outcome G

“Comunica ideas o resultados de manera oral o escrita con claridad y efectividad a públicos de diferentes especialidades y niveles jerárquicos.”

A través de dos presentaciones periódicas al comité de proyectos, se comunicó el problema de Ingeniería de Software a resolver al detalle requerido para transmitir las ideas de forma clara y concisa. Se empleó medios audiovisuales (videos, imágenes y diagramas) para enriquecer las presentaciones. Las observaciones, críticas y mejoras descritas por el comité fueron resueltas e integradas después de su evaluación objetiva durante las presentaciones realizadas y en diferentes reuniones presenciales.

Luego de realizar una reunión con algún miembro del comité, el cliente del proyecto, o un stakeholder se procedió a crear un acta de reunión, en donde se explica a detalle los temas discutidos, acuerdos concretados y actividades a realizar. Además, se elaboraron actas de conformidad de indicadores cumplidos en los dos periodos académicos que abarcó el desarrollo del proyecto, como también el acta de cierre final del mismo. En adicional, se enviaron correos para agendar reuniones y comunicar ideas o actividades a trabajar con los stakeholders, y solicitar los servicios de las empresas de IT service y Software factory.

2.1.8. Student Outcome H

“Identifica el impacto de las soluciones de Ingeniería de Software en el contexto global, económico y del entorno de la sociedad.”

Durante la etapa de investigación del proyecto se analizaron las distintas tecnologías de comunicación Cross-device para los sistemas operativos iOS y Android, así como los proyectos relacionados a este tipo de comunicación realizados por investigadores de distintas nacionalidades. Esto se ve reflejado en el [Cap. 4] correspondiente al estado del arte del proyecto. En los artículos revisados, los autores explican su aporte y el impacto que este puede efectuar sobre diversas variables en el desarrollo Cross-device como tiempo, recursos y costos.

Tomar decisiones con respecto a qué tecnología utilizar en proyectos similares a estos es crítico. En ocasiones, se puede optar por utilizar soluciones pagadas, cuando en realidad el escenario en el que será utilizado no lo amerita. O por el contrario, se puede optar por una solución gratuita, cuando en realidad los recursos que se tienen disponibles a la mano no podrán ser escalados, por lo cual se incurrirá en costos adicionales por migrar toda la solución a una plataforma adicional que lo soporte. Con el aporte que nosotros realizamos, se podrá tomar una decisión mucho más informada, lo cual permitirá utilizar la tecnología más adecuada a nivel económico.

El material realizado fue validado mediante un cuestionario aplicado a expertos en el desarrollo móvil. El propósito de este cuestionario fue medir el impacto que el aporte del presente proyecto pudiera conseguir a este grupo de programadores. Por ello, se seleccionaron solo personas con una experiencia mayor a 3 años en el desarrollo de aplicativos móviles y con un puesto de trabajo elevado. Los resultados de la encuesta fueron positivos en la mayoría de las respuestas, y demuestran que el flujograma desarrollado presenta un impacto en la etapa de investigación inicial en proyectos Cross-device.

2.1.9. Student Outcome I

“Actualiza los conocimientos en metodologías, técnicas, herramientas, por ser necesarios para mantener la vigencia en el desarrollo de software.”

Durante el desarrollo del proyecto aprendimos el uso de 11 frameworks para la creación de aplicaciones Cross-device. Esto permitió el trabajo en diversos lenguajes de programación y plataformas móviles. Por un lado, reforzamos nuestros conocimientos en el desarrollo en iOS, ya que el lenguaje de programación tuvo una cuarta actualización al iniciar el proyecto, por lo que tuvimos que adecuarnos a este cambio. Asimismo, durante el desarrollo del proyecto Google le dio soporte a un nuevo lenguaje de programación para aplicaciones Android: Kotlin. El desarrollo íntegro del proyecto Android fue realizado en este lenguaje, el cual se prevé ser el lenguaje por defecto de los proyectos Android en el futuro. Por otro lado, para trabajar el enfoque de desarrollo híbrido aprendimos variadas tecnologías como Ionic 2, Apache Cordova y Angular 2, siendo estas las tecnologías más relevantes para este enfoque. Además, para la creación del paper y la

memoria del proyecto aprendimos a utilizar el sistema de composición de textos LaTeX, para aplicar los estándares internacionales impuestos por ACM, IEEE, entre otras.

Al realizar el presente proyecto pudimos observar y aprender el cambio que las tecnologías móviles presentan actualmente, ya que, durante el año de desarrollo efectuado, siempre se presentaba alguna actualización o algún cambio en el core de los frameworks trabajados. Debido a esto, siempre tuvimos la iniciativa de continuar aprendiendo, actualizando y mejorando nuestros conocimientos en el desarrollo móvil, para tener un mejor desempeño en el proyecto. A pesar del cambio continuo de esta área de trabajo, seguiremos aprendiendo permanentemente nuevas tecnologías móviles para especializarnos en esta línea de carrera profesional.

2.1.10. Student Outcome J

“Analiza hechos del mundo contemporáneo, identificando el impacto en el desempeño profesional del Ingeniero de Software.”

Durante el año de trabajo en el proyecto se plantearon roles específicos que un Ingeniero de Software debe asumir. El equipo se dividió en Jefe de Proyecto y Jefe de Desarrollo, teniendo asignadas actividades del plan de trabajo correspondientes al rol respectivo. Además, se evidencio la diferencia en dominios de aplicaciones entre las carreras de Ingeniería de Sistemas de Información, Ciencias de la Computación y la propio, al realizar actividades con recursos de estas carreras. Finalmente, se comprobó el cumplimiento de los Student Outcomes correspondientes al perfil de Ingeniero de Software en conjunto al código de ética impuesto por ACM.

Para el trabajo colaborativo en el proyecto utilizamos servicios online que permitieran gestionar la creación de documentos, la comunicación con el cliente y la administración del código fuente de las aplicaciones. En primer lugar, para la creación de documentos de forma colaborativa utilizamos los repositorios de Google Drive y SharePoint. En segundo lugar, para el trabajo en simultaneo en el paper y la memoria, se utilizó el editor en línea ShareLaTeX. En tercer lugar, para la gestión de comunicación y revisión de documentación con cliente del proyecto se utilizó la herramienta Yammer. Finalmente, para la administración del código fuente utilizamos el sistema de control de versiones Git en conjunto con el servicio Github.

2.1.11. Student Outcome K

“Utiliza técnicas, herramientas, metodologías, procesos, frameworks y lenguajes de programación de última generación necesarios para la práctica de la Ingeniería de Software.”

Se planifico el proyecto siguiendo los lineamientos del PMBOK y se finalizó dentro de las restricciones de tiempo establecidas. Se siguió y editó el plan de trabajo siguiendo el framework SCRUM conforme se completaban las historias de usuario correspondientes. Se aplicaron los siguientes procesos de desarrollo en este proyecto. Se generó un Project Charter, se crearon planes de gestión proyecto, se ejecutaron los planes, se controló los riesgos encontrados, se obtuvo el producto final y se cerró el proyecto. Esto cubre las fases correspondientes al ciclo de vida del PMBOK (Inicio, Planeamiento, Ejecución, Control y Cierre).

Se utilizaron diversas herramientas para el desarrollo multiplataforma con los 5 frameworks investigados. Se trabajó con Android Studio, Xcode 8, Visual Studio 2017, Visual Studio Code y Unity para la implementación de las pruebas de concepto. Además, para el trabajo con el enfoque híbrido y el desarrollo en Android y iOS se trabajó con macOS Sierra. Para el despliegue de los servicios REST de la aplicación Cross-device utilizamos el servicio PaaS conocido como AppHarbor.

2.2. Competencias Generales

2.2.1. Comunicación escrita

“Comunicación escrita es la capacidad para construir mensajes con contenido relevante, argumentos sólidos y claramente conectados adecuados a diversos propósitos y audiencia.”

A lo largo del desarrollo del proyecto, hemos elaborado documentos en los que comunicábamos nuestros mensajes a diferentes audiencias, tomando en cuenta que cada uno requiere un tono diferente. Durante la elaboración de nuestro artículo de investigación, plasmamos nuestra investigación en términos técnicos y descriptivos, correspondientes a un trabajo académico serio. A fin de dar a conocer los diversos acuerdos con los involucrados, elaboramos actas en las que se detallaba los puntos acordados.

2.2.2. Comunicación oral

“Capacidad para transmitir oralmente mensajes de manera eficaz dirigidos a diversas audiencias, usando diferentes herramientas que faciliten su comprensión y el logro del propósito.”

Los integrantes del proyecto comunicamos oportunamente el propósito y avance de nuestro proyecto a los diversos interesados. Utilizamos presentaciones formales para las exposiciones con los miembros del comité. Realizamos múltiples reuniones con nuestros profesores gerente y cliente, con los cuales discutíamos mejoras y llegábamos a acuerdos, siempre manteniendo la compostura y la seriedad en la comunicación. Utilizamos herramientas visuales, entregables, gráficos y demás elementos como soporte para expresar nuestro mensaje de forma más clara y didáctica.

2.2.3. Ciudadanía

“Capacidad para valorar la convivencia humana en sociedades plurales, reflexionando acerca de las dimensiones morales de las propias acciones y decisiones, asumiendo la responsabilidad por las consecuencias en el marco del respeto de los derechos y deberes ciudadanos.”

Cada aporte de esta investigación se realizó tomando en cuenta las implicancias que podría tener el desarrollo de nuestra investigación. Consideramos las consecuencias que nuestra investigación podría haber tenido, por ejemplo, sobre las empresas desarrolladoras de las tecnologías que estamos investigando. Nuestro trabajo está orientado a servir como punto de comparación entre las distintas tecnologías disponibles, por lo cual en caso realizáramos mediciones fraudulentas podríamos afectar negativamente las decisiones que una empresa tome con respecto al uso de una tecnología u otra. Nuestro trabajo podría ser utilizado para sustentar una propuesta técnica en beneficio para el Estado, y en caso nuestros números sean erróneos podríamos afectar en la decisión el resultado final de un proyecto. Nuestro informe final se consideraron estos puntos y nos aseguramos de que no hayamos ido en contra de ellos.

2.2.4. Pensamiento innovador

“Capacidad para detectar necesidades y oportunidades para generar proyectos o propuestas innovadoras, viables y rentables. Planifica y toma decisiones eficientes orientadas al objetivo del proyecto”

Parte de todo desarrollo implica considerar las alternativas que se tienen disponibles, y utilizar aquellas que cubran tus necesidades de la mejor forma, en el menor tiempo posible y con menor riesgo en el futuro. Este es un proceso que se tienen que realizar en un gran número de proyectos y emprendimientos, y que no siempre se realizan por falta de recursos. Encontramos dichos problemas y nos encargamos de recopilar toda la información necesaria para ellos. Con nuestra solución ahorramos muchísimas horas de trabajo y ayudamos a tomar decisiones informadas con respecto al uso de tecnologías apropiadas.

2.2.5. Pensamiento crítico

“Capacidad para explorar de manera exhaustiva problemas, ideas o eventos para formular conclusiones u opiniones sólidamente justificadas”

Nuestra investigación promueve que los desarrolladores, en primer lugar, piensen bien acerca de su contexto específico, y las necesidades que tenga su proyecto, para poder aprovechar los resultados de las mediciones que presentamos. Proponemos una serie de dimensiones y escenarios que pueden utilizar como base, pero queda a discreción de cada desarrollador encontrar aquella información que se adecue mejor a sus requerimientos, demostrando así el pensamiento crítico.

2.2.6. Pensamiento cuantitativo

“Capacidad para interpretar, representar, comunicar y utilizar información cuantitativa diversa en situaciones de contexto real. Incluye calcular, razonar, emitir juicios y tomar decisiones con base en esta información cuantitativa”

Nuestra investigación implicó desarrollar experimentos en entornos controlados para llegar a conclusiones objetivas sobre el rendimiento de diferentes tecnologías. Realizamos una investigación científica de esta índole, ya que, haciendo uso de la información recopilada, los desarrolladores podrán tomar decisiones totalmente objetivas sobre cuál tecnología deberían tomar de acuerdo a sus necesidades. Nuestra investigación promueve el pensamiento cuantitativo y la toma de decisiones objetivas.

2.2.7. Manejo de la información

“Capacidad de identificar la información necesaria, así como de buscarla, seleccionarla, evaluarla y usarla éticamente, con la finalidad de resolver un problema”

La primera fase de nuestra investigación consistía en recopilar información acerca de aquellas tecnologías que nos permitían establecer comunicación en tiempo real entre dispositivos móviles. A fin de asegurar la sostenibilidad de nuestra investigación, las analizamos y comparamos en base a diferentes factores, llegando a seleccionar aquellas que cumplieran los requerimientos mínimos de funcionamiento que buscábamos. Procuramos que nuestra selección no diera lugar a ningún tipo de preferencia hacia alguna compañía en particular. Una vez seleccionadas, escogimos herramientas de evaluación que aseguraran que los resultados finales fueran los más objetivos posibles, con la finalidad de encontrar aquella tecnología que resolviera mejor el problema de cada desarrollador.

CAPÍTULO 3: MARCO TEÓRICO

En esta sección, se presentarán y describirán a detalle los conceptos que se identificaron a lo largo del planteamiento, gestión y desarrollo del proyecto.

3.1 CROSS DEVICE INTERACTION

"La interacción Cross-device (XDI) es un tipo de interacción en la cual los usuarios humanos interactúan con múltiples dispositivos de entradas y salidas separadas, donde los dispositivos de entrada serán usados para manipular contenido en dispositivos de salida dentro de espacio de interacción percibido con retroalimentación inmediata y explícita" (Scharf, Wolters, & Herczeg, 2013). Es decir, la interacción Cross-device consiste en hacer uso de un dispositivo para generar interacciones con otros dispositivos completamente separados. Las aplicaciones programadas que soportan este tipo de interacción deben definir cómo los dispositivos se conectarán entre sí, para que la funcionalidad de uno afecte la respuesta en otro dispositivo.

