UNIVERSIDAD DE EL SALVADOR
FACULTAD MULTIDISCIPLINARIA ORIENTAL
DEP. INGENIERIA Y ARQUITECTURA

Aplicación de hora mundial

Asignatura: Diseño de Sistemas II

Docente: Ing. Ligia Astrid Bonilla Hernández

Presentan:

García Meléndez, Yenifer Zuleyma GM15002
Hernández Hernández, Stefany Magaly HH14012
Iraheta Medrano, Luis José IM15005

Ciudad Universitaria - Ciclo II
viernes 21 de septiembre de 2018.

Este presente trabajo se desarrolla el diseño de una aplicación que muestre al seleccionar un país la hora de dicho país y también tendrá un reloj digital de la hora local con el objetivo de saber la hora exacta de un país y notar la diferencia de hora de dos lugares distintos.

Se analizarán los requerimientos necesarios especificados por el usuario para tener una mejor visión de lo que desea y presentarle un prototipo de la aplicación. De esta manera buscamos que sea fácil y rápido el manejo de esta aplicación, que sea funcional y práctico para el usuario.

Se especificarán el modelo de ciclo de vida, paradigma de programación, la estructura del software y el patrón o anti patrón de dicha aplicación con el fin de tener un mejor control y conocimiento de lo que contendrá la aplicación en su etapa de desarrollo. Tomando en cuenta los factores de costo, tiempo y adaptabilidad se analizan las ventajas y los beneficios que tiene contar con una aplicación tan práctica para conocer distintas horas de diferentes partes del mundo.

Contenido

I. Planteamiento del problema

1.1. Situación problemática

Existen diferentes herramientas que se utilizan para conocer la hora, métodos los cuales desde tiempos antiguos van mejorando con el propósito de mejorar o facilitar el conocimiento del tiempo exacto, por ejemplo los relojes de sol que determinaban la hora en base a la proyección de la sobre sobre determinados objetos; curiosos en la historia idearon diferentes medidas para calcular y determinar el tiempo utilizando arena, agua, el sol, hasta llegar a relojes mecánicos hechos de manera mecánica y que eran fabricados con pequeñas piezas como manecillas adaptándolas en pequeños modelos los cuales eran más fáciles de llevar.

En el mercado industrial se fabrican a diario herramientas que se venden con el propósito de dar a conocer la hora a un usuario, pero con diseños o materiales llamativos, igualmente cumplen con el mismo propósito, aunque si se evalúa la precisión se estima que uno de los más exactos ya creados es el modelo atómico. Tales herramientas se caracterizan por el hecho de dar a conocer cierta hora en el lugar encontrado, pero existe la posibilidad de conocer una hora diferente del lugar donde se encuentra una persona; contando con diversas herramientas encontradas en internet empieza el trabajo de navegar e indagar la hora que marca un cierto lugar del mundo.

Desde la lógica presentada en el párrafo precedente y desde la perspectiva de un usuario refleja la curiosidad en el individuo en conocer horas internacionales mas no contando con una facilidad para conocerlo si este cuenta con una herramienta manual o no cuenta con accesos a la red, surge una problemática a la hora de acudir a una solución rápida donde de forma fácil y sin muchos procesos se conozca de manera exacta las diferencias zonas horarias alrededor del mundo.

1.2. Objetivos

General

- Desarrollar una aplicación móvil para IPhone IOS que muestre la hora local y las diferentes horas en zonas horarias al seleccionar un país.

Específicos

- Analizar los requerimientos funcionales y no funcionales de la aplicación.
- Modelar un ciclo de vida con las fases de desarrollo de un modelo de prototipos.
- Realizar un diseño detallado con el lenguaje de modelado unificado (UML).
- Crear aplicación de hora mundial para el móvil IPhone IOS.

1.3. Situación problemática

Este proyecto busca de manera rápida facilitar la búsqueda de la hora exacta en una región sea de nuestra área o de otra región en cualquier parte del mundo, de manera que el usuario seleccione un país del que desee conocer la hora; basta con seleccionar y aparecerá un reloj que indique la zona horaria.

Para la construcción de esa aplicación se utilizan modelos, arquitectura o patrones que vayan de acuerdo a los requerimientos de manera que sea fácil y cumpla de manera correcta la función que se va a realizar; para esto el proyecto está desarrollado a base del modelo de ciclo de vida en prototipos el cual se recolectan los requerimientos, se construye un diseño rápido, se elabora el prototipo para luego ser evaluado y posteriormente seguir o refinar detalles o errores que queden en la aplicación. De esta manera se tiene un mejor enfoque en la eficacia de un algoritmo o en la adaptabilidad del usuario tomando la ventaja de corregir o mejorar aspectos lógicos en la aplicación.

En el desarrollo interno de la aplicación y dentro de los paradigmas de programación se utiliza la Programación Orientada a Objetos (POO) el cual se utiliza por la creación de clases, objetos y métodos que son utilizados, donde viene a innovar la forma de obtener resultados, los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, las clases pueden definir propiedades y comportamientos de un tipo de objeto concreto.

La arquitectura de software es el diseño de más alto nivel de la estructura de un sistema, también denominada arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan un marco definido y claro para interactuar con el código fuente del software, una arquitectura de software se selecciona y diseña con base en objetivos (requisitos) y restricciones. Los objetivos son aquellos prefijados para el sistema de información, pero no solamente los de tipo funcional, también otros objetivos como la mantenibilidad, audibilidad, flexibilidad e interacción con otros sistemas de información. Las restricciones son aquellas limitaciones derivadas de las tecnologías disponibles para implementar sistemas de información. Unas arquitecturas son más recomendables de implementar con ciertas tecnologías mientras que otras tecnologías no son aptas para determinadas arquitecturas. Por ejemplo, no es viable emplear una arquitectura de software de tres capas para implementar sistemas en tiempo real.

Trabajar con la arquitectura cliente-servidor en modelo de diseño de software las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema. La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.

Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño resulta ser una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Los patrones estructurales se enfocan en como las clases y objetos se componen para formar estructuras mayores, los patrones estructurales describen como las estructuras compuestas por clases crecen para crear nuevas funcionalidades de manera de agregar a la estructura flexibilidad y que la misma pueda cambiar en tiempo de ejecución lo cual es imposible con una composición de clases estáticas.

Al igual que en los patrones de creación los patrones estructurales se dividen en patrones orientados a clases y patrones orientados a objetos:

- De clase: Emplea interfaces para combinar clases incompatibles.
    - Patrones: Adaptador (de clases).

- De objeto: Utiliza objetos para la combinación de estructuras.
    - Patrones: Adaptador (de objetos), Puente, Composición, Decorador, Fachada, Flyweight y Proxy.

Como podemos apreciar existe un patrón Adapter orientados a clases y otro orientado a objetos. Se explicarán conjuntamente ya que su objetivo es el mismo, pero como ejemplo sólo veremos el Adapter orientado a los objetos ya que el orientado a clases requiere de herencia múltiple, característica que de la que prescinden los principales lenguajes de programación.

II. Marco Teórico

Aplicaciones Móviles

Una aplicación móvil, aplicación o app, es una aplicación informática diseñada para ser ejecutada en teléfonos inteligentes, tabletas y otros dispositivos móviles. Las aplicaciones permiten al usuario efectuar un conjunto de tareas de cualquier tipo profesional, de ocio, educativas, de acceso a servicios, etc., facilitando las gestiones o actividades a desarrollar. El acortamiento inglés app suele ser incorrectamente pronunciado por los hispanohablantes como /apepé/, tratándolo incorrectamente como una sigla.

Por lo general, se encuentran disponibles a través de plataformas de distribución, operadas por las compañías propietarias de los sistemas operativos móviles como Android, iOS, BlackBerry OS, Windows iPhone, entre otros. Existen aplicaciones móviles gratuitas u otras de pago, donde en promedio el 20 a 30 % del coste de la aplicación se destina al distribuidor y el resto es para el desarrollador. El término app se volvió popular rápidamente, tanto que en 2010 fue listada como la palabra del año de la American Dialect Society.

Al ser aplicaciones residentes en los dispositivos están escritas en algún lenguaje de programación compilado, y su funcionamiento y recursos se encaminan a aportar una serie de ventajas tales como:

- Un acceso más rápido y sencillo a la información necesaria sin necesidad de los datos de autenticación en cada acceso.
- Un almacenamiento de datos personales que, a priori, es de una manera segura.
- Una gran versatilidad en cuanto a su utilización o aplicación práctica.
- La atribución de funcionalidades específicas.
- Mejorar la capacidad de conectividad y disponibilidad de servicios y productos (usuario-usuario, usuario-proveedor de servicios, etc.).

Un sistema operativo es un programa o conjunto de programas informáticos que gestiona el hardware de un dispositivo y administra el servicio de aplicaciones informáticas (Windows, iOS, Android, etc.).

Las aplicaciones web son herramientas alojadas en un servidor, a las que los usuarios pueden acceder desde Internet (o Intranet) mediante un navegador web genérico o específico, dependiendo del lenguaje de programación (moodle). Un servicio de alojamiento informático o web permite a organizaciones e individuos subir, alojar, gestionar o almacenar contenido en servidores físicos o virtuales. Por ejemplo: Dropbox sería una aplicación de software destinada a ser un servicio de alojamiento de archivos multiplataforma en la nube, a la cual se puede acceder a través un interfaz web o de una app.

En los últimos años, los servicios de informática distribuida han permitido que las organizaciones, incluidas las educativas, puedan gestionar sus procesos, actividad y aplicaciones informáticas a través de empresas que ofrecen comercialmente software como servicio (SaaS) alojado en un centro de datos o en servicios en la nube, y grandes redes de ordenadores pueden formar una "malla" que representa una potencia considerable (Google, Amazon, Microsoft).

El desarrollo de aplicaciones para dispositivos móviles requiere tener en cuenta las limitaciones de estos dispositivos. Los dispositivos móviles funcionan con batería y las principales características que se deben considerar son: gran variedad de tamaños de pantalla, datos específicos de software y hardware como también distintas configuraciones. El desarrollo de aplicaciones móviles requiere el uso de entorno de desarrollo integrado.

Las aplicaciones móviles pueden aprovechar mucho más el contexto en el que se ejecutarán, sobre todo si se comparan con las aplicaciones tradicionales. Ello se debe a diferentes factores, entre los que se encuentran las capacidades actuales en hardware de los dispositivos, o la capacidad de acceder a la información del usuario a la que el propio dispositivo tiene acceso. Los dispositivos actuales aportan mucha información sobre el entorno del usuario. Por ejemplo, aportan información sobre la posición geográfica del mismo, lo cual permite desarrollar aplicaciones basadas en la localización, conocidas como (LBS o servicios basados en localización), un ejemplo de tales aplicaciones es el Waze. Así mismo, existen otras informaciones (como, por ejemplo, orientación, presión, luz, etc.). La posibilidad de grabar imágenes, vídeos, y audio también aportan información sobre el entorno del contexto del usuario (por ejemplo, aplicaciones que reaccionan al habla o las de realidad aumentada).