3.2 CROSS PLATFORM DEVELOPMENT

Cross Platform Development hace referencia al desarrollo o ejecución de un Software en más que un solo tipo de Hardware. El desarrollo Cross-platform se basa en una sola base de código, con la cual se puede generar ejecutables para máquinas de distintos sistemas operativos o procesadores. Dos formas que se utilizan hoy en día para lograrlo es la compilación a código de máquina según sistema operativo, utilizado extensamente por C++, y compilación a lenguaje intermedio, popularizado por Java. La compilación a lenguaje intermedio consiste en transformar una base de código inicial a un lenguaje intermedio que es interpretado por una aplicación específica a cada sistema operativo o plataforma.

El desarrollo Cross Platform de aplicaciones móviles está enfocado al ahorro de esfuerzo al realizar múltiples aplicaciones nativas. Es importante describir esta forma de desarrollo, ya que está ligada a la manera en la cual se crea la comunicación entre dispositivos, haciéndolo una arista importante en este ámbito. Generalmente, la opción más común para realizar desarrollos nativos es utilizar tecnologías nativas usando

enfoques de desarrollo propios del desarrollo para cada plataforma. Sin embargo, una de las trabas más grandes en hacer desarrollos nativos es el gran costo que implica, dada la amplia fragmentación de dispositivos disponibles (Aklilu, 2012). A fin de solucionar esos problemas, existen herramientas de desarrollo, tales como Xamarin o Phonegap, las cuales permiten realizar aplicaciones para las plataformas móviles más conocidas usando una sola fuente de código, que se utilizará para generar compilados para todos los sistemas operativos.

3.3 COMUNICACIÓN EN TIEMPO REAL (RTC)

Se establece como comunicación en tiempo real al intercambio instantáneo de información entre un emisor y un receptor. Las características de la comunicación en tiempo real son:

- Rapidez
- Baja tasa de pérdida de información
- Comunicación de punto a punto
- Eficiente a nivel de costos

Existen dos formas grandes formas de categorizar las comunicaciones en tiempo real: por modo de transmisión y por tolerancia a fallos. Dependiendo del modo de transmisión se pueden clasificar en *half duplex* y *full duplex*. La primera consiste en establecer una comunicación bidireccional, pero no en simultaneo. Full duplex, por el otro lado, también es bidireccional, pero permite intercambiar mensajes al mismo tiempo.

Con respecto a la tolerancia a fallos, podemos clasificar a las aplicaciones basadas en comunicación en tiempo real como *soft realtime applications* (SRTA) o *hard realtime applications* (HRTA). Las SRTA toleran, hasta cierto punto, la pérdida de información / calidad durante el intercambio de información. Las HRTA definen un protocolo mucho más estricto, pues busca garantizar la integra transmisión de la información.

Hoy en día la comunicación en tiempo real es aplicada con diversos fines, tales como mensajería instantánea, video conferencia en tiempo real, multicast de contenido multimedia, entre otros.

3.4 ANDROID

Android es un sistema operativo que corre sobre millones de dispositivos a nivel mundial. Es la plataforma con más usuarios, y cuenta con un ritmo de crecimiento que aumenta cada día. Fue desarrollada por la empresa Android Inc., la cual fue posteriormente adquirida por Google y liberada como proyecto open source en 2007⁴.

Figura 2 Arquitectura de Sistemas Android⁵

Este sistema operativo se basa en el kernel de Linux, a través del cual podemos establecer la comunicación con el hardware del dispositivo. Por encima tenemos a Dalvik, la máquina virtual de Java para Android que nos permite ejecutar los compilados necesarios para poder ejecutar las aplicaciones. Asimismo, se tiene acceso a las librerías provistas por Android para facilitar el desarrollo de funcionalidades que interactúen con el sistema operativo (Brahler, 2017).

Para poder desarrollar aplicaciones en este sistema operativo se podía utilizar Eclipse o Android Studio. El año 2015, Android dejó de darle soporte a Eclipse, por lo que Android Studio pasó a ser el IDE de desarrollo por defecto.

⁴ (Android, Android, the world's most popular mobile platform, 2017)

⁵ Imagen tomada de: (Brahler, 2017) Figure 1.1.

El lenguaje por excelencia para el desarrollo en Android siempre ha sido Java, debido a que finalmente la aplicación correrá sobre Dalvik, una adaptación de la máquina virtual de Java. Sin embargo, Android también le da soporte oficial a C++ (por medio del NDK)⁶ y a Kotlin⁷. Es posible utilizar cualquiera de estos tres lenguajes para el desarrollo nativo de aplicaciones Android.

- **Java**: Es una tecnología que se usa para el desarrollo de aplicaciones que convierten a la Web en un elemento más interesante y útil. Java no es lo mismo que Javascript, que se trata de una tecnología sencilla que se usa para crear páginas web y solamente se ejecuta en el explorador⁸.
- **C++ / Android NDK**: Android NDK es un conjunto de herramientas que te permite implementar partes de tu app usando lenguajes de código nativo como C y C++. Para ciertos tipos de apps, esto puede ayudarte a volver a usar las bibliotecas de códigos escritas en esos lenguajes.⁹
- **Kotlin**: Es un lenguaje de programación estáticamente tipado para el desarrollo de aplicaciones multiplataforma. Fue desarrollado por la compañía JetBrains, bajo la modalidad open source. Es un lenguaje que puede correr sobre la JVM, por lo cual es 100% interoperable con Java¹⁰, lo que nos permite realizar proyectos usando ambos lenguajes sin ningún problema.

⁶ (Android, Android NDK, 2017)

⁷ (Android, Kotlin and Android, 2017)

⁸ (Java, 2017)

⁹ (Android, Android NDK, 2017)

¹⁰ (Kotlin, 2017)

3.5 iOS

Es un sistema operativo que corre sobre dispositivos iPad, iPhone y iPhone Touch. El sistema operativo maneja el hardware del dispositivo y provee las tecnologías a ser utilizadas para el desarrollo de las aplicaciones nativas. El SDK de iOS contiene herramientas e interfaces necesarias para desarrollar, instalar, correr, y probar aplicaciones nativas.¹¹

Figura 3 Capas de iOS¹²

Las aplicaciones nativas se construyen sobre los frameworks provistos por Apple para manejar la interfaz (Cocoa Touch) e interactuar con el sistema operativo (Media / Core Services). El desarrollo puede realizarse usando Swift, Objective-C, o ambos.

- **Objective-C**: Es uno de los lenguajes principales para escribir software para macOS y iOS. Es un súper conjunto del lenguaje C y provee capacidades de la programación orientada a objetos en un tiempo de ejecución dinámico. Hereda la sintaxis, tipos primitivos y el flujo de control de C, y añade sintaxis definida para las clases y métodos.¹³
- **Swift**: Es un lenguaje de programación de propósito general, construido usando un enfoque orientado a la seguridad, rendimiento y diseño de patrones de software. Está diseñado para hacer que la escritura y mantenimiento de programas correctos sea más fácil para el desarrollador. Para esto, Swift te permite escribir código seguro, rápido y expresivo. Swift fue lanzado hace tres años por Apple, y en diciembre del 2015 fue liberado como proyecto open source bajo la licencia Apache 2.0. Actualmente, puede ser utilizado para desarrollar aplicaciones sobre iOS, macOS y Linux.

¹¹ (Apple, iOS Technology Overview, 2017)

¹² Imagen tomada de: (Apple, iOS Technology Overview, 2017) Figure 1-1.

¹³ (Apple, Programming with Objective-C, 2017)

3.6 LIBRERÍAS

«Libraries» o bibliotecas en español nacen del concepto de la reutilización de código. Es una práctica que ha estado presente por mucho tiempo; siendo una de las primeras instancias de su uso en 1959 (Wexelblat, 1981).

Conforme su uso se volvió más generalizado, se prefirió el uso de bibliotecas para modularizar aplicaciones. Las aplicaciones grandes y monolíticas eran difíciles de mantener, por lo que una estrategia para volverlas más manejables consiste en utilizar bibliotecas compartidas. Estas son considerablemente más pequeñas y se pueden manejar como componentes individuales que interactúan entre sí.

Las bibliotecas compartidas pueden ser usadas para compartir funcionalidad entre equipos y servicios, así como también servir como colecciones de utilidades (Newman, 2015). Por otro lado, ya que lo único que limita la interacción de bibliotecas es el lenguaje en el que fueron programados, se puede obtener y combinar librerías de terceros para extender la funcionalidad de una lógica de negocio o programa. Entonces, las bibliotecas se pueden utilizar para tercerizar funcionalidad que no es crítica al funcionamiento primario de una aplicación, así como también para facilitar el desarrollo de este.

En la actualidad, un gran porcentaje de software incluye bibliotecas de terceros. En un estudio realizado que toma como muestra veinte proyectos hechos en Java, el 90 % uso bibliotecas de terceros (Heinemann & Robillard, 2012).

CAPÍTULO 4: ESTADO DEL ARTE

4.1 REVISION DE LA LITERATURA

Para el presente trabajo se realizaron investigaciones en el ámbito académico referentes al estudio de comunicación entre dispositivos (cross device interactions). La información encontrada nos ayuda a mostrar el ámbito actual en el que se encuentra nuestro tema de investigación.

Las investigaciones sobre interacciones entre diferentes dispositivos y diferentes plataformas han estado aumentando de manera proporcional a las facilidades acceso a dispositivos como smartphones y tabletas en el mundo. Los siguientes cuatro artículos pretenden validar la importancia de la comunicación entre dispositivos para el desarrollo de aplicaciones multiplataforma, manipulación de dispositivos y experiencia de usuario.

4.2 MANIPULACION DE DIVERSOS DISPOSITIVOS

El primer artículo titulado “Conductor: Enabling and Understanding Cross-Device Interaction” (Hamilton & Wigdor, 2014), define que se ha incrementado la diversidad entre diferentes gamas de dispositivos móviles como smartphones o tablets, generando una necesidad implícita de manejo de información entre ellos. Para ello, en el 2014, Peter Hamilton y Daniel Wigdor, desarrollaron “Conductor”, un framework prototipo que permitía la comunicación de aplicaciones cross-device, mediante las cuales los usuarios podían compartir información, ejecutar tareas en cadena y manejar sesiones entre dispositivos.

Según el artículo, “Conductor” es un sistema prototipo que combina una serie de técnicas de interacción diseñadas para permitir que un solo usuario, mediante aplicaciones cross-device, administre relaciones entre dispositivos y facilite la funcionalidad en cadena. Para su investigación, trataron de solucionar los siguientes obstáculos:

- Cada nuevo dispositivo podría enriquecer la experiencia del usuario, también agrega complejidad a la red de conexiones entre estos dispositivos

- La falta de métodos de interacción que faciliten las experiencias fáciles entre dispositivos sin necesidad de sensores.
- La ausencia de interacciones que permiten el encadenamiento de la funcionalidad del dispositivo, para facilitar la construcción de flujos de trabajo ad hoc para una determinada experiencia de dispositivos.
- El poco estudio formal para explorar el uso de muchos dispositivos cuando las interacciones entre dispositivos son totalmente apoyadas en un DUI (interfaces de usuario distribuidas).

Por otro lado, el proyecto del artículo también se basa en áreas de investigación centradas en la interacción multi-dispositivo: Digital Room (salones digitales), Small and Large Displays (interacciones generales entre pantallas pequeñas, típicamente portátiles, y grandes) y Proxemics (las interacciones prosémicas, sobre el conocimiento del espacio físico). Estas interacciones están). Siendo estas especificadas como:

- **Digital Room:** El uso de grandes espacios físicos que son el lugar de estaciones de trabajo multiterminales, a menudo ubicadas alrededor de una gran pantalla compartida.
- **Small and Large Displays:** La combinación dispositivos de factor de forma pequeños y grandes (pantallas pequeñas como adjuntos de pantallas más grandes). Según el artículo, Fitzmaurice describió su visión de los pequeños dispositivos portátiles que actúan simplemente como nodos para puntos de información que podrían ser transportados físicamente entre instalaciones de dispositivos más grandes.
- **Proxemics:** Varios proyectos han utilizado la proximidad física y la orientación relativa para facilitar la gestión de la información entre dispositivos.

“*Conductor*” fue diseñado con el objetivo de proporcionar técnicas de interacción genéricas que permitan discrecionalmente varias formas de interacción entre dispositivos. Esto permitirá al usuario aprovechar simultáneamente varios dispositivos y hacer la transición fácilmente entre ellos. Para eso, del lado de la implementación se hizo uso de web sockets para la administración de la comunicación mediante mensajes en formato JSON (JavaScript Object Notation), siendo un servidor el encargado de mantener conectados los dispositivos y distribuir la comunicación entre estos. Durante el desarrollo de este framework, se definieron los siguientes tipos de interacción:

- Comunicación entre dispositivos a través de “Cues”: El *conductor* simplifica esto difundiendo la información a cada dispositivo (broadcasting), permitiendo que el dispositivo objetivo sea elegido físicamente en lugar de seleccionado de un menú. Para *Conductor*, una “Cue” muestra información generada por las actividades de otros dispositivos, sobre las cuales el usuario puede actuar o ignorar. El sistema de referencia es conceptualmente similar a un portapapeles de varios elementos, en el que se puede pensar que la radiodifusión “copia” el elemento y pulsa su señal “pega” en el dispositivo de destino.
- Elección de transmisiones para dispositivos: Un usuario también puede optar por “pasar” la información a otro dispositivo dirigiéndola específicamente desde el menú contextual. La orientación de un dispositivo con un elemento hace que ejecute inmediatamente la acción predeterminada apropiada para ese elemento (lo mismo que si se hubiera difundido y la señal hubiese sido pulsada)
- Fusión de funcionalidades mediante “Duets”: Para habilitar experiencias instantáneas entre dispositivos requiere que se forme un enlace entre estos. En *Conductor*, estas combinaciones se denominan “duets”. Los “duets” son instancias que enlazan diversos dispositivos haciendo que la información que se muestre sea compartida, pero el uso se mantenga independiente. El usuario puede arrastrar cualquier “cue” de la lista en el lado izquierdo de la pantalla a una lista de “duets” mostrados en el lado derecho y así extenderlos ya que cada uno de los primeros contiene la identidad del dispositivo fuente y la aplicación, haciendo posible su integración.
- Manejar Sesiones a través de dispositivos: Una preocupación clave con las interfaces de usuario distribuidas es asegurar que el usuario siempre tenga un conocimiento suficiente de la asignación de información a las pantallas, de manera que la ventana de visualización que deseen en ese momento esté inmediatamente lista para usar.
- Agregación de dispositivos a un “Symphony”: El framework distinguía entre el conjunto de dispositivos disponibles para el usuario, y el conjunto de dispositivos activamente en uso representaba una “*Symphony*”. Los mecanismos por los cuales se podían agregar dispositivos a este eran por medio de:
 - Códigos QR: Llamadas de códigos QR a través de las pantallas de los dispositivos
 - NFC: Estableciendo conexión mediante radiofrecuencia

- Bumping: Detección por medio de acelerómetros
- Proxies: Para dispositivos que no poseyeran sensores

Finalmente, en el artículo se describe el uso de un experimento para validar la propuesta del framework *Conductor*. Se realizó un estudio con 8 participantes, a los cuales se les dio una tarea que podían realizar con el uso de varios dispositivos. Para ello también se les enseñó como poder utilizar “*Conductor*”, dándoles como opción poder utilizar cualquier medio para realizar la tarea y el libre uso de la cantidad de dispositivos. El resultado del experimento arroja claramente la importancia del uso de aplicaciones cross-device y sus funcionalidades. Por un lado, con respecto a la tarea, todos los participantes tomaron ventaja de múltiples tabletas. Cuatro de los 8 participantes utilizaron las 10 tabletas. De los 4 participantes restantes, todos usaron al menos 5 dispositivos durante la tarea. Todos afirmaron que su número elegido era ideal, excepto que 2 de los participantes que usaron las 10 tabletas indicaron que habrían usado más si hubieran estado disponibles. Por otro lado, con respecto al uso de *Conductor*, de los 8 participantes, sólo un participante no utilizó ninguna de las herramientas de *Conductor* proporcionadas, aunque si hizo uso intensivo de múltiples tabletas de esta manera. Este estudio valida la importancia de la comunicación entre dispositivos para el desarrollo de actividades Cross-device.

4.3 COMPLEJIDAD DEL DESARROLLO CROSS-DEVICE

El segundo artículo lleva el título: “*Enhancing Cross-Device Interaction Scripting with Interactive Illustrations*” (Chi, Li, & Hartmann, 2016). En este se detalla la creación de DemoScript, una herramienta que analiza automáticamente la interacción entre dispositivos y la visualiza paso a paso. Debido a la aparición de tecnologías móviles y wearable, la cantidad de formas de interacción por medio de dispositivos ha aumentado, y según las capacidades de estos dispositivos los consumidores pueden llegar a tener un ecosistema multi-dispositivo. Estas diferentes interacciones han creado nuevos escenarios para la implementación.

Según el artículo, la implementación Cross-device los siguientes retos: en primer lugar, es difícil soportar la gran variedad de dimensiones y capacidades de cada dispositivo; en segundo lugar, es difícil expresar los flujos de interacción que abarcan múltiples dispositivos; y, en tercer lugar, la prueba de interacciones entre dispositivos sigue siendo difícil. Debido a estos retos, su objetivo para el trabajo es ilustrar el comportamiento del código con ejemplos concretos y generar código mediante directa manipulación visual.

Según el enfoque de los autores, los desarrolladores se mueven entre la lógica de aplicación abstracta y los ejemplos de ejecuciones concretas en tiempo de ejecución. Sus contribuciones para problema fueron las siguientes:

- Un nuevo enfoque para programar interacciones entre dispositivos, basado en un acoplamiento e ilustración visual interactiva de comportamientos.
- Una nueva visualización de la lógica de interacción y un conjunto de conclusiones iniciales sobre la utilidad y utilidad de los guiones gráficos de dispositivos cruzados.
- Un conjunto de métodos para analizar un script de interacción entre dispositivos en un marco de interfaz de usuario dado.

En este sentido, el artículo nos indica que el desarrollo de este proyecto abarca cuatro tópicos de investigación:

- Editores de código mejorados: Las herramientas que visualizan diferentes aspectos del código fuente y permiten la manipulación directa de las abstracciones pueden reducir con éxito la complejidad de la programación.

- IDEs para desarrollo Cross-device: Según los autores, en la investigación de dispositivos múltiples, el soporte de creación tiende a centrarse en la creación gráfica, de manipulación directa o en marcos de programación sin editores visuales.
- Storyboards en diseño de interacción: Los diagramas de flujo de pantalla o storyboards de interfaz de usuario se utilizan ampliamente en la práctica para ayudar a los desarrolladores a visualizar el flujo y pensar en la lógica de una aplicación.
- Iterando con ejemplos: Para los autores del artículo, los ejemplos ayudan a concretar e ilustrar conceptos abstractos, y pueden servir como una guía para explorar espacios de diseño. La investigación ha aportado sistemas centrados en el ejemplo para utilizar ejemplos para buscar, filtrar y explorar espacios de información.

En la figura siguiente podemos ver el StoryBoard de DemoScript que nos muestra como la interacción que se puede lograr mediante el uso de la herramienta al momento de realizar un mapeo de las diversas interacciones entre dispositivos Cross-device.

Figura 4 DemoScript StoryBoard¹⁴

¹⁴ Imagen tomada de: (Chi, Li, & Hartmann, 2016) Figure 1

Figura 5 Promedio de completitud de tiempo para cada sub-tarea de DemoScript¹⁵

Finalmente, validaron la versatilidad de la herramienta mediante una prueba en la cual tenían que reparar y realizar pruebas de un script. Según los resultados que se dieron en el experimento (los cuales se pueden ver en la figura anterior), la mayoría de los desarrolladores tomaron menos tiempo en realizar las tareas en una línea base frente al uso de DemoScript, ya que los ayuda a simplificar las tareas de programación para comportamientos de interacción complejos. Según los autores muchos componentes de DemoScript pueden aplicarse a otros tipos de desarrollo de aplicaciones. Técnicas para seleccionar elementos interactivos (dispositivos) y proporcionar retroalimentación de la interfaz de usuario y escuchar eventos de usuario (como la entrada por contacto o por sensor) son comunes en marcos de interacción o móviles como Android.

¹⁵ Imagen tomada de: (Chi, Li, & Hartmann, 2016) Figure 7

4.4 COMPARACION DE TECNICAS DE MOVIMIENTO CROSS-DEVICE

El tercer artículo académico se titula: “A Comparison of Techniques for Cross-Device Interaction from Mobile Devices to Large Displays” (Paay, y otros, 2016). En este se toma en consideración un estudio de cuatro técnicas para el movimiento de información de un dispositivo móvil a uno de mayor visualización para medir la efectividad, eficiencia y margen de error.

En el trabajo realizado para esta investigación, se tomaron en cuenta diferentes técnicas de interacción mediante gestos, iniciados a partir de un dispositivo móvil en la mano que conversa con un dispositivo mayor. Las cuatro técnicas a revisar son:

- **Pinch:** La técnica Pinch se incluyó para imitar la acción natural de recoger un objeto real, por ejemplo, un trozo de papel, y moverlo a otro lugar.

Figura 6 La Técnica Pinch¹⁶

- **Swipe:** Se describe la técnica como "interacción entre dispositivos con el toque y el aire apuntando" y el movimiento de deslizamiento se describe como el “deslizamiento en la pantalla táctil”.

Figura 7 La técnica swipe¹⁷

¹⁶ Imagen Tomada de: (Paay, y otros, 2016) Figure 1

¹⁷ Imagen tomada de: (Paay, y otros, 2016) Figure 2

- **Throw:** La técnica Throw es una combinación de una técnica para apuntar, es decir, usando una mano como un cursor en el aire, y la moviendo el brazo utilizado para compartir información en grandes pantallas públicas.

Figura 8 La Técnica throw¹⁸

- **Tilt:** La técnica de inclinación se utiliza para transferir un objeto desde una pantalla grande al teléfono inteligente del usuario. La forma inclinación usada en la investigación es una copia de esta, pero invertida.

Figura 9 La Técnica Tilt¹⁹

Al finalizar la revisión de la literatura, los autores encontraron que la investigación cuantitativa sobre interacciones entre dispositivos para transferir datos de un dispositivo móvil a una pantalla grande está actualmente limitada en HCI.

Para poder aportar nuevos conocimientos sobre las fortalezas y debilidades comparativas de las diferentes técnicas de interacción entre dispositivos, crearon un experimento para medir empíricamente la tasa de éxito de éxito, el tiempo de transferencia de datos y la distancia de error para objetivos perdidos de cuatro diferentes (Pinch, Swipe, Throw y Tilt) en una situación de laboratorio. En esta prueba, a cada participante se le dieron tres intentos de práctica por técnica, para familiarizarse con ella. Después de la fase de

¹⁸ Imagen tomada de: (Paay, y otros, 2016) Figure 3

¹⁹ Imagen tomada de: (Paay, y otros, 2016) Figure 4

práctica, aparecería un objetivo de calibración para poder controlar la distancia cubierta. Después de la fase de práctica, los participantes irían a través de la prueba, compuesto por 18 objetivos. Al final, cada participante experimentó tres rondas de práctica + un objetivo de calibración (para cada técnica) que no se incluyeron en los resultados, y (9 pequeños + 9 objetivos grandes) x (4 técnicas) = 72 objetivos, como datos registrados. La prueba por participante tomó un promedio de 15 minutos. No se permitieron intentos repetidos en un objetivo y el sistema pasó al siguiente objetivo independientemente de si el participante golpeó el objetivo o no lo hizo. De estas pruebas se encontraron los siguientes resultados:

- **Efectividad:** En este estudio, se definió la efectividad como el número de intentos exitosos por usuario, por técnica. Los intentos de éxito promedio por usuario para cada técnica, para objetivos pequeños y grandes, respectivamente, se enumeran en la siguiente figura.

Figura 10 Efectividad : Promedio exitoso de intentos por usuario por técnica, para blancos grandes y pequeños²⁰

- **Eficiencia:** Por eficiencia, definieron el tiempo que se tarda en completar una acción según lo requerido por cada técnica. El número de intentos, el tiempo medio para

²⁰ Imagen tomada de: (Paay, y otros, 2016) Figure 8

alcanzar un objetivo en segundos y las desviaciones estándar para cada técnica y para ambos tamaños de objetivo, se muestran en la siguiente figura.

Figura 11 Eficiencia : Promedio de tiempo toma por blanco por cada técnica para blancos pequeños y grandes²¹

- **Margen de Error:** Definieron el tamaño de error como la distancia en píxeles que el cursor tenía de un objetivo cada vez que un participante perdía el objetivo. Las medias y desviaciones estándar para la distancia desde la meta en píxeles se pueden ver en la siguiente figura.

Figura 12 Margen de Error : Distancia promedio en pixeles por técnica por blancos pequeños y grandes²²

Finalmente, los resultados encontrados del artículo demuestran que Swipe se desempeñó mejor en todas las pruebas. Swipe fue la técnica más eficaz, con el mayor número de intentos exitosos, tanto para objetivos pequeños como grandes. Al mismo tiempo, Swipe

²¹ Imagen tomada de: (Paay, y otros, 2016) Figure 9

²² Imagen tomada de: (Paay, y otros, 2016) Figure 10

fue la técnica más eficiente, siendo el más rápido de usar, tanto para los objetivos pequeños y grandes. La técnica de Swipe también fue la más precisa al mirar el tamaño de los errores encontrados. Una vez más, este fue el caso tanto para los objetivos pequeños como para los grandes. Según eso, los autores concluyen que para diseñar una técnica de interacción Cross-device utilizada para enviar datos de un dispositivo móvil de mano a una pantalla grande, una técnica de swiping, como la presentada aquí, debería ser una primera consideración. En términos de efectividad, la técnica de inclinación realizó lo peor, y especialmente con objetivos pequeños. En términos de eficiencia y tamaño de error, la técnica Pinch fue la más lenta y también la más imprecisa. También se encontró que el tamaño objetivo importaba considerablemente para todas las técnicas, lo que confirma la investigación anterior, pero sorprendentemente cuando la gente perdió un objetivo, perdieron blancos más grandes en mayor medida que los más pequeños.

CAPÍTULO 5: DESARROLLO DEL PROYECTO

En este capítulo, se explicará a detalle el trabajo realizado durante el desarrollo del proyecto. En primer lugar, se explicarán las tecnologías de comunicación Cross-device elegidas. Luego, se describirá la implementación de las pruebas de concepto realizadas con las mismas, así como la aplicación de conectividad desarrollada y se expondrán los diagramas de arquitectura de alto nivel con los mecanismos de conexión. Posteriormente, se mostrarán los escenarios de prueba y las mediciones tomadas a partir de este con respecto al tiempo de envío y recepción de la información para cada tipo de tecnología. Finalmente, se presentará el desarrollo del análisis comparativo, las implicancias que tiene y el resultado de la encuesta de validación con desarrolladores de aplicaciones móviles.

5.1 TECNOLOGIAS DE COMUNICACIÓN

La siguiente sección lista las diferentes tecnologías que fueron medidas para el análisis. Para la selección de tecnologías, indagamos sobre diferentes medios de comunicación real time, tomando en consideración la importancia de estas en el mercado y el tiempo de permanencia. Por otro lado, se tomaron las tecnologías utilizadas en la tesis de Chiolino y Contreras para ser parte del análisis comparativo. Estas se volvieron a compilar y se incluyeron a la toma de mediciones.

5.1.1 DESARROLLO DE PROYECTO BASE

Este proyecto se desarrolló teniendo en consideración que sin importar el comportamiento interno de cualquier tecnología que nos permita realizar una comunicación en tiempo real entre dispositivos móviles, las funcionalidades más básicas que cualquier de estas debería cumplir son:

Envío de mensajes
Recepción de mensajes

A fin de ejecutar las pruebas de cada tecnología bajo el mismo escenario, definimos una estructura que nos permitiría generar compilados para cada tecnología utilizando un solo proyecto base. Tecnologías que no puedan ser desarrolladas en lenguajes nativos (Java / Kotlin para Android, Swift / Objective-C para iOS), no pueden ser incluidas en este proyecto base. Para las tecnologías que hemos desarrollado, se tuvo que desarrollar SignalR en un proyecto aparte, ya que su desarrollo se basa en C#.