React Native: desarrollando apps nativas usando JavaScript

React Native es un framework desarrollado por Facebook que permite desarrollar apps nativas iOS y Android usando JavaScript. Desarrolla apps nativas usando JavaScript, sin Objective-C o Java de por medio. React Native lleva como un "puente" donde su función es la de traducir el código React Native en Objective-C, para el caso de iOS, y Java en Android.

Es algo fácil que permite a los programadores web poder desarrollar apps nativas sin tener que aprender nuevos lenguajes de programación muy específicos para cada una de las plataformas. A parte que la sintaxis de React Native es bastante clara y sencilla, además de heredar el mismo diseño que React, aportando flexibilidad y reaprovechamiento en el código. Faltaría añadir por último que el código fuente que desarrollemos, en nuestra primera app, será un 100% compartido con iOS y Android. Esto quiere decir que el código JavaScript que hagamos nos servirá tanto para la app en iOS como en Android.

Lo primero es que React Native compila código JS en el correspondiente código nativo directamente. Pero es bastante dura de realizar, ya que Java y Objective C/Swift son lenguajes fuertemente tipados mientras que JavaScript no lo es. En lugar de eso, RN hace algo más inteligente: React Native es, en esencia, un conjunto de componentes React, donde cada uno de ellos tiene su correspondiente equivalente en views y componentes nativos.

Por ejemplo, el componente nativo TextInput tiene su correspondiente en RN que puede ser importado dentro del código JS y ser utilizado como cualquier componente React. De ahí que el programador frontend escriba código como si estuviera desarrollando una web en ReactJS. Hay tres componentes 'behind the scenes' cuando se ejecuta una RN app:

1. Módulos/códigos nativos: son todos los módulos necesarios (tanto de iOS como de Android) para que cuando hagamos nuestra RN app no tengamos que preocuparnos de escribir código nativo.

2. JavaScript VM: es la Virtual Machine de JavaScript que ejecutará nuestro código JS. Tanto en iOS como en Android se utiliza JavaScript Core, que es el motor JavaScript que utiliza Webkit para Safari. Esta pieza de software ya viene incluida en los dispositivos iOS, pero no en Android. Por lo que RN empaquetará esta pieza dentro del apk, lo que incrementa el tamaño en 3-4 megabytes.

3. React Native Bridge: es un bridge React Native escrito en C++/Java y es responsable de comunicar los hilos de JavaScript y de nativo. Entre ellos se hablan en un protocolo de mensajes.

Hay principalmente dos formas de trabajar con React Native:

1. La oficial: la llevamos a cabo instalando la utilidad node create-react-native-app e instalando en el móvil la Expo app para visualizar las aplicaciones que se van desarrollando. Con esta vía es posible que requieras de ajustes en código nativo (XCode/Android Studio) sobre todo a la hora de hacer releases y publicar la aplicación. Por tanto, requiere de tener cierto conocimiento de las plataformas nativas o al menos tener algún compañero cerca con conocimientos de ello.
2. La otra forma es utilizar Expo, que lo forman un grupo de desarrolladores que han creado un conjunto de herramientas que facilitan el trabajo con React Native. Por ejemplo, con la herramienta de línea de comando Expo CLI se puede crear 'from scratch' una app RN:

Con Expo se hace más fácil las pruebas en simuladores y en dispositivos. Además, distribuyen una librería Expo SDK que facilita integrar capacidades de los móviles como Push Notifications, Facebook login, Instant updating, etc.

También tiene la posibilidad de publicar tus aplicaciones bajo urls que puedes compartir de forma fácil. Y, sobre todo, aísla completamente al programador de conocer lenguajes de programación nativos.

En todos los casos se obtiene un código QR que, escaneándolo desde la Expo App, se puede ejecutar nuestro código en los dispositivos físicos.

También disponemos de las ventajas de tener hot reloading y debugging con Chrome. Eldebugging con Chrome se consigue haciendo que el código JS se ejecute dentro de Chrome en vez de en JavascriptCore y se comunique con los módulos nativos vía Websocket.

III. Análisis

3.1. Análisis de requerimientos

3.1.1. Requerimientos Funcionales

- Al seleccionar un país que muestre la zona horaria de dicho país.
- La aplicación tiene que ser orientada a iOS

3.1.2. Requerimientos no funcionales

- Interfaz tiene que ser amigable
- La ventana que se ajuste a la pantalla del celular
- Con un reloj digital como inicio de la aplicación.
- La aplicación debe funcionar sin conexión a internet.

3.2. Matriz de alternativas de la aplicación

Criterio de Factibilidad	**Alternativa 1: App Nativa Xcode**	**Alternativa 2: Reac-Native**	**Alternativa 3: Xamarin**
Factibilidad operativa: descripción de a qué grado la solución beneficiaria y que tan bien trabaja el sistema.	Entorno solo para desarrollo de app en iOS. **Calificación:7**	Permite crear aplicaciones móviles usando solo JavaScript. Utiliza el mismo diseño que React. **calificación: 10**	Las aplicaciones compiladas mediante Xamarin contienen controles de interfaz de usuario estándar y nativos. **Calificación:10**
Factibilidad Económica: Costo para desarrollar	se encuentra disponible de manera gratuita en el Mac App Store **Calificación:10**	Se encuentra gratuita **Calificación:10**	Gratuita **Calificación:10**
Factibilidad Técnica: Evaluación de la disponibilidad y necesidad de la tecnología.	Su primera versión tiene origen en el año 2003 y actualmente su versión número 9. **Calificación:6**	Se tiene conocimiento del lenguaje de programación de este entorno. **Calificación:10**	No se está muy familiarizado con este entorno. **Calificación:8**
Calificación	**22**	**30**	**28**

Por ser la calificación más alta y habiendo comparado factibilidades se trabajará con el entorno de desarrollo React Native por el conocimiento previo que se tiene de JavaScript, por la factibilidad económica y operativa que se tiene.

3.3. Diseño lógico

3.2.1. Modelo de ciclo de vida

El término ciclo de vida del software describe el desarrollo de software, desde la fase inicial hasta la fase final. El propósito de este programa es definir las distintas fases intermedias que se requieren para validar el desarrollo de la aplicación, es decir, para garantizar que el software cumpla los requisitos para la aplicación y verificación de los procedimientos de desarrollo: se asegura de que los métodos utilizados son apropiados.

Estos programas se originan en el hecho de que es muy costoso rectificar los errores que se detectan tarde dentro de la fase de implementación. El ciclo de vida permite que los errores se detecten lo antes posible y, por lo tanto, permite a los desarrolladores concentrarse en la calidad del software, en los plazos de implementación y en los costos asociados.

	CASCADA	SCRUM	PROTOTIPO
RH	8	4	9
Tamaño del proyecto	8	7	9
Tiempo reducido	8	5	9
Costo	10	10	10
Documentación	10	7	10
Total	44	33	47

El modelo de ciclo de vida que se presentará para la aplicación será el siguiente:

Modelo de proceso basado en prototipo

Un prototipo es un modelo experimental de un sistema o de un componente de un sistema que tiene los suficientes elementos que permiten su uso

Objetivos:

- Son un medio eficaz para aclarar los requisitos de los usuarios e identificar las características de un sistema que deben cambiarse o añadirse
- Mediante el prototipo se puede verificar la viabilidad del diseño de un sistema mediante el prototipo se puede verificar la viabilidad del diseño de un sistema.

Características:

- Es una aplicación que funciona
- Su finalidad es probar varias suposiciones con respecto a las características requeridas por el sistema
- Se crean con rapidez Se crean con rapidez
- Evolucionan a través de un proceso iterativo
- Tienen un costo bajo de desarrollo

Elegimos el modelo de ciclo de vida basado en prototipo porque el software se presentará una pequeña suposición de cómo quedaría la aplicación, por el poco tiempo que se nos dio para entregar el proyecto y porque es fácilmente modificable, si no le llegara gustar al cliente pues se modifica, y no habría que esperar hasta que el proyecto esté totalmente termino para que el cliente le pueda dar el visto bueno y si le pareció el prototipo.

A continuación, se detallará cada fase del modelado basado en prototipo:

Recolección y refinanciamiento de requerimientos:

Esta es la primera etapa en que se recolecta los requerimientos especificados por el cliente:

Requerimientos del software

Requerimientos funcionales: aplicación para iOS

Descripción del software: al seleccionar un país que muestre la zona horaria de dicho país.

Interfaz: una aplicación amigable, la ventaja que se ajuste a la pantalla del celular y con un reloj digital.

Requerimientos del hardware: la aplicación debe funcionar sin conexión a internet.

Diseño rápido

En esta estaba se propuso trabajar con modelado de basado en prototipo por el tiempo que se solicita la aplicación y la adaptabilidad del proyecto.

Construcción del prototipo

El desarrollo de la aplicación será en React Native que permite crear aplicaciones móviles usando solo JavaScript. Utiliza el mismo diseño que React, lo que permite componer una interfaz de usuario móvil rica a partir de componentes declarativos.

Evaluación del prototipo por el cliente

Se espera la evaluación por parte del cliente y verificar si se cumplieron los requerimientos especificados por el cliente.

Refinanciamiento de prototipo

Si el prototipo necesita de unos cambios porque no se han satisfecho las necesidades del cliente, se tomará un periodo de tiempo para hacer otro prototipo. Si el cliente acepta el prototipo se procederá a la parte de refinar los últimos detalles de la aplicación.

Producto de ingeniería

Ya con el proyecto finalizado se tiene un producto para ser entregado al cliente.

3.2.2. Paradigma de programación

Existe una infinidad de definiciones de lo que es un paradigma. Un paradigma es un determinado marco desde el cual miramos el mundo, lo comprendemos, lo interpretamos e intervenimos sobre él. Abarca desde el conjunto de conocimientos científicos que imperan en una época determinada hasta las formas de pensar y de sentir de la gente en un determinado lugar y momento histórico. Adam Smith define paradigma, en su libro "Los poderes de la mente", como "un conjunto compartido de suposiciones. Es la manera como percibimos el mundo: agua para el pez. El paradigma nos explica el mundo y nos ayuda a predecir su comportamiento".

En nuestro contexto, el paradigma debe ser concebido como una forma aceptada de resolver un problema en la ciencia, que más tarde es utilizada como modelo para la investigación y la formación de una teoría. También, el paradigma debe ser concebido como un conjunto de métodos, reglas y generalizaciones utilizadas conjuntamente por aquellos entrenados para realizar el trabajo científico de investigación.

En nuestro contexto, los paradigmas de programación nos indican las diversas formas que, a lo largo de la evolución de los lenguajes, han sido aceptadas como estilos para programar y para resolver los problemas por medio de una computadora.

	Lógica	POO	Eventos
Manejo al aplicar el paradigma	**4**	**9**	**6**
Facilidad a la hora de implementar	**5**	**8**	**7**
Compatibilidad al uso de patrones	**2**	**8**	**7**
Adaptabilidad al proyecto	**2**	**9**	**8**
Total	**13**	**34**	**28**

Programación orientada a objetos

Una clase se puede definir de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ella. La Herencia es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables registrados como "públicos" o public en C. Los componentes registrados como "privados" o private también se heredan, pero se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos. Para poder acceder a un atributo u operación de una clase en cualquiera de sus subclases, pero mantenerla oculta para otras clases es necesario registrar los componentes como "protegidos" (protected), de esta manera serán visibles en C y en D, pero no en otras clases.