5.1.1.1 Android

Para lograr esto, en primer lugar, creamos la interfaz *ChatRepository* que define dos funciones, una para el envío y otra para la recepción:

```
interface ChatRepository{  
    fun send(message: String)  
    fun receive(): Flowable<Message>  
}
```

El método *send* permite al cliente enviar una cadena de texto, para que sea distribuido a los demás dispositivos. El método *receive* devuelve un tipo de dato *Flowable<Message>*. Esto representa un flujo de mensajes, lo cual nos permite detectar cada vez que otro dispositivo nos haya enviado un mensaje. Para poder manejar el flujo de mensajes con *Flowable*, hacemos uso de la librería RxJava.

Una vez definida la interfaz, la implementación para cada tecnología extenderá de esta interfaz. Hemos creado cinco implementaciones:

AblyRepository
FirebaseRepository
PubnubRepository
LightstreamerRepository
RabbitMQRepository

Creando las variantes

Usaremos este mismo proyecto para generar las aplicaciones Android de cada tecnología, garantizando que cada una de ellas se esté ejecutando bajo un escenario común.

El primer paso es entrar al archivo “build.gradle” y definir cada una de las variaciones que podría tener el app. Para cada variación, indicamos un sufijo y un nombre.

```

productFlavors{
    general{
        resValue "string", "app_name", "Demo Chat"
    }

    firebase{
        applicationIdSuffix ".firebase"
        versionNameSuffix "-firebase"
        resValue "string", "app_name", "Demo Firebase"
    }

    ably{
        applicationIdSuffix ".ably"
        versionNameSuffix "-ably"
        resValue "string", "app_name", "Demo Ably"
    }

    pubnub{
        applicationIdSuffix ".pubnub"
        versionNameSuffix "-pubnub"
        resValue "string", "app_name", "Demo Pubnub"
    }

    lightstreamer{
        applicationIdSuffix ".lightstreamer"
        versionNameSuffix "-lightstreamer"
        resValue "string", "app_name", "Demo Lightstreamer"
    }

    p2p{
        applicationIdSuffix ".p2p"
        versionNameSuffix "-p2p"
        resValue "string", "app_name", "Demo P2PKit"
    }

    rabbitmq{
        applicationIdSuffix ".rabbitmq"
        versionNameSuffix "-rabbitmq"
        resValue "string", "app_name", "Demo RabbitMQ"
    }
}

```

Inyectando los repositorios

Creamos una carpeta para cada una de las tecnologías, al mismo nivel del módulo app/src. Dentro de estas, creamos la clase *ChatInjection*, la cual devolverá estáticamente la implementación adecuada del repositorio. Por ejemplo:

```

class ChatInjection{
    companion object{
        fun getRepo(context: Context): ChatRepository = ChatFactory.create(context, ChatProviders.ABLY)
    }
}

```

Al realizar esto, al momento de compilar el proyecto podemos escoger la variante que necesitemos, e inmediatamente generaría la aplicación utilizando el repositorio que necesita.

5.1.1.2 iOS

De forma similar a como lo hicimos en Android, creamos el protocolo ChatRepository que define dos funciones, una para el envío y otra para la recepción:

```
protocol ChatRepository{  
    func send(message: String)  
    func receive() -> Observable<TextEntry>  
}
```

El método send permite al cliente enviar una cadena de texto, para que sea distribuido a los demás dispositivos.

El método receive devuelve un tipo de dato *Observable<Message>*. Esto representa un flujo de mensajes, lo cual nos permite detectar cada vez que otro dispositivo nos haya enviado un mensaje. Para poder manejar el flujo de mensajes con Observable, hacemos uso de la librería RxSwift.

Una vez definido el protocolo, la implementación para cada tecnología extenderá de este protocolo. Hemos creado cinco implementaciones:

AblyRepository
FirebaseRepository
PubnubRepository
LightstreamerRepository
RabbitMQRepository

Definimos además una clase llamada *ChatFactory* a la cual únicamente le pasamos el tipo de repository que necesitamos, y nos devuelva una nueva instancia de la implementación que solicitamos.

Configuración de los Targets

Usaremos este mismo proyecto para generar las aplicaciones iOS de cada tecnología, garantizando que cada una de ellas se esté ejecutando bajo un escenario común. Para lograrlo, la aplicación estará basada en múltiples “targets”.

De acuerdo a la documentación de Apple, un target especifica una distribución única de un producto. Un proyecto puede contar con varios targets, pero cada uno de ellos puede

especificar un conjunto de archivos e instrucciones con los se realizará el proceso de construcción del app, con lo cual se generará una distribución específica para dicho target.

Los pasos a seguir para habilitar un target para cada tecnología fueron:

1. Creamos una carpeta “Targets” en el proyecto, y además creamos carpetas adicionales para cada tecnología.
2. Dentro de la carpeta de cada tecnología, creamos dos archivos: *ChatInjection.swift* y *Info.plist*
3. En el archivo *ChatInjection.swift*, creamos una variable que devolverá la implementación del repositorio para la tecnología que necesitamos.

```
class ChatInjection {  
    static let repository = ChatRepositoryFactory.create(type: .rabbitmq)  
}
```

4. En el archivo *Info.plist*, definimos las configuraciones que queremos que apliquen para nuestro target. En nuestro caso, todos los targets comparten las mismas configuraciones.
5. En el archivo *.xcproject*, encontramos un único target la primera vez. Lo seleccionamos, y lo duplicamos.
6. Una vez duplicado, le cambiamos el nombre presionando la tecla “Enter” y editando el campo.
7. Al duplicar el target, se habrá generado una copia del *Info.plist* original. Lo eliminamos del proyecto.
8. Le asignamos el archivo *Info.plist* que creamos en la carpeta propia de cada tecnología. En la sección “Identity”, editamos el campo *Bundle Identifier* del proyecto. Es importante que cada nuevo target tenga un identificador diferente.
9. En la sección *App Icons and Launch Images* le asignamos íconos propios a cada target.
10. Repetimos los pasos 5-9 para cada tecnología.
11. Regresamos a la carpeta que creamos para cada tecnología. Seleccionamos los archivos *ChatInjection.swift* e *Info.plist*, y abrimos el *File Inspector*. En la sección “Target Membership”, marcamos únicamente el target específico a la tecnología.
12. Repetimos el paso anterior para cada tecnología.
13. Al terminar, volvemos a verificar que cada *ChatInjection.swift* e *Info.plist* apunte a un solo target.

5.1.3 IMPLEMENTACION DE TECNOLOGIAS INVESTIGADAS

Definimos las 6 tecnologías siguientes como objeto de estudio y adoptamos 5 tecnologías adicionales de un proyecto anterior para tener un panorama más completo de la interacción entre dispositivos.

5.1.3.1 ABLY

DEFINICIÓN

Aby es una plataforma de envío de información en tiempo real. Provee librerías listas para ser utilizadas por las aplicaciones cliente (Aby, 2017).

CARACTERÍSTICAS

Tabla 3 Características de Aby

	Android	iOS
IDE de Desarrollo	Android Studio 2.3 o superior	Xcode 8.3 o superior
Versión de la librería	1.0.0	0.8.9
Lenguajes de programación compatibles	Java Kotlin	Swift Objective-C
Versiones de Android soportadas	API 14 o superior	iOS 8

DIAGRAMAS

Figura 13 Diagrama de Componentes - Ably

Figura 14 Diagrama de Despliegue - Ably

Figura 15 Diagrama de Secuencia - Ably

IMPLEMENTACIÓN

Android

1. En el archivo *build.gradle* del módulo app, ir a la sección *dependencies* y añadir la dependencia al proyecto:

```
compile 'io.ably:ably-android:1.0.0'
```

2. Configurar Ably usando la llave que ellos te proveen, y el nombre del canal por el cual transmitirás los mensajes.

```
private val realtime: AblyRealtime = AblyRealtime("YOUR_ABLY_KEY")
private val channel = realtime.channels.get("YOUR_CHANNEL_NAME")
```

3. Suscribirse al canal, para poder realizar las operaciones que deseemos con el mensaje que recibamos

```
channel.subscribe {
    val content = it.data.toString()
    //handle the message as you wish
}
```

4. Para enviar mensajes, invocar el método *publish* sobre el canal que creamos:

```
channel.publish("send", "your message")
```

En este proyecto, realizamos dicha implementación siguiendo el patrón Repository, invocando los métodos de subscripción / publicación en los lugares correspondientes. El resultado final es la clase *AblyChatRepository*

```
class AblyChatRepository() : ChatRepository{
    override val name: String get() = "Ably Chat"
    private val realtime: AblyRealtime = AblyRealtime("YOUR_ABLY_KEY")
    private val channel = realtime.channels.get("YOUR_ABLY_CHANNEL")
    private val messageSubject = PublishSubject<Message>.create()

    init {
        channel.subscribe {
            val content = it.data.toString()
            val message = ChatUtils.parseMessage(content)
            messageSubject.onNext(message)
        }
    }

    override fun send(message: String) {
        channel.publish("send", ChatUtils.createData(message))
    }

    override fun receive(): Flowable<Message> {
        return messageSubject.toFlowable(BackpressureStrategy.BUFFER)
    }
}
```

iOS

1. En el archivo Podfile, agregamos la dependencia de Ably

```
pod 'Ably'
```

2. Configurar Ably usando la llave que ellos te proveen, y el nombre del canal por el cual transmitirás los mensajes.

```
private static let API_KEY = "YOUR_API_KEY"
private static let CHANNEL_NAME = "chat"
let client = ARTRealtime(key: API_KEY)
let channel = client.channels.get(AblyChatRepository.CHANNEL_NAME)
```

3. Suscribirse al canal, para poder realizar las operaciones que deseemos con el mensaje que recibamos

```
channel?.subscribe { message in
    let now = String(Int((Date().timeIntervalSince1970 * 1000.0).rounded()))
    guard let data = message.data as? String else {
        return
    }
    //do something with the data
}
```

4. Para enviar mensajes, invocar el método *publish* sobre el canal que creamos:

```
channel?.publish("send", data: "your message")
```

En este proyecto, realizamos dicha implementación siguiendo el patrón Repository, invocando los métodos de subscripción / publicación en los lugares correspondientes. El resultado final es la clase *AblyChatRepository*

```
class AblyChatRepository : ChatRepository{

    private static let API_KEY = "gZ_l9g.y5d3Tw:kSJRo2CIeXcw9w2p"
    private static let CHANNEL_NAME = "chat"
    let client = ARTRealtime(key: API_KEY)
    var channel : ARTRealtimeChannel?
    private var messagesSubject = PublishSubject<TextEntry>()

    func name() -> String { return "Ably Chat" }
    init() {

        client.connection.on { state in }
        channel = client.channels.get(AblyChatRepository.CHANNEL_NAME)
        channel?.subscribe { message in
            let now = String(Int((Date().timeIntervalSince1970 * 1000.0).rounded()))
            print("Llegada: \(now)")
            guard let data = message.data as? String else {
                return
            }

            if let entry = ChatUtils.from(data: data){
                self.messagesSubject.onNext(entry)
            }
        }
    }
}
```

```
    }  
}  
  
func send(message: String){  
    channel?.publish("send", data: ChatUtils.createData(for: message))  
}  
  
func receive() -> Observable<TextEntry>{  
    return messagesSubject.asObservable()  
}  
  
func color() -> String {  
    return "#f79e20"  
}  
}
```

5.1.3.2. FIREBASE

DEFINICIÓN

- **Firebase:** plataforma de Google, que ofrece a los desarrolladores de aplicaciones móviles / web una suite de herramientas para el desarrollo, mantenimiento y monitoreo.
- **Firebase Database:** una de las tecnologías ofrecidas por Firebase. Te permite almacenar información en una base de datos no relacional. Los clientes de Firebase Database pueden suscribirse a esta base de datos, para ser notificados cada vez que ocurra algún cambio sobre esta.

CARACTERÍSTICAS

Tabla 4 Características de Firebase

	Android	iOS
IDE de Desarrollo	Android Studio 2.3 o superior	Xcode 8.3 o superior
Versión de la librería	10.0.2	3.14.0
Lenguajes de programación compatibles	Java Kotlin	Swift Objective-C
Versiones de Android soportadas	API 14 o superior	iOS 8

DIAGRAMAS

Figura 16 Diagrama de Componentes - Firebase

Figura 17 Diagrama de Despliegue - Firebase

Figura 18 Diagrama de Secuencia - Firebase

IMPLEMENTACIÓN

Android

1. En el archivo build.gradle del módulo [app](#), ir a la sección dependencies:

- Añadir la dependencia al proyecto:

```
compile 'com.google.firebase:firebase-database:10.2.0'
```

- Agregar el archivo google-services.json que se obtiene desde la consola de Firebase, e incluirlo en el módulo app del proyecto
- Al final del archivo, agregar el plugin de google services:

```
apply plugin: 'com.google.gms.google-services'
```

2. Crear una referencia al base de datos, indicando el nombre del nodo del cual se desea publicar / recibir mensajes.

```
private val mDatabase: DatabaseReference =  
    FirebaseDatabase.getInstance().reference.child("chat")
```

3. Suscribirse al canal, para poder realizar las operaciones que deseemos con el mensaje que recibamos

```
val childEventListener = object : ChildEventListener {  
  
    override fun onChildAdded(dataSnapshot: DataSnapshot,  
                             previousChildName: String?) {  
        val message = dataSnapshot.getValue(..)  
        //handle the message as you wish  
    }  
  
    override fun onChildChanged(dataSnapshot: DataSnapshot,  
                               previousChildName: String?) {  
  
    }  
  
    override fun onChildRemoved(dataSnapshot: DataSnapshot) {}  
  
    override fun onChildMoved(dataSnapshot: DataSnapshot,  
                             previousChildName: String?) {}  
  
    override fun onCancelled(databaseError: DatabaseError) {}  
}  
mDatabase.addChildEventListener(childEventListener)
```

4. Para enviar mensajes, invocar el método push sobre la referencia a la base de datos:

```
mDatabase.push().setValue("your message")
```


En este proyecto, realizamos dicha implementación siguiendo el patrón Repository, invocando los métodos de subscripción / publicación en los lugares correspondientes. El resultado final es la clase `FirebaseRepository`:

```
class FirebaseChatRepository() : ChatRepository {
    override val name: String get() = "Firebase"

    private val mDatabase: DatabaseReference = FirebaseDatabase
                                                .getInstance()
                                                .reference
                                                .child("chat")

    private val messageSubject: PublishSubject<Message> = PublishSubject.create()

    init {
        val childEventListener = object : ChildEventListener {
            override fun onChildAdded(dataSnapshot: DataSnapshot, previousChildName: String?) {

                val data = dataSnapshot.getValue(ChatUtils.SendMessage::class.java)
                val message = ChatUtils.parseMessage(data)
                if (message != null){
                    messageSubject.onNext(message)
                }
            }

            override fun onChildChanged(dataSnapshot: DataSnapshot,
                                         previousChildName: String?) {}

            override fun onChildRemoved(dataSnapshot: DataSnapshot) {}

            override fun onChildMoved(dataSnapshot: DataSnapshot, previousChildName: String?) {}

            override fun onCancelled(databaseError: DatabaseError) {}
        }
        mDatabase.addChildEventListener(childEventListener)
    }

    override fun send(message: String) {
        ChatUtils.createSendMessage(message).let {
            mDatabase.push().setValue(it)
        }
    }

    override fun receive(): Flowable<Message> {
        return messageSubject.toFlowable(BackpressureStrategy.BUFFER)
    }
}
```

iOS

1. En el archivo Podfile, agregamos la dependencia de Firebase

```
pod 'Firebase'
pod 'Firebase/Database'
```

2. Agregar el archivo `GoogleService-Info.plist` que se obtiene desde la consola de Firebase al proyecto.
3. Configurar Firebase usando la llave que ellos te proveen, y el nombre del canal por el cual transmitirás los mensajes.

```
FIRApp.configure()
let ref = FIRDatabase.database().reference()
```

4. Suscribirse a los cambios sobre la base de datos, para poder realizar las operaciones que deseemos con el mensaje que recibamos:

```
let textsRefs = ref.child(CHANNEL_NAME)

textsRefs.observe(.childAdded, with: { (snapshot) -> Void in
    if let info = snapshot.value as? [String: Any] {
        //do something with the data
    }
})
```

5. Para enviar mensajes, invocar el método *setValue* sobre la referencia a la base de datos:

```
self.ref.child(CHANNEL_NAME)
    .childByAutoId()
    .setValue("your message")
```

En este proyecto, realizamos dicha implementación siguiendo el patrón Repository, invocando los métodos de subscripción / publicación en los lugares correspondientes. El resultado final es la clase *FirebaseChatRepository*

```
class FirebaseChatRepository : ChatRepository{

    private let CHANNEL_NAME = "chat"
    private var messagesSubject = PublishSubject<TextEntry>()
    private var ref : FIRDatabaseReference!

    func name() -> String { return "Firebase Chat" }
    init() {
        FIRApp.configure()
        ref = FIRDatabase.database().reference()

        let textsRefs = ref.child(CHANNEL_NAME)

        textsRefs.observe(.childAdded, with: { (snapshot) -> Void in
            if let info = snapshot.value as? [String: Any],
                let entry = ChatUtils.from(dictionary: info) {
                self.messagesSubject.onNext(entry)
            }
        })
    }

    func send(message: String){
        self.ref.child(CHANNEL_NAME)
            .childByAutoId()
            .setValue( ChatUtils.createDictionaryData(for: message))
    }

    func receive() -> Observable<TextEntry>{
        return messagesSubject.asObservable()
    }

    func color() -> String {
```

```

    return "#ffc929"
}
}

```

5.1.3.3. PUBNUB

DEFINICIÓN

Plataforma que provee APIs de mensajería instantánea para móviles, web, internet de las cosas, computadoras de escritorio y servidores, permitiendo establecer mensajería bidireccional de baja latencia en cualquier lugar del mundo.

CARACTERÍSTICAS

Tabla 5 Características de Pubnub

	Android	iOS
IDE de Desarrollo	Android Studio 2.3 o superior	Xcode 8.3 o superior
Versión de la librería	4.6.2	4.5.13
Lenguajes de programación compatibles	Java Kotlin	Swift Objective-C
Versiones de Android soportadas	API 14 o superior	iOS 8

DIAGRAMAS

Figura 19 Diagrama de Componentes - Pubnub

Figura 20 Diagrama de Despliegue - Pubnub

Figura 21 Diagrama de Secuencia - Pubnub

IMPLEMENTACIÓN

Android

1. En el archivo build.gradle del módulo **app**, ir a la sección dependencies, y añadir la dependencia al proyecto:

```
compile group: 'com.pubnub', name: 'pubnub-gson', version: '4.6.2'
```

2. Crear un cliente Pubnub, indicando las llaves de subscripción y publicación.

```
val pnConfiguration = PNConfiguration()
pnConfiguration.setSubscribeKey("YOUR_SUBSCRIBE_KEY")
pnConfiguration.setPublishKey("YOUR_PUBLISHER_KEY")

val pubnub = PubNub(pnConfiguration)
```

3. Suscribirse al canal, para poder realizar las operaciones que deseemos con el mensaje que recibamos

```
pubnub.addListener(object : SubscribeCallback() {
    override fun status(pubnub: PubNub, status: PNStatus) {
        ...
    }

    override fun message(pubnub: PubNub, result: PNMessageResult) {
        // Handle new message stored in message.message
        var data : String?
        try{
            data = result.message.asString
        } catch (e: Exception){
            data = result.message.toString().replace("\\\\", "\\")
        }
    }

    override fun presence(pubnub: PubNub,
                          presence: PNPresenceEventResult) {

    }
})

pubnub.subscribe()
    .channels(Arrays.asList("your_channel_name"))
    .execute()
```

4. Para enviar mensajes, invocar el método *publish* sobre el cliente:

```
pubnub.publish().channel("chat")
    .message(ChatUtils.createData(message))
    .async(object : PNCallback<PNPublishResult>() {
        override fun onResponse(result: PNPublishResult?,
                                status: PNStatus?) {

            //do something with the data
        }
    })
```

En este proyecto, realizamos dicha implementación siguiendo el patrón Repository, invocando los métodos de subscripción / publicación en los lugares correspondientes. El resultado final es la clase PubnubChatRepository

```
class PubNubRepository() : ChatRepository {
    override val name: String get() = "PubNub Chat"
    val pubnub : PubNub
    private val messageSubject= PublishSubject<Message>.create()

    init{
        val pnConfiguration = PNConfiguration()
        pnConfiguration.setSubscribeKey("key-sub")
        pnConfiguration.setPublishKey("key-pub")

        pubnub = PubNub(pnConfiguration)

        pubnub.addListener(object : SubscribeCallback() {
            override fun status(pubnub: PubNub, status: PNStatus) { }

            override fun message(pubnub: PubNub, result: PNMessageResult) {
                // Handle new message stored in message.message
                var data : String?
                try{
                    data = result.message.asString
                }catch (e: Exception){
                    data = result.message.toString().replace("\\", "")
                }

                if(data != null){
                    val message = ChatUtils.parseMessage(data)
                    if (message != null){
                        messageSubject.onNext(message)
                    }
                }
            }
        })

        override fun presence(pubnub: PubNub,
                               presence: PNPresenceEventResult) { }

        pubnub.subscribe().channels(Arrays.asList("chat")).execute()
    }

    override fun send(message: String) {
        pubnub.publish().channel("chat")
            .message(ChatUtils.createData(message))
            .async(object : PNCallback<PNPublishResult>() {
                override fun onResponse(result: PNPublishResult?,
                                         status: PNStatus?) {
                    //do nothing
                }
            })
    }

    override fun receive(): Flowable<Message> {
        return messageSubject.toFlowable(BackpressureStrategy.BUFFER)
    }
}
```

iOS

1. En el archivo Podfile, agregamos la dependencia de Pubnub

```
pod 'Pubnub'
```

2. Configurar Pubnub usando las llaves de publicación (para enviar mensajes) y suscripción (para recibir mensajes) que ellos te proveen, y el nombre del canal por el cual transmitirás los mensajes.

```
let configuration = PNConfiguration(publishKey: "PUBLISHER_KEY",
                                   subscribeKey: "SUBSCRIBE_KEY")
self.client = PubNub.clientWithConfiguration(configuration)
```

3. Suscribirse a los cambios el canal, para poder realizar las operaciones que deseemos con el mensaje que recibamos:

```
self.client?.addListener(self)
self.client?.subscribeToChannels([CHANNEL_NAME], withPresence: true)

func client(_ client: PubNub, didReceiveMessage message: PNMessageResult) {
    let data = message.data.message as? [String:Any]
    //do something with the data
}
```

4. Para enviar mensajes, invocar el método *setValue* sobre la referencia a la base de datos:

```
self.client?.publish("your message",
                    toChannel: CHANNEL_NAME,
                    compressed: false,
                    withCompletion: { (status) in
                        //do nothing
                    })
```

En este proyecto, realizamos dicha implementación siguiendo el patrón Repository, invocando los métodos de suscripción / publicación en los lugares correspondientes. El resultado final es la clase *PubnubChatRepository*

```
class PubnubChatRepository : NSObject, ChatRepository, PNOBJECTEventListener {

    private let CHANNEL_NAME = "chat"
    var client: PubNub?
    private var messagesSubject = PublishSubject<TextEntry>()

    func name() -> String { return "Pubnub Chat" }
    override init() {
        super.init()

        // Initialize and configure PubNub client instance
        let configuration = PNConfiguration(publishKey: "pub-c-8aa801f5-05d1-42bd-9c6f-6ee9e829f37f",
                                           subscribeKey: "sub-c-010f45da-2d2d-11e7-a696-0619f8945a4f")
        self.client = PubNub.clientWithConfiguration(configuration)
        self.client?.addListener(self)

        // Subscribe to demo channel with presence observation
        self.client?.subscribeToChannels([CHANNEL_NAME], withPresence: true)
    }

    func send(message: String){
```

```

        self.client?.publish(
            ChatUtils.createData(for: message),
            toChannel: CHANNEL_NAME,
            compressed: false,
            withCompletion: { (status) in
                //do nothing
            })
    }

    func receive() -> Observable<TextEntry>{
        return messagesSubject.asObservable()
    }

    // Handle new message from one of channels on which client has been subscribed.
    func client(_ client: PubNub, didReceiveMessage message: PNMessageResult) {
        var entry: TextEntry?
        if let data = message.data.message as? [String:Any] {
            entry = ChatUtils.from(dictionary: data)
        } else if let dataString = message.data.message as? String {
            entry = ChatUtils.from(data: dataString)
        }
        if entry != nil {
            messagesSubject.onNext(entry!)
        }
    }

    func color() -> String {
        return "#cf2127"
    }
}

```


5.1.3.4. RABBITMQ

DEFINICIÓN

Es una tecnología que puede ser desplegada sobre cualquier servidor Linux / Windows. Soporta múltiples protocolos de mensajería del lado servidor, los cuales pueden ser consumidos por clientes de escritorio, móvil o internet de las cosas por medio de sus librerías.

CARACTERÍSTICAS

Tabla 6 Características de RabbitMQ

	Android	iOS
IDE de Desarrollo	Android Studio 2.3 o superior	Xcode 8.3 o superior
Versión de la librería	4.1.1	0.10.0
Lenguajes de programación compatibles	Java Kotlin	Swift Objective-C
Versiones de Android soportadas	API 14 o superior	iOS 8

DIAGRAMAS

Figura 22 Diagrama de Componentes - RabbitMQ

Figura 23 Diagrama de Secuencia - RabbitMQ

IMPLEMENTACIÓN

Android

1. En el archivo build.gradle del módulo app, ir a la sección dependencies, y añadir la dependencia al proyecto:

```
compile 'com.rabbitmq:amqp-client:4.1.1'
```

2. Crear un cliente RabbitMQ, indicando la IP, puerto, y credenciales para conectarse al servidor.
3. Suscribirse al canal, para poder realizar las operaciones que deseemos con el mensaje que recibamos

```
init { subscribe(handler) }

val handler = object : Handler() {
    override fun handleMessage(msg: android.os.Message) {
        val message = msg.data.getString("msg")
        //handle the message as you wish
    }
}

fun subscribe(handler: Handler) {
    subscribeThread = Thread(Runnable {
        try{
            connection = factory.newConnection()
            val channel = connection.createChannel()

            channel.exchangeDeclare(EXCHANGE_NAME, "fanout")
            val queueName = channel.queueDeclare().queue
            channel.queueBind(queueName, EXCHANGE_NAME, "")

            val consumer = object : DefaultConsumer(channel) {
                @Throws(IOException::class)
                override fun handleDelivery(consumerTag: String, envelope: Envelope,
                    properties: AMQP.BasicProperties, body: ByteArray) {
                    val message = String(body)
                    println("[x] Received '$message'")
                    val msg = handler.obtainMessage()
                    val bundle = Bundle()
                    bundle.putString("msg", message)
                    msg.data = bundle
                    handler.sendMessage(msg)
                }
            }

            channel.basicConsume(queueName, true, consumer)
        } catch (e: Exception) {
            Handler().postDelayed({
                subscribe(handler)
            }, 5000)
        }
    })
    subscribeThread.start()
}
```

4. Para enviar mensajes, invocar el método publish sobre el objeto cliente:

```
private fun publishMessage(message: String){
    val channel = connection.createChannel()
```

```

channel.exchangeDeclare(EXCHANGE_NAME, "fanout")

channel.basicPublish(EXCHANGE_NAME, "", null, message.toByteArray())
println(" [x] Sent '$message'")

channel.close()

try {
    Log.d("RabbitMQ", "[q] " + message);
    queue.putLast(message);
} catch (e: Exception) {

}
}