Los Objetos son las Instancias de una clase. Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos. Se corresponden con los objetos reales del mundo que nos rodea, o con objetos internos del sistema (del programa).

Los Métodos son los algoritmos asociados a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

En el Evento Es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento la reacción que puede desencadenar un objeto; es decir, la acción que genera.

El problema con la abstracción de datos es que no hay ninguna distinción entre las propiedades generales y las particulares de un conjunto de objetos. Expresar esta distinción y aprovecharla es lo que define a la OOP a través del concepto de herencia. El paradigma de la programación orientada a objetos es, entonces:

a) Definir que clases se desean

b) Proporcionar un conjunto completo de operaciones para cada clase

c) Indicar explícitamente lo que los objetos de la clase tienen en común empleando el concepto de herencia.

En algunas áreas las posibilidades de la OOP son enormes. Sin embargo, en otras aplicaciones, como las que usan los tipos aritméticos básicos y los cálculos basados en ellos, se requiere únicamente la abstracción de datos y/o programación por procedimientos, por lo que los recursos necesarios para apoyar la POO podrían salir sobrando.

3.2.3. Arquitectura del software

Para el desarrollo de esta aplicación se contara con el modelo en capas utilizando dos capas, la capa de presentación y la capa de negocio

	Monolítica	2 Capas (p/s)	3 Capas (MVC)
Experiencia de uso	4	9	8
RH	6	8	8
Tamaño del proyecto	5	9	5
Adaptabilidad al proyecto	4	9	3
Total	19	35	24

Modelo en capas

Es un modelo de desarrollo software en el que el objetivo primordial es la separación (desacoplamiento) de las partes que componen un sistema software o también una arquitectura cliente-servidor: lógica de negocios, capa de presentación y capa de datos. De esta forma, por ejemplo, es sencillo y mantenible crear diferentes interfaces sobre un mismo sistema sin requerirse cambio alguno en la capa de datos o lógica.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, solo afectará al nivel requerido sin tener que revisar entre el código fuente de otros módulos, dado que se habrá reducido el Acoplamiento informático hasta una interfaz de paso de mensajes.

Además, permite distribuir el trabajo de creación de una aplicación por niveles; de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, de forma que basta con conocer la API que existe entre niveles.

En el diseño de sistemas informáticos actual se suelen usar las arquitecturas multinivel o programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten). Capa de presentación: la que ve el usuario (también se la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario. Esta capa se comunica únicamente con la capa de negocio.

Capa de negocio: es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.

En una arquitectura de tres niveles, los términos "capas" y "niveles" no significan lo mismo ni son similares.

El término "capa" hace referencia a la forma como una solución es segmentada desde el punto de vista lógico:

- **Presentación**. (Conocida como capa Web en aplicaciones Web o como capa de usuario en Aplicaciones Nativas)
- **Lógica de Negocio**. (Conocida como capa Aplicativa)

3.2.4. Patrones de diseño

Fábricas

Para comenzar con nuestro estudio debemos definir claramente qué es una factoría o fábrica. De la misma forma que sucede con muchos conceptos en la Ingeniería del Software, existen varias definiciones para este término (en la entrega anterior hemos experimentado esta situación para el concepto de patrón); por tanto, es fundamental clarificar este concepto para sentar las bases de nuestro estudio posterior.

	Simple Factory	Abstract Factory	Singleton
Dominio del patrón	7	6	6
Eficiencia	7	6	7
Facilidad de implementación	6	5	5
Encaje del sistema	7	5	7
Total	27	22	25

El término **factory** (o fábrica) adolece de ser sobre utilizado e impreciso. Mucha gente se refiere a factory para indicar una implementación de **Factory Method** o de **Abstract Factory** (patrones definidos en el libro del GoF). Otros, sin embargo, se refieren a un objeto que crea instancias de otros, aunque no sigue las reglas de los patrones mencionados anteriormente.

La falta de consenso sobre el término dificulta la comunicación que, como ya hemos visto anteriormente, es uno de los objetivos de los patrones (poder referirse en forma unívoca a una abstracción de mayor nivel que la clase individual, incrementando así el vocabulario de los ingenieros de software). Recordemos entonces una importante frase de la entrega anterior:

Los Patrones permiten establecer un vocabulario común de diseño, cambiando el nivel de abstracción a colaboraciones entre clases y permitiendo comunicar experiencia sobre dichos problemas y soluciones. Son también un gran mecanismo de comunicación para transmitir la experiencia de los ingenieros y diseñadores experimentados a los más nóveles, convirtiéndose en unas de las vías para la gestión del conocimiento.

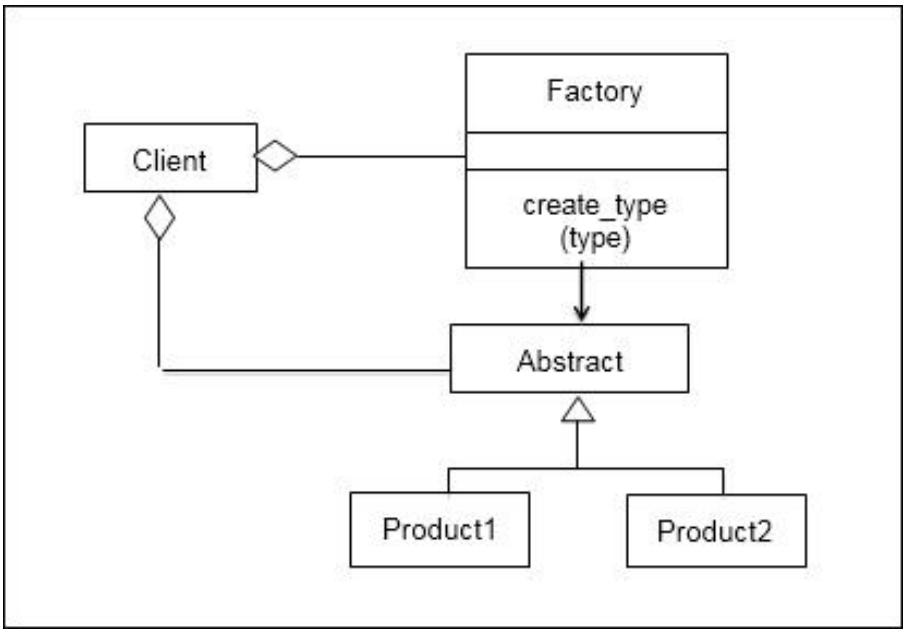
Por tanto, es muy importante establecer un significado claro para este término, a los efectos de construir este vocabulario común.

Llamaremos fábrica, factoría o factory a una clase que implemente uno o más métodos de creación, que son los métodos que se encargan de crear instancias de objetos (estas instancias pueden ser de esta misma clase o de otras). Esta clase tiene entre sus responsabilidades la creación de instancias de objetos, pero puede tener también otras responsabilidades adicionales. Los métodos de creación pueden ser estáticos.

Existen diferentes "tipos" de fábricas. A continuación, enumeraremos cada una de ellos, a los efectos de dejar bien en claro sus significados:

Simple Factory

Clase utilizada para crear nuevas instancias de objetos.



3.2.5. Diagramas

3.2.5.1. Diagrama de Casos de Uso

En el Lenguaje de Modelado Unificado, un diagrama de casos de uso es una forma de diagrama de comportamiento UML mejorado. El Lenguaje de Modelado Unificado (UML), define una notación gráfica para representar casos de uso llamada modelo de casos de uso. UML no define estándares para que el formato escrito describa los casos de uso, y así mucha gente no entiende que esta notación gráfica define la naturaleza de un caso de uso; sin embargo una notación gráfica puede solo dar una vista general simple de un caso de uso o un conjunto de casos de uso.

Los diagramas de casos de uso son a menudo confundidos con los casos de uso. Mientras los dos conceptos están relacionados, los casos de uso son mucho más detallados que los diagramas de casos de uso. En los conceptos se debe detallar más de un caso de uso para poder identificar qué es lo que hace un caso de uso.

- La descripción escrita del comportamiento del sistema al afrontar una tarea de negocio o un requisito de negocio. Esta descripción se enfoca en el valor suministrado por el sistema a entidades externas tales como usuarios humanos u otros sistemas.

- La posición o contexto del caso de uso entre otros casos de uso. Dado que es un mecanismo de organización, un conjunto de casos de usos coherentes y consistentes promueven una imagen fácil de comprender del comportamiento del sistema, un entendimiento común entre el cliente/propietario/usuario y el equipo de desarrollo.

En el diagrama de casos de uso de esta aplicación móvil muestra la interacción que tiene el usuario con la aplicación a desarrollar, explicando de manera gráfica y detallada los movimientos que realiza el usuario con el fin de cumplir el propósito de la aplicación que es mostrar la hora del país seleccionado.

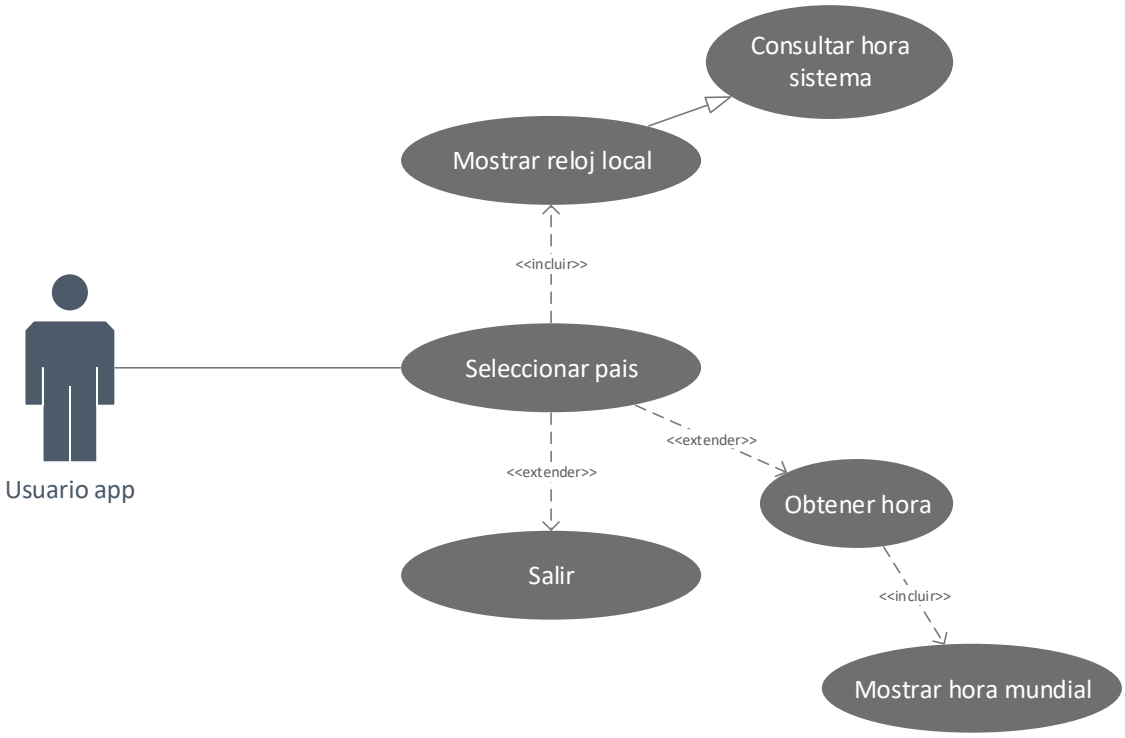


Diagrama 1: Casos de Uso

Aplicación hora mundial

Autores:
Luis Iraheta, Yenifer García, Magaly Hernández

Fecha: Jueves 13 de septiembre de 2018
Versión: 1.0

Nombre de caso de uso:	Seleccionar país	Caso de usos aplicado a la información del usuario.
ID del Caso de uso:	CU-0001	
Prioridad:	Alta	
Fuente:	Requerimiento CU-0001.R1	
Actor primario de negocios:	Usuario de app	
Otros actores participantes:	Ninguno	
Otros Involucrados interesados:	Ninguno	
Descripción:	Este caso de uso describe el evento de un usuario de la aplicación que selecciona un país, Una vez seleccionado se obtiene la hora del país para ser mostrado. Al completarse, al actor se le muestra la hora del país que selecciona.	