```

En este proyecto, realizamos dicha implementación siguiendo el patrón Repository, invocando los métodos de subscripción / publicación en los lugares correspondientes. El resultado final es la clase RabbitMQRepository

```

class RabbitMQRepository() : ChatRepository{
    override val name: String get() = "RabbitMQ"

    private val HOST = "10.11.80.94"
    private val PORT = 5673
    private val URI = "amqp://$HOST:$PORT"

    private val EXCHANGE_NAME = "chat"

    private lateinit var connection : Connection

    val factory = ConnectionFactory().apply {
        isAutomaticRecoveryEnabled = false
        host = HOST
        port = PORT
        username = "mobile"
        password = "1234"
        setUri(URI)
    }

    val handler = object : Handler() {

        override fun handleMessage(msg: android.os.Message) {
            val message = msg.data.getString("msg")
            ChatUtils.parseMessage(message)?.let {
                messageSubject.onNext(it)
            }
        }
    }

    init {
        subscribe(handler)
    }

    private val messageSubject: PublishSubject<Message> = PublishSubject.create()
    override fun receive(): Flowable<Message> {
        return messageSubject.toFlowable(BackpressureStrategy.BUFFER)
    }

    val queue = LinkedBlockingDeque<String>()
    override fun send(message: String) {
        publishMessage(ChatUtils.createData(message))
    }

    //region //Publish
    private fun publishMessage(message: String){

```

```

    val channel = connection.createChannel()

    channel.exchangeDeclare(EXCHANGE_NAME, "fanout")

    channel.basicPublish(EXCHANGE_NAME, "", null, message.toByteArray())
    println(" [x] Sent '$message'")

    channel.close()

    try {
        Log.d("RabbitMQ", "[q] " + message);
        queue.putLast(message);
    } catch (e: Exception) {

    }
}

//endregion

//region //Subscribe

private lateinit var subscribeThread: Thread

fun subscribe(handler: Handler) {
    subscribeThread = Thread(Runnable {
        try {
            connection = factory.newConnection()
            val channel = connection.createChannel()

            channel.exchangeDeclare(EXCHANGE_NAME, "fanout")
            val queueName = channel.queueDeclare().queue
            channel.queueBind(queueName, EXCHANGE_NAME, "")

            val consumer = object : DefaultConsumer(channel) {
                @Throws(IOException::class)
                override fun handleDelivery(consumerTag: String, envelope: Envelope,
                    properties: AMQP.BasicProperties, body: ByteArray) {
                    val message = String(body)
                    println(" [x] Received '$message'")
                    val msg = handler.obtainMessage()
                    val bundle = Bundle()
                    bundle.putString("msg", message)
                    msg.data = bundle
                    handler.sendMessage(msg)
                }
            }

            channel.basicConsume(queueName, true, consumer)
        } catch (e: Exception) {
            Handler().postDelayed({
                subscribe(handler)
            }, 5000)
        }
    })
    subscribeThread.start()
}

//endregion
}

```

iOS

1. En el archivo Podfile, agregamos la dependencia de Pubnub

```
pod 'RMQClient'
```

2. Crear un cliente RabbitMQ, indicando la IP, puerto, y credenciales para conectarse al servidor.

```

private let URI = "amqp://mobile:1234@YOUR_IP:YOUR_PORT"
private let USERNAME = "username"
private let PASS = "password"

let conn : RMQConnection

func name() -> String { return "RabbitMQ Chat" }
init() {
    conn = RMQConnection(uri: URI, delegate: RMQConnectionDelegateLogger())
    conn.start()
}

```

3. Suscribirse a los cambios el canal, para poder realizar las operaciones que deseemos con el mensaje que recibamos:

```

private func subscribe(){
    DispatchQueue.global(qos: .background).async {

        let ch = self.conn.createChannel()

        ch.exchangeDeclare("chat", type: "fanout")

        let name = ch.queue("").name!
        ch.queueBind(name, exchange: "chat", routingKey: "")

        let consumer = RMQConsumer(channel: ch, queueName: name, options: .exclusive)
        consumer?.onDelivery({ (message) in
            if let content = String(data: message.body, encoding: .utf8){
                print("Received \(content)")

                DispatchQueue.main.async {
                    if let entry = ChatUtils.from(data: content) {
                        self.messagesSubject.onNext(entry)
                    }
                }
            }
        })

        ch.basicConsume(consumer!)

    }
}

```

4. Para enviar mensajes, invocar el método *basicPublish* sobre el canal:

```

DispatchQueue.global(qos: .background).async {
    let ch = self.conn.createChannel()

    ch.exchangeDeclare("chat", type: "fanout")

    ch.basicPublish("your message",
        routingKey: "",
        exchange: "chat",
        properties: [] )
}

```

```

        ch.blockingWait(on: type(of: self))
    }

```

En este proyecto, realizamos dicha implementación siguiendo el patrón Repository, invocando los métodos de subscripción / publicación en los lugares correspondientes. El resultado final es la clase *RabbitMQChatRepository*

```

class RabbitMQChatRepository : ChatRepository{

    private let URI = "amqp://mobile:1234@10.11.80.80:5673"
    private let USERNAME = "mobile"
    private let PASS = "1234"
    let conn : RMQConnection

    func name() -> String { return "RabbitMQ Chat" }
    init() {
        conn = RMQConnection(uri: URI, delegate: RMQConnectionDelegateLogger())
        conn.start()
        subscribe()
    }
    func color() -> String { return "#ff6600" }

    func send(message: String){
        DispatchQueue.global(qos: .background).async {
            let ch = self.conn.createChannel()

            ch.exchangeDeclare("chat", type: "fanout")

            if let data = ChatUtils.createData(for: message).data(using: .utf8){
                ch.basicPublish(data,
                                routingKey: "",
                                exchange: "chat",
                                properties: [] )
                ch.blockingWait(on: type(of: self))
            }
        }
    }

    var messagesSubject = PublishSubject<TextEntry>()
    func receive() -> Observable<TextEntry>{
        return messagesSubject.asObservable()
    }
    private func subscribe(){

        DispatchQueue.global(qos: .background).async {
            let ch = self.conn.createChannel()
            ch.exchangeDeclare("chat", type: "fanout")
            let name = ch.queue("").name!
            ch.queueBind(name, exchange: "chat", routingKey: "")
            let consumer = RMQConsumer(channel: ch, queueName: name, options: .exclusive)
            consumer?.onDelivery({ (message) in
                if let content = String(data: message.body, encoding: .utf8){
                    DispatchQueue.main.async {
                        if let entry = ChatUtils.from(data: content) {
                            self.messagesSubject.onNext(entry)
                        }
                    }
                }
            })

            ch.basicConsume(consumer!)
        }
    }
}

```

5.1.3.5. LIGHSTREAMER

DEFINICIÓN

Es una implementación para servidores que permite múltiples formas de comunicación en tiempo real. Es lo suficientemente flexible para ser utilizada en múltiples escenarios.

CARACTERÍSTICAS

Tabla 7 Características de Lighstreamer

	Android	iOS
IDE de Desarrollo	Android Studio 2.3 o superior	Xcode 8.3 o superior
Versión de la librería	3.0.0	2.1.2
Lenguajes de programación compatibles	Java Kotlin	Swift Objective-C
Versiones de Android soportadas	API 14 o superior	iOS 8

DIAGRAMAS

Figura 24 Diagrama de Componentes - Lighstreamer

Figura 25 Diagrama de Despliegue - Lighstreamer

Figura 26 Diagrama de Secuencia - Lighstreamer

IMPLEMENTACIÓN

Android

1. En el archivo build.gradle del módulo app, ir a la sección dependencies, y añadir la dependencia al proyecto:

```
compile 'com.lightstreamer:ls-android-client:3.0.0'
```

2. Crear un cliente Lighstreamer, indicando el modo, el item, el adapter y los elementos que se transmitirán.

```
val serverAddress = "http://YOUR_SERVER_IP:YOUR_SERVER_PORT"
val subscription = Subscription("DISTINCT","chat_room", arrayOf("your","variables")).apply {
    addListener(this@LightstreamerRepository)
    dataAdapter = "CHAT_ROOM"
    requestedSnapshot = "yes"
}
val lsClient = LightstreamerClient(serverAddress,"CHAT").apply {
    connectionDetails.serverAddress = serverAddress
    connect()
    subscribe(subscription)
    addListener(this@LightstreamerRepository)
}
```

3. Opcionalmente, implementar la interfaz *ClientListener* para detectar eventos relacionados a la conexión.
4. Suscribirse al canal implementando la interfaz *SubscriptionListener*, para poder realizar las operaciones que deseemos con el mensaje que recibamos

```
override fun onItemUpdate(p0: ItemUpdate?) {
    try {
        if(p0 != null && p0.fields != null){
            val message = p0.fields["message"]
            //handle the message as you wish
        }else{
            print("Algo es nulo")
        }
    }
    catch (e: Exception){
        print("Exc: $e")
    }
}
```

5. Para enviar mensajes, invocar el método sendMessage sobre el objeto cliente (indicar el nombre del chat):

```
lsClient.sendMessage("CHAT| your message")
```

En este proyecto, realizamos dicha implementación siguiendo el patrón Repository, invocando los métodos de subscripción / publicación en los lugares correspondientes. El resultado final es la clase *LightstreamerChatRepository*

```
class LightstreamerRepository : ChatRepository,
    SubscriptionListener by LightstreamerListenerLogger,
    ClientListener by LightstreamerListenerLogger{
```

```

override val name: String get() = "Demo Lightstreamer"

val serverAddress = "http://10.11.80.88:8080"
val subscription = Subscription("DISTINCT", "chat_room", arrayOf("message", "raw_timestamp", "IP", "nick")).apply {
    addListener(this@LightstreamerRepository)
    dataAdapter = "CHAT_ROOM"
    requestedSnapshot = "yes"
}
val lsClient = LightstreamerClient(serverAddress, "CHAT").apply {
    connectionDetails.serverAddress = serverAddress
    connect()
    subscribe(subscription)
    addListener(this@LightstreamerRepository)
}

private val messageSubject: PublishSubject<Message> = PublishSubject.create()

override fun send(message: String) {
    ChatUtils.createData(message).let {
        lsClient.sendMessage("CHAT| $it")
    }
}

override fun receive(): Flowable<Message> {
    return messageSubject.toFlowable(BackpressureStrategy.BUFFER)
}

override fun onItemUpdate(p0: ItemUpdate?) {
    try {
        if(p0 != null && p0.fields != null){
            val message = p0.fields["message"]
            if (message != null){
                val entry = ChatUtils.parseMessage(message)
                messageSubject.onNext(entry)
            }else{
                print("Message es null")
            }
        }else{
            print("Algo es nulo")
        }
    }

    /*p0?.fields?.get("message")?.let{
        val message = ChatUtils.parseMessage(it)
        messageSubject.onNext(message)
    }*/

    }catch (e: Exception){
        print("Exc: $e")
    }
}

}

object LightstreamerListenerLogger : SubscriptionListener, ClientListener{
    //implementing all the methods to log each event...
}

```

iOS

1. En el archivo Podfile, agregamos la dependencia de Pubnub


```
pod 'Lightstreamer_iOS_Client'
```
2. Crear un cliente Lighstreamer, indicando el modo, el item, el adpter y los elementos que se transmitirán.

```

let SERVER_URL = "http://10.11.80.88:8080"
let ADAPTER_SET = "CHAT"
let DATA_ADAPTER = "CHAT_ROOM"

var _client : LSLightstreamerClient!
var _subscription : LSSubscription!

```

```
required override init() {

    super.init()

    _subscription = LSSubscription(subscriptionMode: "DISTINCT", item: "chat_room", fields:
["message", "raw_timestamp", "IP"])
    _subscription.dataAdapter = DATA_ADAPTER
    _subscription.requestedSnapshot = "yes"

    _client = LSLightstreamerClient(serverAddress: SERVER_URL, adapterSet:
ADAPTER_SET)
    _client.connectionDetails.serverAddress = SERVER_URL

}
```

3. Suscribirse a los cambios el canal, implementando la *LSClientDelegate* y *LSSubscriptionDelegate*:

```
extension LightStreamerChatRepository : LSClientDelegate, LSSubscriptionDelegate {
    func subscription(_ subscription: LSSubscription, didClearSnapshotForItemName itemName:
String?, itemPos: UInt) {
    }

    func subscription(_ subscription: LSSubscription, didEndSnapshotForItemName itemName:
String?, itemPos: UInt) {
        NSLog("LS Subscription snapshot did end")
    }

    func subscription(_ subscription: LSSubscription, didUpdateItem itemUpdate: LSItemUpdate)
    {
        let message = itemUpdate.value(withFieldName: "message") ?? ""
        //do something with the data
    }
}
```

4. Para enviar mensajes, invocar el método *basicPublish* sobre el canal:

```
self._client.sendMessage("CHAT|" + "your message")
```

En este proyecto, realizamos dicha implementación siguiendo el patrón Repository, invocando los métodos de subscripción / publicación en los lugares correspondientes. El resultado final es la clase *LightStreamerChatRepository*

```
class LightStreamerChatRepository: NSObject, ChatRepository {

    // Configuration for local installation
    let SERVER_URL = "http://10.11.80.88:8080"
    let ADAPTER_SET = "CHAT"
    let DATA_ADAPTER = "CHAT_ROOM"

    var _client : LSLightstreamerClient!
    var _subscription : LSSubscription!
    var messagesSubject = PublishSubject<TextEntry>()

    func name() -> String { return "LightStreamer Chat" }
```

```

required override init() {

    super.init()

    _subscription = LSSubscription(subscriptionMode: "DISTINCT", item: "chat_room", fields:
["message", "raw_timestamp", "IP"])
    _subscription.dataAdapter = DATA_ADAPTER
    _subscription.requestedSnapshot = "yes"
    _subscription.addDelegate(self)

    _client = LSLightstreamerClient(serverAddress: SERVER_URL, adapterSet: ADAPTER_SET)
    _client.connectionDetails.serverAddress = SERVER_URL

    _client.addDelegate(self)
    _client.subscribe(_subscription)
    _client.connect()

}

func send(message: String){
    self._client.sendMessage("CHAT|" + ChatUtils.createData(for: message))
}

func receive() -> Observable<TextEntry>{
    return messagesSubject.asObservable()
}

func color() -> String {
    return "#689968"
}

}

extension LightStreamerChatRepository : LSClientDelegate, LSSubscriptionDelegate{
    func subscription(_ subscription: LSSubscription, didClearSnapshotForItemName itemName: String?,
itemPos: UInt) {
    }

    func subscription(_ subscription: LSSubscription, didEndSnapshotForItemName itemName: String?,
itemPos: UInt) {
        NSLog("LS Subscription snapshot did end")
    }

    func subscription(_ subscription: LSSubscription, didUpdateItem itemUpdate: LSItemUpdate) {
        let message = itemUpdate.value(withFieldName: "message") ?? ""
        if let entry = ChatUtils.from(data: message){
            self.messagesSubject.onNext(entry)
        }
    }
}

```

5.1.3.6. XAMARIN

DEFINICIÓN

- Xamarin: plataforma que nos permite desarrollar aplicaciones nativas para Android, iOS, y Windows Phone, con la ventaja de poder desarrollar para todas esas plataformas en un solo proyecto, con una única fuente de código (escrito en C#)
 - Xamarin.Forms: enfoque que te permite desarrollar las interfaces visuales usando un único código fuente. Xamarin se encarga de generar las interfaces correspondientes para cada plataforma automáticamente.
 - Xamarin.iOS & Xamarin.Android: enfoque en el cual puedes reutilizar la lógica de negocio del proyecto, pero el desarrollo de la interfaz se hace por separado. Te da un poco de flexibilidad al momento de diseñar, pero implica más trabajo. Este es el enfoque que utilizamos en la aplicación.
- SignalR: "es una librería para desarrolladores ASP.NET, que hace increíblemente simple añadir funcionalidad en tiempo real a tus aplicaciones"

CARACTERÍSTICAS

Tabla 8 Características de Xamarin

IDE de Desarrollo	Xamarin Studio Community 6.3
Versión de la librería	2.1.2.2
Lenguajes de programación compatibles	C#
Versiones de Android soportadas	Android API 11 o superior iOS 7 o superior

DIAGRAMAS

Figura 27 Diagrama de Componentes - Xamarin

Figura 28 Diagrama de Despliegue - Xamarin

Figura 29 Diagrama de Secuencia - Xamarin

IMPLEMENTACIÓN

Shared Project

En el proyecto compartido, crearemos una clase `SignalConnector`, la cual será el punto central por el cual las aplicaciones Android / iOS podrán conectarse con el servidor web.