Aplicación hora mundial

Autores:
Luis Iraheta, Yenifer García, Magaly Hernández

Fecha: Jueves 13 de septiembre de 2018
Versión: 1.0

Nombre de caso de uso:	Mostrar reloj local	Caso de usos aplicado a la información del usuario.
ID del Caso de uso:	CU-0002	
Prioridad:	Alta	
Fuente:	Requerimiento CU-0002.R3	
Actor primario de negocios:	Usuario de app	
Otros actores participantes:	Ninguno	
Otros Involucrados interesados:	Ninguno	
Descripción:	Se muestra la información del reloj local donde se encuentra ubicado el dispositivo. La actualización del reloj se hará cada segundo. Se mostrara al actor principal la hora en un reloj digital.	

Aplicación hora mundial

Autores:
Luis Iraheta, Yenifer García, Magaly Hernández

Fecha: Jueves 13 de septiembre de 2018
Versión: 1.0

Nombre de caso de uso:	Consultar hora sistema	Caso de usos aplicado a la información del usuario.
ID del Caso de uso:	CU-0003	
Prioridad:	Alta	
Fuente:	Requerimiento CU-0003.R3	
Actor primario de negocios:	Usuario de app	
Otros actores participantes:	Ninguno	
Otros Involucrados interesados:	Ninguno	
Descripción:	Este caso de uso incluye mostrar la hora local del sistema para que este pueda ser invocado por la case Date y pueda obtener la hora del sistema.	

Aplicación hora mundial

Autores: Luis Iraheta, Yenifer García, Magaly Hernández

Fecha: Jueves 13 de septiembre de 2018

Versión: 1.0

Nombre de caso de uso:	Obtener hora	Caso de usos aplicado a la información del usuario.
ID del Caso de uso:	CU-0004	
Prioridad:	Alta	
Fuente:	Requerimiento CU-0004.R1	
Actor primario de negocios:	Usuario de app	
Otros actores participantes:	Ninguno	
Otros Involucrados interesados:	Ninguno	
Descripción:	Este caso de uso se extiende seleccionar hora, se obtendrá la información de la UTC en la que se encuentra al país, para ver cuánto es la diferencia de horas entre los países.	

Aplicación hora mundial

Autores:
Luis Iraheta, Yenifer García, Magaly Hernández

Fecha: Jueves 13 de septiembre de 2018
Versión: 1.0

Nombre de caso de uso:	Mostrar hora mundial	Caso de usos aplicado a la información del usuario.
ID del Caso de uso:	CU-0005	
Prioridad:	Alta	
Fuente:	Requerimiento CU-0005.R1	
Actor primario de negocios:	Usuario de app	
Otros actores participantes:	Ninguno	
Otros Involucrados interesados:	Ninguno	
Descripción:	Del caso de uso anterior (CU-0004) incluye que se hagan las operaciones en este caso de uso, sumando o restando las horas que hay de diferencia entre el país local y el país extranjero.	

Aplicación hora mundial

Autores:
Luis Iraheta, Yenifer García, Magaly Hernández

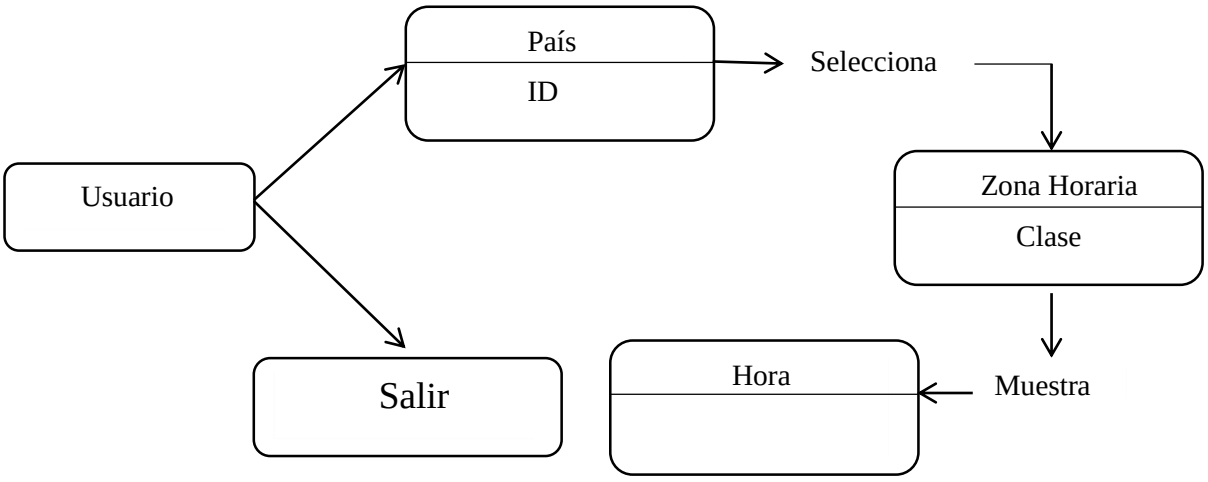
Fecha: Jueves 13 de septiembre de 2018
Versión: 1.0

Nombre de caso de uso:	Mostrar hora mundial	Caso de usos aplicado a la información del usuario.
ID del Caso de uso:	CU-0006	
Prioridad:	Alta	
Fuente:	--	
Actor primario de negocios:	Usuario de app	
Otros actores participantes:	Ninguno	
Otros Involucrados interesados:	Ninguno	
Descripción:	En los requerimientos no se establece la salida de la aplicación, más sin embargo en el diagrama UML se ha contemplado que exista un caso de uso que incluye salir de la aplicación, este cerrara la aplicación cuando se presione el botón.	

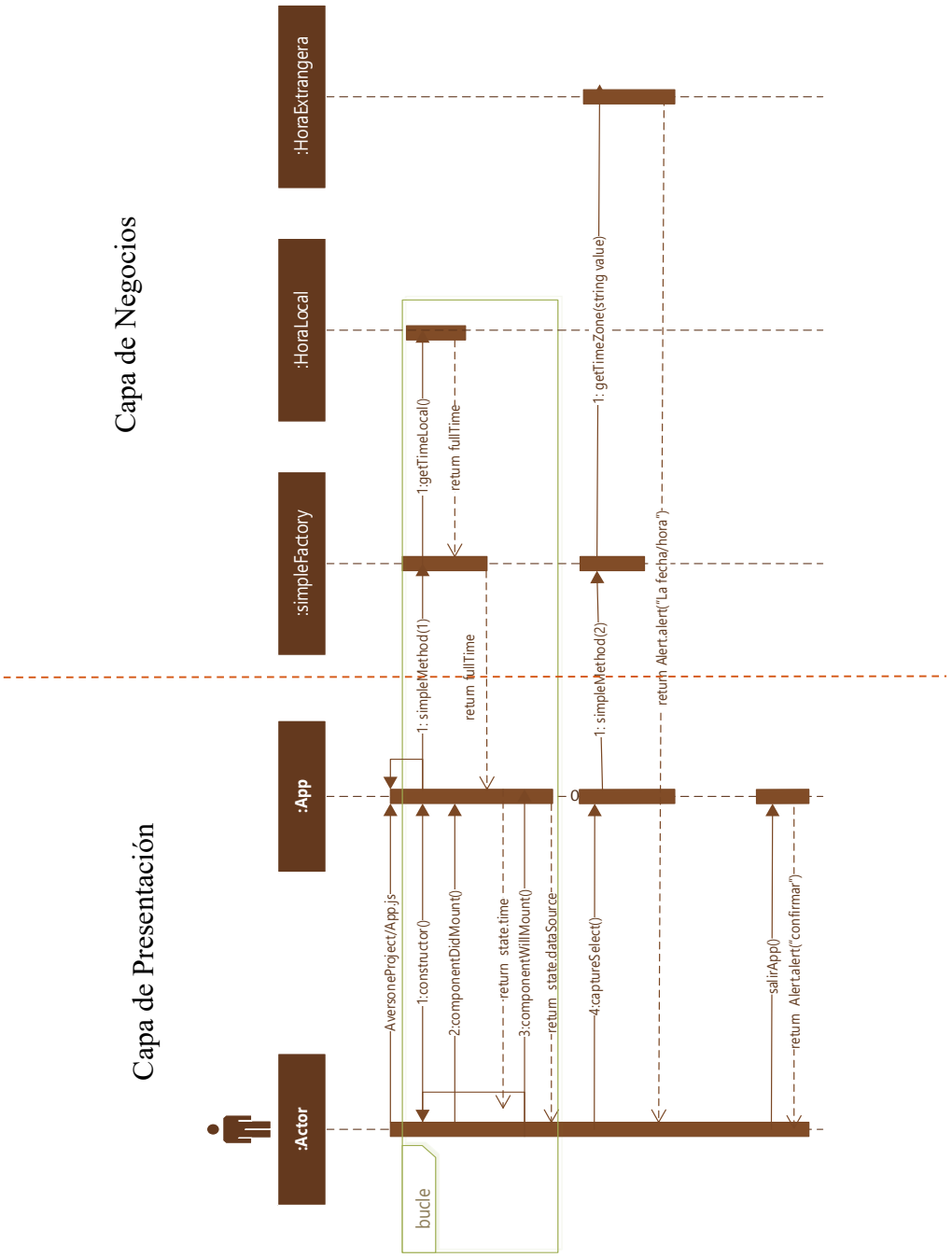
3.2.5.2. Diagrama de Modelo Conceptual

Modelo Conceptual, el cual nos muestra los conceptos presentes en el dominio del problema. Un concepto para este caso, en términos de la Programación Orientada a Objetos, es un objeto del mundo real; es decir, es la representación de cosas del mundo real y NO de componentes de *software*. En él no se definen operaciones (o métodos); en este modelo se pueden mostrar los conceptos, los atributos de los conceptos (opcionalmente) y la relación o asociación entre ellos. Informalmente podríamos decir que un concepto es una idea, cosa u objeto. Para descubrirlos debemos analizar los sustantivos en las descripciones textuales del dominio del problema, es decir, de la descripción del sistema, de los requerimientos y de los Casos de Uso

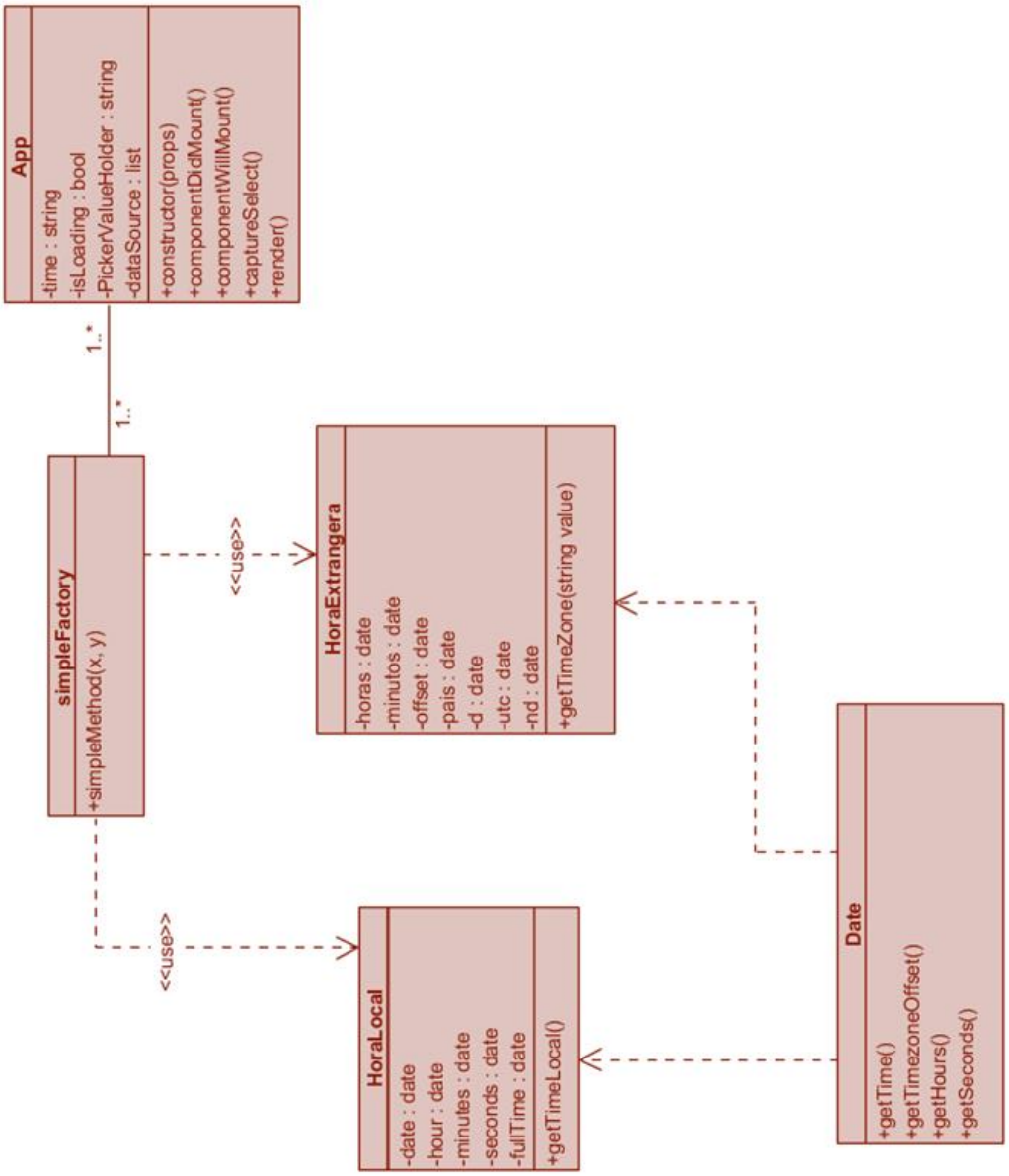
En la aplicación



3.2.5.3. Diagrama de Secuencia



3.2.5.4. Diagrama de Clases



3.2.6. Prototipo

El significado de los colores

Pero acertar con el color no es una mera cuestión de estética o de seguir la moda en cuestión de diseñar aplicaciones. Debemos recordar que el color también es una forma de comunicar, por eso cada uno de ellos significa algo diferente. Veamos qué se desprende de cada tonalidad y algunos ejemplos de apps de éxito.

Rojo: invita a la acción, a hacer algo, abre el apetito, simboliza la pasión y el amor, etc. Invita a ser impulsivo porque crea sensación de urgencia.

Amarillo: representa felicidad y aporta optimismo.

Azul: Aporta serenidad y paz al cliente/usuario, aumenta la productividad. Genera una sensación de seguridad y confianza.

Verde: Asociado con la naturaleza, la tranquilidad y la salud. Aunque está asociado con la riqueza y simboliza el dinero, se utiliza en marketing para serenar al cliente.

Morado: éxito y sabiduría. Utilizado generalmente para representar marcas creativas y originales.

Naranja: Refleja emoción y entusiasmo, por eso su uso se limita a llamadas a la acción y para generar leads y compras impulsivas. Pero siempre siendo confiable.

El color en diseño de apps según el sexo.

Aunque cada color diga una cosa, también debemos tener en cuenta el sexo de nuestro usuario tipo. Los hombres no comunican igual que las mujeres, por eso el discurso que emane el diseño de apps que tengamos en mente debe tener eso en cuenta.

¿Qué colores gustan más?

El azul es el color favorito del 57% de los hombres, el verde para el 14%, el negro para el 9% y el rojo para el 7%. Una clara ausencia entre los colores es el morado, del que hemos hablado antes y que está relacionado con el éxito. De hecho, el 22% afirma que éste él es color que menos les gusta.

Sin embargo, el 23% de las mujeres dice que el morado es precisamente su favorito. Muy repartido con el azul, que es el favorito del 35% de féminas. Empatan con los hombres al 14% con el verde, y un 9% de ellas adoran el rojo.

Pero en algo sí que se ponen de acuerdo hombres y mujeres, y es que los colores menos populares son los que se mueven entre el marrón y el naranja. Mientras que el azul sale victorioso en todos los sentidos.

En cuanto a la percepción, el estudio de Hubspot también nos lleva a descubrir que los hombres prefieren los colores brillantes mientras que las mujeres se sienten más afines a suaves o tonos pastel. Y del mismo modo el género masculino prefiere los colores oscurecidos o sombreados, mientras que el femenino prefiere la luz y colores con luz.

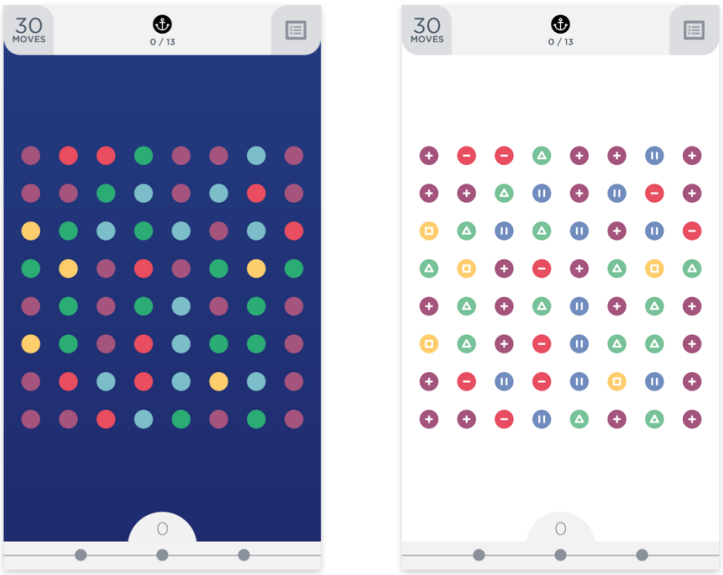
Diseño de apps y el daltonismo

Un aspecto que no se tiene muy en cuenta es que casi el 9% de la población sufre de daltonismo. Esta enfermedad afecta directamente a la capacidad de distinguir algunos colores entre sí. Si echamos cuentas, si no adaptamos el diseño de apps a este tipo de usuarios perderemos prácticamente a uno de cada 10 de los usuarios que descarguen nuestra app.

¿Cómo podemos solucionarlo?

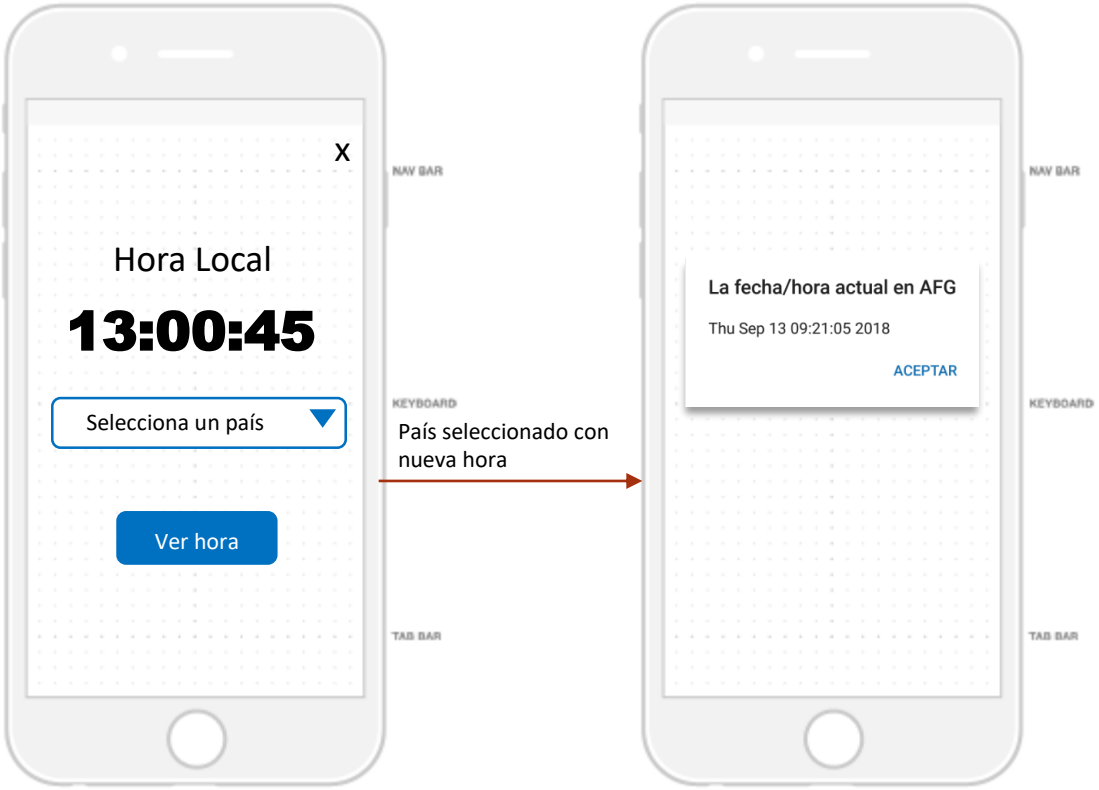
El daltonismo no consiste en ver en blanco y negro, sino que hay variantes de un color que no sabe distinguir. Por ejemplo, no distinguir las hojas marrones, naranjas y amarillas de un árbol con la llegada del otoño. Un daltónico vería todas del mismo color.