Al momento de crear una instancia de esta clase, estableceremos una conexión con el servidor (que también utiliza SignalR). Para este caso, haremos referencia al Hub llamado `ChatHub`. Luego, nos suscribiremos al evento *`addNewMessageToPage`*, el cual nos devolverá el mensaje enviado y el timestamp de cuando este fue enviado.

```
HubConnection chatConnection;
IHubProxy SignalRChatHub;

public SignalConnector(Action<Response> listener)
{
    chatConnection = new HubConnection("http://chatsignalrtst.apphb.com/",true);
    SignalRChatHub = chatConnection.CreateHubProxy("ChatHub");

    SignalRChatHub.On<string, string>("addNewMessageToPage", (timestamp,message) =>
    {
        var response = new Response() { Size = MeasureSize(message), Delay = MeasureDelay(timestamp), Message =
message };
        listener(response);
    });
}

public async virtual void JoinChat()
{
    try
    {
        await chatConnection.Start();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Cuando el usuario envíe un mensaje desde la interfaz de la aplicación, invocará el método *`send(string message)`*. En ese momento, enviaremos el mensaje *`Send`* al hub que registramos previamente.

```
public async void Send(string message)
{
    if (chatConnection.State == ConnectionState.Connected)
    {
        var timestamp = CreateTimestamp();
        await SignalRChatHub.Invoke("Send", timestamp, message);
    }
}
```



```
private static readonly string DateFormat = "yyyy-MM-dd hh:mm:ss.ffffff";
private string CreateTimestamp()
{
    return DateTime.Now.ToString(DateFormat);
}
```

Xamarin.Android

1. Para empezar, añadiremos SignalR al proyecto por medio de NuGet. Tenemos que dar click derecho en el proyecto > Agregar > Agregar paquete NuGet
2. Luego, abriremos la clase de MainActivity.cs y en el método OnCreate instanciaremos la clase SignalConnector

```
Connector = new SignalConnector((Response response) => RunOnUiThread(() =>
{
    //Actualiza la lista de mensajes
    adapter.Add(response);
}));
```

```
Connector.JoinChat();
```

3. Asimismo, cuando el usuario presione el botón "Enviar" invocaremos al método Send del conector

```
Button button = FindViewById<Button>(Resource.Id.ChatButton);
button.Click += delegate
{
    if (string.IsNullOrEmpty(input.Text))
        return;

    Connector.Send(input.Text);
    input.Text = "";
};
```

4. Implementamos la clase [ResponseAdapter](#), la cual únicamente se encargará de mostrar las respuestas en la vista.

Xamarin.iOS

1. Añadimos el paquete de SignalR de la misma forma que en Android.
2. Luego, abriremos la clase de ViewController.cs y en el método ViewDidLoad instanciaremos la clase SignalConnector

```
Connector = new SignalConnector((Response response) => RunOnUiThread(() =>
{
    //Actualiza la lista de mensajes
    source.Add(response);
    TableView.ReloadData();
}));
```

```
Connector.JoinChat();
```

3. Asimismo, cuando el usuario presione el botón "Enviar" invocaremos al método Send del conector

```
SendButton.TouchDown += (sender, e) =>
```

```
{  
    if (string.IsNullOrEmpty(MessageTextField.Text))  
        return;  
  