Por eso, muchas veces nos centramos en diseños elaborados que utilizan gamas de color limitadas y que juegan con ligeras variaciones. Y eso está muy bien. Pero también deberíamos tener en cuenta que habrá gente para la que esta opción será de todo menos usable.



3.2.6.1. Diseño de interface.

A continuación, se muestra un bosquejo de la aplicación no funcional con los requerimientos estipulados por el cliente.



El objetivo de una aplicación móvil es facilitar y satisfacer una necesidad de un usuario, se desarrolla con el propósito de optimizar procesos, satisfacer curiosidades, transacciones, etc. La aplicación de Hora Mundial nos permite conocer el tiempo exacto en el que se encuentra un país en cualquier parte del mundo, por lo tanto, el usuario debe hacer procesos como la búsqueda del país, dar click en el botón de obtener la hora y así sucesivamente.

El desarrollo de una app es importante cumplir el propósito para el que se está desarrollando de manera que tenga un óptimo funcionamiento, también necesita tener un diseño el cual sea familiar y que sea de fácil entendimiento para el usuario, ya que muchas aplicaciones no cuentan con lo que es un guía o un manual que indique como debe usarse, de manera que a la hora del diseño de la aplicación se analice cada uno de los aspectos que contiene la aplicación.

En nuestra aplicación móvil una vez que el usuario entra en la aplicación, se observa un reloj que trabaja de manera continua, situado al centro de la aplicación para que de esta manera el usuario tenga completa atención al reloj por su tamaño, posición y color lo hacen atractivo al ojo del usuario ya que tiene un tamaño adecuado y un color discreto. De igual forma un texto que nos indica de que es ese reloj “Hora Local”, nos indica la hora según nuestra localización.

En la parte inferior de la App tenemos una lista la cual contiene los diferentes países que existen en el mundo, estos al ser más de 250 países se colocan en la lista, de manera que el usuario busque de forma más rápida el país que desee, el espacio de la lista es el que abarca la mayor parte de la pantalla ya que esta deberá ser de un tamaño que se ajuste a la búsqueda del país, permitiendo así ver los países anteriores o siguientes al que se encuentra seleccionado.

Una vez seleccionado el país de la lista el usuario puede observar en la interfaz de la aplicación, un botón el cual al dar Click, se desplegará una alerta la cual contiene en la parte superior, un texto "La fecha/Hora actual en (País Seleccionado)" que se encuentra distinto por una fuente en Negrita, este nos indica el formato que se muestra en la parte inferior.

En la parte inferior se observa un formato de Día/Mes/Año y a un costado la hora del país que ha sido seleccionado en la lista, así el usuario conoce y cumple el propósito por el que ha sido desarrollada la aplicación. Si el usuario desea volver y seguir indagando o buscar otros horarios, se coloca un botón el cual se encuentra embebido en la alerta y este nos sirve para salir de la alerta y volver a la interfaz inicial.

IV. Estudio de Viabilidades

4.1. Viabilidad Operacional.

Es la menos técnica pero sí la más importante. Realizar el estudio para analizar si las necesidades del negocio pueden llegar a cumplirse a través de la idea propuesta. Además, medir en qué grado el sistema propuesto resuelve problemas y si se aprovecha de las oportunidades. Aquí se incluyen parámetros de diseño como la fiabilidad, compatibilidad del proyecto, facilidad de uso, facilidad de mantenimiento, accesibilidad y otros.

-En este caso la aplicación no se tiene la certeza de que será implementada después de desarrollada y por ende no tendrá mantenimiento de la aplicación.

-No se cuenta con toda la tecnología para desarrollar la aplicación ya que se requiere una computadora con Mac OS y es altamente costosa para nuestros recursos

- la aplicación solo podrá ser usada en iPhone 6 o posterior

- se podrá usar en iOS 8 o posterior.

4.2. Viabilidad Económica.

Permite a las empresas evaluar la viabilidad, estimando beneficios antes de asignar los recursos financieros y costes. Buenos resultados en esta área dan mucha fiabilidad al proyecto. Es un análisis generalizado de costes/beneficios.

Beneficios: la aplicación no tendrá ningún beneficio porque los costes para desarrollar la aplicación son muy altos debido a la plataforma en la que se está implementando.

Presupuesto de la aplicación se encuentra en anexos.

4.2.1. Puntos de caso de Usos

UUCW (Peso de los Casos de Uso sin Ajustar (1/2)

Categorías de los casos de uso sin ajustar

Categoría de los casos de uso	Descripción	Factor(Peso)
Simple	Transacciones=3 o menos Clases: menos de 5	5
Medio	Transacciones=4 a7 Clases:5 a 10	10
Complejo	Transacciones=más de 7 transacciones Clases: más de 10 clases	15

UAW (Pesos de los Casos de Uso sin Ajustar (2/2)

Formula: UUCW = Σ (cantidad de un tipo de caso de uso * Factor)

Tipo de Actor	Descripción	Peso (Factor)	Número de Actores	Resultado
Simple	Transacciones=3 o menos Clases: menos de 5	5	1	5
Medio	Transacciones=4 a7 Clases:5 a 10	10	0	0
Complejo	Transacciones=más de 7 transacciones Clases: más de 10 clases	15	0	0
Total UUCW				5

UAW (Peso de los actores sin Ajustar (1/2)

Evaluación de la complejidad de los actores con los que interactúa la aplicación.

Categoría de los casos de uso	Descripción	Factor(Peso)
Simple	Otro sistema que interactúa con el sistema a desarrollar mediante una interfaz de programación (API).	1
Medio	Otro sistema interactuando mediante un protocolo o una persona interactuando a través de una interfaz en modo texto.	2
Complejo	Una persona que interactúa mediante una interfaz gráfica (GUI).	3

UAW (Pesos de los actores sin Ajustar (2/2)

Formula: UAW = Σ (cantidad de un tipo de Factor * Factor)

Tipo de Actor	Descripción	Peso (Factor)	Número de Actores	Resultado
Simple	Otro sistema que interactúa con el sistema a desarrollar mediante una interfaz de programación (API).	1	0	0
Medio	Otro sistema interactuando mediante un protocolo o una persona interactuando a través de una interfaz en modo texto.	2	0	0
Complejo	Una persona que interactúa mediante una interfaz gráfica (GUI).Clases: más de 10 clases	3	1	3
Total UAW				3

Calculo de los puntos de caso de uso sin ajustar

UUCP=UUCW+UAW

UUCP=5+3

UUCP=7

2. Factor de Complejidad Técnica (2/5)

Compuesto por 13 puntos que evalúan la complejidad de los módulos del sistema que se desarrolla.

Factor Técnico	Descripción	Peso Dado
T1	Sistema distribuido	2
T2	Rendimiento o tiempo de respuesta	1
T3	Eficiencia del usuario final	1
T4	Procesamiento interno complejo	1
T5	El código debe de ser reutilizable	1
T6	Facilidad de instalación	0.5
T7	Facilidad de uso	0.5
T8	Portabilidad	2
T9	Facilidad de cambio	1
T10	Concurrencia	1
T11	Características especiales de seguridad	1
T12	Provee acceso directo a terceras partes	1
T13	Se requiere facilidades especiales de entrenamiento a usuario	1

Factor de Complejidad Técnica (3/5)

Cada uno de los factores se debe evaluar según la siguiente escala:

Descripción	Valor
Irrelevante	0 a 2

Medio	3 a 4
Esencial	5

TCF (Factor de Complejidad Técnica) (4/5)

Factor Técnico	Descripción	Peso	Impacto percibido	Factor calculado
T1	Sistema distribuido	2	2	4
T2	Rendimiento o tiempo de respuesta	1	2	2
T3	Eficiencia del usuario final	1	3	3
T4	Procesamiento interno complejo	1	0	0
T5	El código debe de ser reutilizable	1	4	4
T6	Facilidad de instalación	0.5	2	1
T7	Facilidad de uso	0.5	3	1.5
T8	Portabilidad	2	5	10
T9	Facilidad de cambio	1	4	4
T10	Concurrencia	1	2	2
T11	Características especiales de seguridad	1	0	0
T12	Provee acceso directo a terceras partes	1	0	0
T13	Se requiere facilidades especiales de entrenamiento a usuario	1	0	0
Factor Total Técnico				31.5

TCF=0.6+ (0.1*factor total técnico)

TCF=0.6+ (0.01*31.5)

TCF=0.915

3. ECF (Factor de Complejidad Ambiental) (2/4)

Factor Ambiental	Descripción	Peso
E1	Familiaridad con el modelo del proyecto utilizado Familiaridad con UML	1.5
E2	Personal tiempo parcial	-1
E3	Capacidad del analista líder	0.5
E4	Experiencia en la aplicación	0.5
E5	Experiencia en orientada a objetos	1
E6	Motivación	1
E7	Dificultad del lenguaje de programación	-1
E8	Estabilidad de los requerimientos	2

ECF (Factor de Complejidad Ambiental) (3/4)

Factor Ambiental	Descripción	Peso	Impacto percibido	Factor calculado
E1	Familiaridad con el modelo del proyecto utilizado Familiaridad con UML	1.5	1	1.5
E2	Personal tiempo parcial	-1	1	-1
E3	Capacidad del analista líder	0.5	3	1.5
E4	Experiencia en la aplicación	0.5	1	0.5
E5	Experiencia en orientada a objetos	1	3	3
E6	Motivación	1	4	4
E7	Dificultad del lenguaje de programación	-1	2	-2
E8	Estabilidad de los requerimientos	2	3	6
Factor Ambiental Total				13.5

ECF= 1.4+ (-0.3*factor ambiental total)

ECF=1.4+ (-0.03*13.5)

ECF=0.995

Calculando los UCP

UCP=UUCP*TCF*ECF

Los valores obtenidos

UUCP=7

TCF=0.915

ECF=0.995

UCP=7*0.915*0.995

UCP=6.372975

Agregando la productividad

Factor de productividad

Total, de horas estimadas=UCP*PF

Total, de horas estimadas=6.372975*20

Total, de horas estimadas=127

Para el costo del desarrollo del proyecto se utiliza una tarifa de pago al programador $25 dólares por hora.

Total en mano de obra para el desarrollo = $\boldsymbol{127h * \$25/h = \$3,175.00}$

4.3. Viabilidad Técnica.

Permite evaluar si los equipos, sistemas y software están disponibles y tienen las capacidades técnicas necesarias para cada propuesta de diseño planificado. También se analiza el factor humano, es decir, si el personal cuenta con la experiencia y conocimientos técnicos requeridos para el sistema que se propone. Es un análisis de los recursos técnicos disponibles en la organización.

Hardware

No se cuenta con una maquina con Mac OS para poder compilar el código, ante esto no se tiene la factibilidad de poder terminar de desarrollar la aplicación.

La aplicación va orientada a utilizarse en iPhone 6 en adelante.

Software

Se cuenta con la tecnología de software necesaria para desarrollar la aplicación como lo son el entorno de desarrollo que se especificó anteriormente React Native ya que es gratuita y se tiene el conocimiento para desarrollar en ella.

Recurso humano

- Analista de sistemas.
- Diseñador de sistemas.
- Programador.

Analista de sistemas

Descripción
Los analistas de sistemas informáticos adaptan y diseñan sistemas de información para ayudar a las empresas trabajar de forma más rápida y eficiente. Trabajan en estrecha colaboración con personal de todas las categorías para averiguar los problemas que surjan en el sistema existente, y para cumplir con las expectativas del cliente a la hora de crear un nuevo sistema. Los analistas producen una especificación para un sistema que satisfaga las necesidades de la empresa.

Actividades laborales
Los analistas de sistemas utilizan tecnologías de la información y comunicación (TIC) para ayudar a las empresas a trabajar de forma más rápida y eficiente. Investigan un problema y luego diseñan o adaptan un sistema informático para mejorar el funcionamiento de la empresa.

Una vez la empresa ha aprobado el sistema, el analista comienza a trabajar en estrecha colaboración con los especialistas de TIC, diseñadores de sistemas y programadores para crear el sistema.

Cuando el proyecto está terminado, los analistas examinan cuidadosamente el nuevo sistema para asegurarse de que cumple con sus funciones y de que los usuarios están satisfechos con este.

Perfil profesional
Para ser analista de sistemas informáticos se necesita:
- Disfrutar a la hora de plantearse retos para la solución de problemas y de sopesar los pros y los contras de las diferentes soluciones.
- Capacidades lógicas, analíticas y de investigación, así como habilidades creativas.
- Conocimientos de informática y técnicas de programación.
- Fuertes habilidades de comunicación verbal y escrita.
- Saber escuchar y tener la capacidad de hacer las preguntas correctas.

Competencias
Analiza necesidades en software.
Capacidad de análisis.
Capaz de plantear preguntas con claridad.
Capaz de trabajar bajo presión.
Capaz de trabajar con vencimientos.
Conocimientos especializados en informática.
Creativo.
Destrezas en informática.
Diseña y adapta sistemas informáticos.
Habilidad para resolver problemas.
Planifica y realiza la introducción de nuevos sistemas.
Trabaja en equipo.

Diseñador de sistemas

Descripción:

El diseñador identifica y define las responsabilidades, operaciones, atributos y relaciones de los elementos de diseño. El diseñador se asegura de que el diseño sea coherente con la arquitectura de software, y que esté detallado hasta un punto en que pueda proceder la implementación.

Habilidades:

El diseñador debe tener conocimientos laborales sólidos de:

- requisitos del sistema
- la arquitectura del sistema
- técnicas de diseño de software, incluyendo técnicas de análisis y diseño orientado a objetos, y el Lenguaje unificado de modelado
- tecnologías con las que se implementará el sistema
- directrices de proyecto sobre cómo se relaciona el diseño con la implementación incluyendo el nivel de detalle esperado en el diseño antes de que proceda la implementación.
- Propuestas de asignación

A un diseñador se le puede asignar la responsabilidad de implementar una parte estructural del sistema (como un subsistema de implementación o de clases), o una parte funcional del sistema, como la ejecución de guiones de uso o sus características que cruza clases/subsistemas.

Programador
Descripción:
Los programadores informáticos escriben y prueban los programas de ordenador. Escriben las instrucciones en un lenguaje informático que el ordenador puede leer, para llevar a cabo tareas tales como el control de stock en un almacén o de registro de ventas. Los programadores desarrollan los pasos lógicos necesarios para crear un programa, lo prueban y almacenan los registros de forma segura, para poder adaptar los programas en el momento que se necesite.

Actividades laborales:
Los programadores informáticos escriben programas computacionales que dan instrucciones a un ordenador para que realice las tareas necesarias para almacenar la información introducida por los usuarios. Este trabajo permite, por ejemplo, controlar las acciones de la empresa, hacer cálculos salariales o mantener registros de personal.
Perfil profesional:
Para ser programador informático hay que tener las características siguientes:
- Tener conocimientos de programación.
- Ser analítico y lógico en el enfoque para la solución de problemas.
- Prestar atención a los detalles.
- Tener habilidades comunicativas y de trabajo en equipo.
- Concentrarse durante largos períodos de tiempo.
- Contar con habilidades de comunicación escrita para la compilación de informes y la elaboración de manuales.
- Administrar el tiempo de forma eficiente, priorizar tareas y trabajar bajo la presión de cumplir plazos determinados.
- Mantener registros exactos del trabajo realizado.
- Estar siempre al día sobre la evolución de los lenguajes de software y de programación, así como de las nuevas herramientas informáticas.
- Según el lugar de trabajo, se pueden requerir habilidades de negociación.

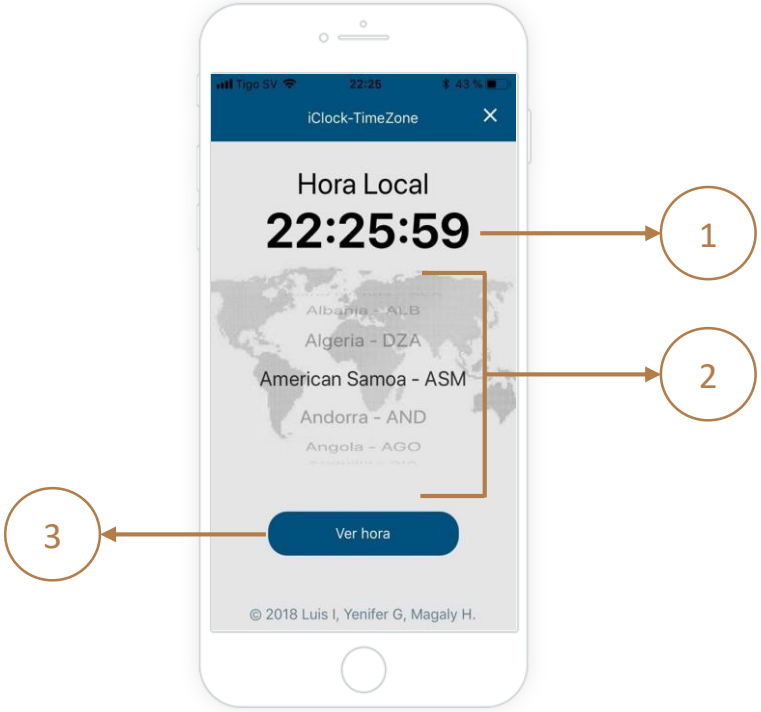
Competencias:
- Adapta programas existentes.
- Analiza necesidades en software.
- Aptitudes para gestionar el tiempo.
- Aptitudes para llevar registros.
- Capacidad de análisis.
- Capacidad para concentrarse.
- Capacidad para priorizar tareas.
- Capacidad para trabajar en equipo.
- Capaz de mantenerse al día de los avances tecnológicos.
- Capaz de prestar atención al detalle.
- Capaz de trabajar bajo presión.
- Capaz de trabajar con vencimientos.
- Corrige defectos de software.
- Destrezas en informática.
- Escribe y desarrolla programas informáticos.
- Habilidad para la programación.
- Habilidad para resolver problemas.
- Trabaja en equipo.
- Utiliza códigos, herramientas y lenguajes de programación.

4.4. Viabilidad Ambiental.

Entre los factores a considerar en la factibilidad ambiental de un proyecto están las características culturales, sociales, políticas, legales, históricas, territoriales y medio ambientales de la zona, y las restricciones que estas características traen consigo.

Cualquier proyecto o modelo a llevar a cabo debe tomar en cuenta todos esos factores, donde por supuesto, el medio ambiente tiene algo que decir, y a esto se le denomina factibilidad ambiental. Ya que es una aplicación pequeña y sola mostrará la hora de dicho país desde el celular, no se necesitará imprimir y hacer uso de papel ni de otro componente ambiental.

V. Manual de usuario



1. El objeto [1] muestra la hora local de donde se encuentra el dispositivo actualmente.

2. Se selecciona el país [2] de la lista al cual se requiere saber la hora.

3. Se oprime el botón [3] ver hora, este mostrara una alerta con la información de la fecha y hora del país seleccionado.

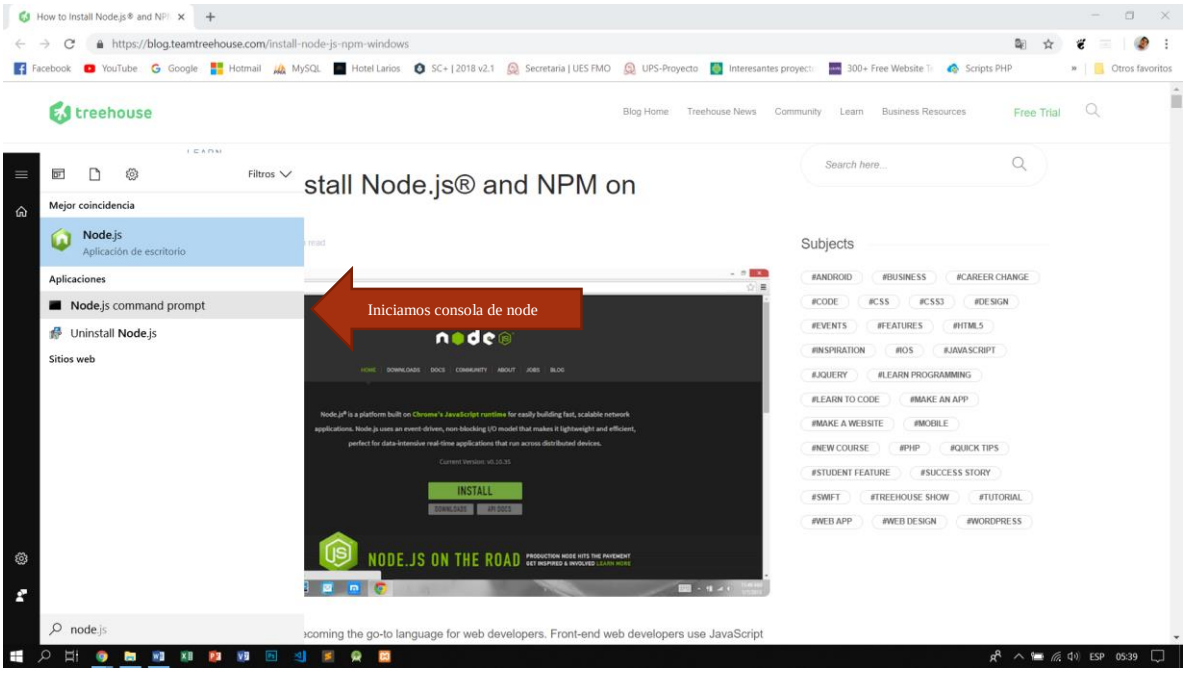
VI. Codificación de la aplicación

¿Qué es react native?

react native es un frameworks, es decir: un conjunto de herramientas para facilitar un fin, sabiendo esto hablemos brevemente de su historia, en el Facebook's internal hackathon project del 2013, se da a conocer la idea de una herramienta para crear aplicaciones híbridas, pero no es hasta marzo del 2015, que el proyecto queda como open source en Github.