    Connector.Send(MessageTextField.Text);  
    MessageTextField.Text = "";  
    MessageTextField.ResignFirstResponder();  
};
```

4. Implementamos las clases [MessagesDataSource](#) y [ResponseEntryCell](#), las cuales únicamente se encargarán de mostrar los mensajes en la vista.

5.2 DESARROLLO DE MEDICIONES

Para el desarrollo de mediciones se definieron diferentes escenarios de pruebas según las tecnologías utilizadas, y se aplicó la medición para cada una de ellas como se describe en esta sección.

5.2.1 ENTORNO DE DESARROLLO DE PRUEBAS

LUGAR

Elegimos una habitación cerrada donde se pueda hacer las pruebas, el espacio central se encontraba vacío para las pruebas de distancia. La ubicación fue en pueblo libre, distrito en la cual se cuenta con señal 4g estable y fuera de una calle transitada para la existencia de ruido. Se contaba con acceso a internet por medio de WiFi, el router se encontraba al otro lado de la habitación con señal optima y solo se mantenían conectados (acceso a internet) los dispositivos que se encontraban bajo prueba.

DISPOSITIVOS

Se utilizaron 6 dispositivos para esta prueba, los cuales están listados a continuación.

Tabla 9 Dispositivos utilizados durante las pruebas

Dispositivo	Tipo	SO	Función
Computadora	Laptop Toshiba	Windows 10	Servidor
	MacBook Pro	Mac OS	Beacon
Celular	Samsung Galaxy J5	Android	Emisor - Receptor
	Nexus	Android	Emisor – Receptor
	iPhone s5	iOS	Emisor – Receptor
	iPhone s6	iOS	Emisor – Receptor

Para cada tecnología, se utilizó una combinación entre estos dispositivos los cuales se detallarán en los escenarios de prueba.

COMPILADOS A PROBAR

Las tecnologías por probar eran 11, con un total de 26 compilados: 11 Aplicativos Android, 11 Aplicativos iOS, 4 servicios y 2 beacons.

La descripción de lo necesario para cada tecnología se muestra a continuación:

Tabla 10 Aplicaciones requeridas por tecnología

Tecnología	Aplicativo Android	Aplicativo iOS	Servidor	Beacon
Ably	X	X		
FireBase	X	X		
Pub/Nub	X	X		
RabbitMQ	X	X	X	
LightStreamer	X	X	X	
SignalR	X	X	X	
Eddystone	X	X		X
Nearby	X	X		X
Peer.JS	X	X	X	
RTM	X	X		X
iBeacons	X	X		X

ESCENARIOS DE PRUEBA

Para cada escenario de prueba se probó el envío de información de manera bidireccional, por lo que las mediciones se hicieron 5 veces para tener su totalidad: de Android – Android, iOS – iOS, iOS – Android y Android – iOS.

Si bien generamos un entorno similar para cada mecanismo de comunicación, los separamos en 3 grupos ya que, según sus especificaciones y accesos, no podían ser probados bajo el mismo escenario y dimensión.

- Escenario 1: A grupo pertenecen a las tecnologías de comunicación que utilizan un servidor externo cloud que se encarga de realizar la distribución de la información (consideramos a SignalR dentro de este grupo ya que su infraestructura esa diseñada para ser usado de forma web/cloud).
- Escenario 2: A grupo pertenecen las tecnologías que necesitan la implementación de un servidor para su demostración. Se probaron en un entorno de red, ya que es la ventaja que poseen al ser implementados directamente.
- Escenario 3: A este grupo pertenecen todas las tecnologías que hacen uso del bluetooth tanto para el descubrimiento o el envío de información.

Tabla 11 Escenarios de prueba

Escenario 1	Tecnologías	Servidor cloud	Ably Firebase Pub/Nub SignalR RTM
	Dimensiones	Acceso a Red	Wifi 3G/4g
		Dirección	Android – Android iOS – iOS iOS – Android Android – iOS
Escenario 2	Tecnologías	Servidor red	Peer.js RabbitMQ LightStreamer
	Dimensiones	Acceso a Red	Wifi
		Dirección	Android – Android iOS – iOS iOS – Android Android – iOS
Escenario 3	Tecnologías	Beacons	Eddystone Nearby iBeacons
	Dimensiones	Acceso a Red	WiFi 3G/\$G
		Dirección	Android – Android iOS – iOS iOS – Android Android – iOS
		Distancia	Corta (0,50 m.) Media (1,00 m.) Larga (2.50 m.)

5.2.2 SINCRONIZACION DE TIEMPO REAL Y TIEMPOS DE ENVIO

Un problema que se encontró durante el desarrollo del proyecto fue determinar el tiempo real y sincronizar los dispositivos a este para lograr el cálculo de tiempo de envío. Al ser tecnologías que permiten la comunicación de manera muy rápida (transmisión variante menor a 3 segundos aproximadamente) la forma en la cual podíamos comparar los tiempos de envío era comparándolo en milisegundos. Si bien los dispositivos se sincronizan de manera automática a un servidor externo, estos no cuentan con la precisión a milisegundos, por lo que se genera una dificultad al análisis.

CALCULO DEL TIEMPO REAL

Definimos tiempo real como el tiempo que transcurre de forma constante, por lo que si tuviéramos dos relojes que se encuentran con horas diferentes, ambos existen bajo un tiempo real a pesar de sus desfases. Entonces, bajo este contexto, existe un tiempo real, y sobre este los diferentes dispositivos poseen un desfase que puede ser positivo o negativo, dependiendo del tiempo local que presente el dispositivo. Para ello definimos la siguiente fórmula:

$$T_{REAL} = t_{LOCAL} + dt_{LOCAL}$$

Para poder calibrar los dispositivos a este tiempo, es necesario generar el cálculo a partir de esta fórmula. Por ello, es necesario al menos dos de estas variables para poder generar el cálculo, teniendo en cuenta que la precisión necesaria es en milisegundos.

CALCULO DEL TIEMPO DE ENVIO

Las mediciones comparan el tiempo de envío para cada tecnología, por lo que esta se determina a partir de dos tiempos, un tiempo inicial que es el tiempo de emisión y un tiempo final que es el tiempo de recepción. Si los comparamos con el tiempo real tendríamos las siguientes fórmulas.

$$T_E = T_{recepción} - T_{emisión}$$

$$T_{E_{REAL}} = T_{REAL_1} - T_{REAL_0}$$

Si adicionamos la dimensión de los tiempos de cada dispositivo, tendríamos la siguiente fórmula:

$$T_{emisión} = T_{REAL_0} = T_{DISP_EMI_0} + dt_{DISP_EMI}$$

$$T_{recepción} = T_{REAL_1} = T_{DISP_RCP_1} + dt_{DISP_RCP}$$

Según esto, la formula final sería la siguiente:

$$T_{E_{REAL}} = T_{REAL_1} - T_{REAL_0} = T_{DISP_RCP_1} + dt_{DISP_RCP} - T_{DISP_EMI_0} + dt_{DISP_EMI}$$

Sin embargo, la información que se recopila a través de los celulares no toma el tiempo real en si, sino se adquiere un tiempo de envío relativo que contiene los efectos de los desfases, por lo que el escenario que se genera es el siguiente:

$$T_{E_{RELATIVO}} = T_{DISP_RCP_1} - T_{DISP_EMI_0}$$

Si se reemplaza con las fórmulas de tiempo real, se puede derivar los tiempos con respecto a los desfases y hallar el tiempo de envío real.

$$T_{DISP_EMI_0} = T_{REAL_0} - dt_{DISP_EMI}$$

$$T_{DISP_RCP_1} = T_{REAL_1} - dt_{DISP_RCP}$$

$$T_{E_{RELATIVO}} = T_{REAL_1} - dt_{DISP_RCP} - (T_{REAL_0} - dt_{DISP_EMI})$$

$$T_{REAL_1} - T_{REAL_0} = T_{E_{RELATIVO}} + dt_{DISP_RCP} - dt_{DISP_EMI}$$

$$T_{E_{REAL}} = T_{E_{RELATIVO}} + dt_{DISP_RCP} - dt_{DISP_EMI}$$

6. CAPÍTULO 6: RESULTADOS DEL PROYECTO

6.1. Resultados de las mediciones

Una vez listo el ambiente de pruebas descrito en el capítulo anterior, se procedió a realizar la toma de mediciones para cada uno de los escenarios de prueba planteados. En total, se registraron 6200 envíos de información entre los dispositivos móviles. Se realizaron 50 mediciones bajo cada combinación de cada dimensión, para cada escenario de prueba, para cada tecnología.

Una vez registradas las mediciones, se realizó el ajuste de los valores utilizando las formulas y el desfase de tiempo de cada dispositivo de prueba, tal como se describió en la sección anterior. Los resultados a continuación muestran los valores luego de realizar el ajuste correspondiente.

Figura 30 Tiempo de Envío en Milisegundos de Tecnologías basadas en comunicación por la nube

Figura 31 Tiempo de Envío en Milisegundos de Tecnologías basadas en comunicación en una red local

Figura 32 Tiempo de Envío en Milisegundos de Tecnologías basadas en Bluetooth

6.2. Validación de los resultados

La información recopilada tiene como finalidad ayudar a los desarrolladores de aplicaciones móviles a tomar una decisión informada con respecto a qué tecnología de comunicación deberían utilizar de acuerdo a las necesidades de su proyecto. Por ello, se llevó a cabo una encuesta a profesionales del tema para validar la investigación realizada.

6.2.1. Desarrollo de la encuesta

El cuestionario se divide en 4 secciones. En la primera sección, se solicita información con respecto al perfil de los profesionales, con la finalidad de conocer su experiencia a nivel general, y en el contexto específico de aplicaciones basadas en comunicación en tiempo real. En la segunda sección, se les plantea un escenario en el cuál son los encargados de desarrollar un proyecto que tiene como prioridad establecer comunicación en tiempo real entre dispositivos Android y iOS, bajo diferentes escenarios. En dicha sección, se les preguntaría acerca de las decisiones que tomarían con respecto a qué tecnología utilizar, y cuánto tiempo les tomaría llegar a dicha decisión. En la tercera sección, se les da a conocer información general con respecto a las tecnologías que investigamos. En la cuarta sección, se les da a conocer los resultados de las mediciones realizadas para cada tecnología. Por último, se le hace pregunta con respecto a la utilidad de la investigación presentada, y el impacto que esta tendría sobre la toma de decisiones sobre el proyecto planteado.

6.2.2. Encuestados

Las personas a encuestar fueron elegidas en base a su perfil profesional y amplia experiencia en el desarrollo de aplicaciones móviles. Se tomó una muestra de 10 desarrolladores, especialistas en tecnologías Android o iOS. A fin obtener resultados más significativos, se solicitó la ayuda de desarrolladores de diversos lugares del mundo: Perú, Alemania, Suecia, Alemania, Romania, Estados Unidos y Francia.

Figura 33 Encuestados : Rol en su compañía

Figura 34 Encuestados : Años de experiencia desarrollando aplicaciones

Como parte de la encuesta, se obtuvo que de todos los encuestados, 4 tenían experiencia solo en Android, 1 sólo en iOS, y 5 en ambos sistemas operativos. En promedio, los encuestados han desarrollado aplicaciones por 5 años. Además, todos ellos son o han sido desarrolladores de aplicaciones, y 4 de ellos cuentan ocupan roles de liderazgo, por lo cual tienen mayor impacto en cuanto a la toma de decisiones sobre el uso de alguna tecnología. Asimismo, los encuestados trabajaban en su mayoría en empresas reconocidas internacionalmente (Google / Atlassian / Fandango / Belatrix / Santex / Evry), o son consultores expertos.

6.2.3. Resultados

Los resultados más relevantes se presentan a continuación:

Figura 35 Tecnologías que los desarrolladores conocían antes de leer nuestra investigación

How useful do you think the presented research would be, once the requirements for the project are specified on detail?

11 responses

Figura 36 Encuesta : Percepción de la utilidad de la investigación

How much time do you consider the initial research phase of the project would take, if you use the presented resources?

11 responses

Figura 37 Encuesta : Tiempo destinado a la elección de una tecnología, luego de ver la investigación

Would you recommend the presented research to someone developing a similar project?

11 responses

Figura 38 Encuesta : Recomendación de la investigación

7. CAPÍTULO 7: GESTIÓN DEL PROYECTO

En esta sección, se presentarán y describirán a detalle los principales planes de gestión utilizados durante el transcurso del proyecto. Los planes elaborados involucran la gestión del tiempo, recursos humanos, comunicaciones y riesgos.

7.1. Gestión del Tiempo

El plan de gestión de tiempo tiene como objetivo gestionar las actividades que involucran cada entregable teniendo en cuenta las fechas de presentaciones definidas en el cronograma, y con ello poder realizar dichas actividades en el tiempo planificado para asegurar el cumplimiento de entregables críticos del proyecto.

La gestión semanal del proyecto se ve reflejada en el plan de trabajo semanal del proyecto, donde se indican las horas que las actividades deberían tomar, así como las horas reales tomadas por actividad. También se registra el avance del proyecto según los porcentajes de cumplimiento a la fecha, con ello se busca identificar posibles puntos que generen sobre tiempo y puedan afectar los procesos de gestión y desarrollo.

En caso se presente algún cambio en el alcance del proyecto durante su transcurso se deberán identificar los elementos afectados y se deberá replantear las actividades del cronograma estableciendo el nuevo objetivo, así como también los documentos de gestión de proyectos. Este proceso debe ser aprobado por medio de una solicitud de cambios por todos los interesados para efectuar los cambios correspondientes.

7.2. Gestión de los Recursos Humanos

El plan de gestión de recursos humanos tiene como objetivo organizar y dirigir el equipo de trabajo del presente proyecto. Por ello cada participante cumple un rol y responsabilidades específicas dependiendo de sus habilidades y fortalezas.

Durante el transcurso de la primera etapa del proyecto, no se solicitó los servicios de la empresa Software Factory para obtener recursos que pudieran apoyar con los procesos de gestión y desarrollo del proyecto. Debido a ser un proyecto de investigación de nuevas tecnologías de comunicación y por la variabilidad del trabajo no resulto necesario contar con el apoyo de un recurso adicional de software. Las aplicaciones desarrolladas fueron pruebas de concepto que lograron alcanzar los objetivos propuestos.

Descripción de Roles

Los roles involucrados en el desarrollo del proyecto son los siguientes:

Tabla 12 Descripción de Roles²³

Rol	Descripción
Comité de Proyectos	Rol correspondiente al conjunto de coordinadores y director de carrera encargados de calificar el cumplimiento del proyecto según lo planificado.
Gerente General Innova TI	Rol correspondiente al gerente de la empresa Innova TI, encargado de establecer y cumplir los objetivos principales de la empresa con respecto a la participación en el proyecto.
Jefe de proyecto	Rol correspondiente al alumno encargado del proyecto Análisis Comparativo de Mecanismos de Comunicación Cross-platform, encargado de cumplir con los objetivos propuestos para el proyecto.
Jefe de desarrollo	Rol correspondiente al alumno encargado del proyecto Análisis Comparativo de Mecanismos de Comunicación Cross-platform, encargado de cumplir con las pruebas de concepto y aplicativo final del proyecto.

²³ Fuente: Elaboración Propia

Cliente	Rol que desempeña el interesado asignado al proyecto, el cual se encarga de ver los requerimientos de la aplicación y el avance del proyecto, de acuerdo a las actividades que solicita.
Recurso Software Factory	Rol que desempeña el recurso de Taller de desempeño profesional, el cual se encarga de realizar ciertas actividades en el desarrollo de las pruebas de concepto y del aplicativo final.
Recurso Quality Services	Rol que desempeña el recurso de Taller de desempeño profesional, el cual se encarga de revisar los documentos de gestión del proyecto.

7.3 Gestión de las Comunicaciones

El plan de gestión de las comunicaciones tiene como objetivo que todas las necesidades de los Stakeholders sean transmitidas durante el transcurso del proyecto, así como todo cambio que afecte el alcance del mismo.

Relación entre el Jefe de Proyecto e Involucrados.

El principal medio de comunicación entre el equipo de proyecto y los interesados se realiza por medio del correo electrónico. La relación del Jefe de Proyecto con los involucrados es la siguiente:

- **Jefe de Proyecto y Cliente:** Con respecto al cliente del proyecto la comunicación será vía correo electrónico y presencial, donde semanalmente se presentaron los avances y entregables del proyecto, así como cualquier duda o inconveniente presentada. Además, se acordó que se tendrá una reunión semanal los días sábados a las 8 pm para la presentación de los entregables respectivos. Asimismo, durante la primera etapa del proyecto se llevaron a cabo todas las reuniones programadas.
- **Jefe de Proyecto y Recursos:** Con respecto a los recursos de las empresas de Software Factory no fue necesario concretar ninguna reunión o comunicación debido a que todo el desarrollo fue completado por el jefe de desarrollo.
- **Jefe de Proyecto y Empresa Innova TI:** Con respecto a empresa Innova TI la comunicación será por medio de un correo electrónico donde todas consultas serán respondidas por el Profesor Gerente de dicha empresa con un plazo no mayor a cuarenta y ocho horas. Además, se tiene una reunión semanal con el Gerente de la empresa para presentar los avances del proyecto
- **Jefe de Proyecto y Otras Empresas:** Con respecto al resto de empresas la comunicación será por medio de un correo electrónico donde todas consultas serán respondidas por el Alumno Gerente correspondiente a cada empresa con un plazo no mayor a cuarenta y ocho horas, además se tiene la opción de elaborar una reunión personal con el Gerente de alguna de las empresas también mediante un correo electrónico.

Guía de reuniones.

Las reuniones siguieron las siguientes pautas:

- Se manejó una agenda para cada reunión de acuerdo a los temas de interés a tratarse, así como también las tareas pendientes de reuniones anteriores.
- Se contó con un horario específico para cada reunión, sin embargo, será necesario confirmar o recordar la reunión vía correo electrónico un día antes como máximo.
- No se postergó ninguna reunión.
- Al finalizar cada reunión se recopiló lo tratado, resaltando los acuerdos para posteriormente elaborar el acta de reunión.
- Se contó con actas de reunión firmadas por el jefe de proyecto, jefe de desarrollo, gerente general y cliente.

Guía para Elaboración de Correos.

Las remisiones de correos electrónicos siguieron las siguientes pautas:

- Cada correo electrónico fue redactado de una manera formal.
- Todo correo electrónico enviado al cliente fue copiado a todo el equipo de proyecto; en especial al jefe de proyecto y desarrollo.
- Los correos electrónicos entre el gerente de la empresa y el cliente fueron enviados por el jefe de proyecto para establecer un estándar en la conexión.

CONCLUSIONES

Luego de haber realizado la investigación, análisis, diseño y desarrollo del presente proyecto: “Análisis Comparativo de Tecnologías de Comunicación Realtime sobre Cross-plaform”, se mostrarán las conclusiones que obtuvimos en la experiencia adquirida a lo largo del todo el proyecto.

- Se investigaron y desarrollaron pruebas de rendimiento de 11 tecnologías que nos permiten enviar mensajes en tiempo real sobre aplicaciones capaces de ser desplegadas en dispositivos Android y iOS. Cada una de ellas cumple su función, aunque fueron desarrolladas para cubrir diferentes casos de uso.
- En general, dependiendo del escenario en el que se esté realizando la prueba, se obtendrán diferentes resultados que son consistentes para todas las tecnologías.
- El envío por una red local resulta mucho más rápido ya que la información viaja una distancia muchísimo menor. Sin embargo, esta alternativa no es viable para proyectos que requieran comunicación entre dispositivos que no están en la misma red.
- Hay escenarios en los que algunas tecnologías destacan y otros en los que no. No existe un ganador absoluto, por lo cual aquellos desarrolladores deberán escoger la tecnología que mejor se adecue a las necesidades de su proyecto.
- La investigación demostró ser beneficiosa para la toma de decisiones con respecto al uso de una determinada tecnología. El 100% de los desarrolladores encuentran la investigación útil o muy útil, y el 90% de ellos recomendaría la investigación a personas que estén desarrollando proyectos relacionados a comunicación en tiempo real.

BIBLIOGRAFIA

- Aklilu, R. (2012). *Cross platform Mobile Applications, Mobile Apps Mobility. Norwegian University of Science and Technology.*
- Android. (2017). *Android NDK*. Recuperado el 20 de Junio de 2017, de developer.android.com: <https://developer.android.com/ndk/index.html>
- Android. (2017). *Android, the world's most popular mobile platform*. Recuperado el 28 de Abril de 2017, de developer.android.com: <https://developer.android.com/about/android.html>
- Android. (2017). *Kotlin and Android*. Recuperado el 20 de Junio de 2017, de developer.android.com: <https://developer.android.com/kotlin/index.html>
- Apple. (2017). *iOS Technology Overview*. Recuperado el 20 de Junio de 2017, de developer.apple.com: <https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html>
- Apple. (2017). *Programming with Objective-C*. Recuperado el 20 de Junio de 2017, de developer.apple.com: <https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>
- Brahler, S. (2017). *www.kit.edu*. Obtenido de Kit Institute for Tecnology: https://os.itec.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf
- Chi, P.-Y., Li, Y., & Hartmann, B. (2016). Enhancing Cross-Device Interaction Scripting with Interactive Illustrations. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 5482-5493.
- Gartner. (2016). *Gartner Says Five of Top 10 Worldwide Mobile Phone Vendors Increased Sales in Second Quarter of 2016*. Recuperado el 28 de Abril de 2017, de Gartner.com: <http://www.gartner.com/newsroom/id/3415117>
- Hamilton, P., & Wigdor, D. J. (2014). Conductor: enabling and understanding cross-device interaction. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2773-2782.
- Heinemann, L., & Robillard, M. (2012). *Effective and efficient reuse with software libraries*.

- IDC. (2016). *Smartphone Vendor Market Share, 2016 Q3*. Recuperado el 28 de Abril de 2017, de idc.com: <http://www.idc.com/promo/smartphone-market-share/vendor>
- Java. (2017). *¿Qué es Java?* Recuperado el 20 de Junio de 2017, de java.com: https://www.java.com/es/about/whatis_java.jsp
- Kotlin. (2017). *Using Kotlin for Android Development*. Recuperado el 20 de Junio de 2017, de kotlinlang.org: <https://kotlinlang.org/docs/reference/android-overview.html>
- Newman, S. (2015). *Building Microservices*. O'Reilly Media, Inc.
- Paay, J., Raptis, D., Kjeldskov, J., Lauridsen, B., Penchey, I., Ringhaug, E., & Ruder, E. (2016). A Comparison of Techniques for Cross-Device Interaction from Mobile Devices to Large Displays. *Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media*, 137-146.
- Scharf, F., Wolters, C., & Herczeg, M. (2013). Cross-Device Interaction: Definition, Taxonomy and Applications. *AMBIENT*, 35-41.
- Wexelblat, R. L. (1981). *History of programming languages I*. New York, NY, USA: ACM.