Lista De Requerimientos

- Node.js https://nodejs.org/en/



Revisar la documentación de como emular la aplicación con Expo

https://medium.com/surabayadev/easy-react-native-with-expo-278e30803ee4

Para iniciar a tu proyecto solo necesitas instalar una herramienta, ejecuta:

npm install –g iClock

La codificación de nuestro proyecto utilizaremos los siguientes archivos

```
App.js

├──assets
│   icon.png
│   map.png
│
├──class
│   HoraExtrangera.js
│   HoraLocal.js
│   simpleFactory.js
│
└──data
    paises.js
```

Clase principal: App.js

```
import React, { Component } from 'react';
import { Text, BackHandler, StyleSheet, View, Picker, ActivityIndicator, Alert, ImageBackground } from 'react-native';
import { Header, Badge } from 'react-native-elements';
import { simpleFactory } from './class/simpleFactory';
import { Data } from './data/paises';

export default class App extends Component {
  //instancias a la clase simpleFactory
  hora = new simpleFactory();

  constructor(props) {
    super(props);
    this.state = {
      time: '',
      isLoading: true,
      dataSource: [],
      PickerValueHolder: ''
    }
  }
  componentDidMount() {
    //actualizacion del reloj principal
    this.Clock = setInterval(() => this.setState({ time: this.hora.simpleMethod(1), }), 1000);
  }
  componentWillMount() {
    //Local Data
    this.setState({
      dataSource: Data,
      isLoading: false
    });
    //API REST para traer informacion de los paises
    // fetch('https://restcountries.eu/rest/v2/all')
    // .then((response) => response.json())
    // .then((responseJson) => {
    //   this.setState({
    //     isLoading: false,
    //     dataSource: responseJson
    //   }, function () {
    //     // In this block you can do something with new state.
    //   });
    // })
    // .catch((error) => {
    //   console.error(error);
    // });
  }
  //metodo para alerta con la hora del pais seleccionado
  captureSelect = () => {
    if (this.state.PickerValueHolder != '') {
      this.hora.simpleMethod(2, this.state.PickerValueHolder)
      //this.hora.getTimeZone(this.state.PickerValueHolder);
    } else {
      Alert.alert("Error ", "Seleccione un pais", [{ text: 'Aceptar', },]);
    }

  }

  render() {

    if (this.state.isLoading) {
      return (
        <View style={{ justifyContent: 'center', flex: 1, paddingTop: 50 }}>
          <ActivityIndicator size='large' color="#0f6fc6" />
        </View>
      );
    }
```

```
    return (

     <View style={styles.MainContainer}>
      <ImageBackground source={require('./assets/map.jpg')} style={{ width: '100%', height: '100%' }} >
       <View style={{ flex: 0.1, }}>
        <Header backgroundColor='#00507d' style={{ height: 100 }}
         centerComponent={{ text: 'iClock-TimeZone', style: { color: '#fff', fontSize: 15 } }}
         rightComponent={{ onPress: () => BackHandler.exitApp(), icon: 'close', color: '#fff', }}
        />
       </View>
       <View style={{ margin: 30, flex: 0.8 }}>
        <View style={{ flex: 0.3, justifyContent: 'center', alignItems: 'center', }}>
         <Text style={{ textAlign: 'center', fontSize: 30, }} >Hora Local </Text>
         <Text style={styles.TextStyle}> {this.state.time} </Text>
        </View>
        <View style={{ flex: 0.6, }}>
         <Picker
          selectedValue={this.state.PickerValueHolder}
          style={{ width: '100%', height: 100 }}
          onValueChange={(itemValue, itemIndex) => this.setState({ PickerValueHolder: itemValue })} >
          <Picker.Item style={{ color: '#fff', }} label='Seleccione un pais' value='' />
          {this.state.dataSource.map((item, key) => (
           <Picker.Item label={item.name + ' - ' + item.alpha3Code} value={item.timezones[0].substr(3, 6) + item.alpha3Code}
key={key} />)
          )}
         </Picker>
        </View>
        <View style={{ flex: 0.1, alignItems: 'center' }}>
         <Badge containerStyle={styles.buttonHora} onPress={this.captureSelect}>
          <Text style={{ color: '#fff', fontSize: 15 }} >Ver hora</Text>
         </Badge>
        </View>
       </View>
       <View style={{ flex: 0.1 }}>

        <Header backgroundColor='#e5e5e5' style={{ height: 100 }}
         centerComponent={{ text: '© 2018 Luis I, Yenifer G, Magaly H.', style: { color: '#607d8b', fontSize: 15, paddingBottom: 10, }
}}
        />
       </View>
      </ImageBackground>
     </View>

    );
  }
}

const styles = StyleSheet.create({
  MainContainer: {
    flex: 1,
    backgroundColor: "#f8f8f8",
  },
  buttonHora: {
    backgroundColor: '#00507d',
    width: 200,
    height: 45,
  },
  TextStyle:
  {
    fontWeight: 'bold',
    fontSize: 50,
    textAlign: 'center',
    color: '#000',
    // marginBottom: 20,
  },
});
```

Clase simpleFactory.js

```
import { HoraLocal } from './HoraLocal';
import { HoraExtrangera } from './HoraExtrangera';

export class simpleFactory{
   simpleMethod(x,y){
      if(x==1)
      {
         horalocal = new HoraLocal();
         return horalocal.getTimeLocal();
      }
      else{
         horaExtrangera = new HoraExtrangera();
         return horaExtrangera.getTimeZone(y);
      }
   }
}
```

Clase HoraLocal.js

```
// Initializing a class Hora
export class HoraLocal{
  //method getTimeLocal
  getTimeLocal() {
    var date, hour, minutes, seconds, fullTime;
    date = new Date();
    hour = date.getHours();
    minutes = date.getMinutes();
    if (minutes < 10) {
      minutes = '0' + minutes.toString();
    }
    seconds = date.getSeconds();
    if (seconds < 10) {
      seconds = '0' + seconds.toString();
    }
    return fullTime = hour.toString() + ':' + minutes.toString() + ':' + seconds.toString();
  }
}
```

Clase HoraExtrangera.js

```
import { Alert } from 'react-native';

export class HoraExtrangera{
  //method getTimeLocal
  getTimeZone = (PickerValueHolder) => {
    var horas, minutos, offset, pais, d, utc, nd;
    // creamos el objeto Date (la selecciona de la máquina cliente)
    horas = PickerValueHolder.substr(0, 3).toString();
    minutos = PickerValueHolder.substr(4, 2).toString();
    if (minutos >= '30') {
      minutos = '50';
    }
    offset = horas + '.' + minutos;
    pais = PickerValueHolder.substr(6, 3).toString();
    //offset = '-03.00';
    d = new Date();
    // lo convierte  a milisegundos
    // añade la dirferencia horaria
    // recupera la hora en formato UTC
    utc = d.getTime() + (d.getTimezoneOffset() * 60000);
    // crea un nuevo objeto Date usando la diferencia dada.
    nd = new Date(utc + (3600000 * offset));
    // devuelve la hora como string.
    return Alert.alert("La fecha/hora actual en " + pais, nd.toLocaleString(), [{ text: 'Aceptar', },]);
  }
}
```

Lista con todos los países del mundo: Países.js

A continuación, se presenta un formato de cómo son creados cada uno de los países con la información de cada uno.

```
// Este archivo js conteniene a informacion de todos los paises del mundo

export const Data =
   [
  {
    name: "Afghanistan",
    topLevelDomain: [
      ".af"
    ],
    alpha2Code: "AF",
    alpha3Code: "AFG",
    callingCodes: [
      "93"
    ],
    capital: "Kabul",
    altSpellings: [
      "AF",
      "Afġānistān"
    ],
    region: "Asia",
    subregion: "Southern Asia",
    population: 27657145,
    latlng: [
      33,
      65
    ],
    demonym: "Afghan",
    area: 652230,
    gini: 27.8,
    timezones: [
      "UTC+04:30"
    ],
    borders: [
      "IRN",
      "PAK",
      "TKM",
      "UZB",
      "TJK",
      "CHN"
    ],
    nativeName: "افغانستان",
    numericCode: "004",
    currencies: [
      {
        code: "AFN",
        name: "Afghan afghani",
        symbol: "؋"
      }
    ],
    languages: [
      {
        iso639_1: "ps",
        iso639_2: "pus",
        name: "Pashto",
        nativeName: "پښتو"
      },
      {
        iso639_1: "uz",
        iso639_2: "uzb",
        name: "Uzbek",
        nativeName: "Oʻzbek"
      },
      {
        iso639_1: "tk",
        iso639_2: "tuk",
        name: "Turkmen",
        nativeName: "Türkmen"
      }
    ],
    translations: {
      de: "Afghanistan",
      es: "Afganistán",
      fr: "Afghanistan",
      ja: "アフガニスタン",
      it: "Afghanistan",
      br: "Afeganistão",
      ppt: "Afeganistão",
      nl: "Afghanistan",
      hrl: "Afganistan",
      fa: "افغانستان"
    },
    flag: "https://restcountries.eu/data/afg.svg",
    regionalBlocs: [
      {
        acronym: "SAARC",
        name: "South Asian Association for Regional Cooperation",
        otherAcronyms: [],
        otherNames: []
      }
    ],
    cioc: "AFG"
  },
]
```

VII. Conclusiones

En la actualidad las tendencias de desarrollo van encaminadas a los dispositivos móviles, el mayor tiempo del día lo dedicamos al celular, debido a ello nosotros como estudiantes de una carrea de ciencias y tecnologías nos vemos en la obligación de conocer a cerca de estas tecnologías, hacer un buen análisis nos puede facilitar el desarrollo del proyecto, ya que se emplean diferentes técnicas para que a los que les corresponda la construcción de la aplicación no tengan por qué estar pensando demasiado en la lógica de la aplicación que desarrollaran.

El desarrollo de aplicaciones móviles en específico para dispositivos IPhone IOS se vuelve tedioso por el motivo de que no se cuentan con las herramientas necesarias para la construcción de la aplicación, más sin embargo hay entornos de desarrollo que nos permite al menos emular nuestra aplicación para poder visualizarla, utilizar una tecnología como Ract Native nos permitirá el desarrollo multiplataforma nativo, que esto abarca los aspectos antes mencionados.

VIII. Anexos

Referencias bibliografías

E. KENDALL, K. y. (2005). *Análisis y diseño de sistemas. Sexta edición.* México: PEARSON EDUCACIÓN.

Loira, R. d. (2005). *Ingenieria del Software*. Madrid: PEARSON EDUCATION SA.

Presuman, R. S. (2010). *INGENIERÍA DEL SOFTWARE. UN ENFOQUE PRÁCTICO.* México: McGraw-Hill.

Presupuesto

A continuación, se detalla el presupuesto para el desarrollo del proyecto, los costos son financiados por cada uno de los estudiantes involucrados.

Descripción	Nombre	Precio
Entorno de desarrollo	React Native	$0.00
Computadora Mac	MacBook con Mac OS	$1629.00
Licencia de desarrollador Apple		$99.00

Nota: se necesita una Mac para el desarrollo en este tiempo de plataforma en IOS para poder compilar el proyecto esto conlleva un costo económico extra.

Cronograma

Actividades por día desde viernes 07 de septiembre de 2018 a viernes 21 de septiembre de 2018.

Descripción de actividad	1-2	3-4	5-6	7-8	9-10	11-12	13-14	15-16
Recolección de requerimientos								
Diseño rápido								
Construcción del prototipo								
Evaluación del prototipo por el cliente								
Refinanciamiento del prototipo								
Producto de ingeniería								