BeatIt!

Prototipo (Riesgos Técnicos)

Versión 1.2.1

Historial de versiones

Fecha	Versión	Descripción	Autor
06/09/2014	1.0		Luciana Martinez Cristian Bauza Joaquin Velazquez
06/09/2014	1.1	Riesgos Arquitectura	Gonzalo Javiel
07/09/2014	1.2	Revisión SQA y V&V	Emiliano Vázquez
13/09/2014	1.2.1	Revisión SQA	Pablo Olivera

Contenido

1. Riesgos asociados a la implementación de la Mobile App

1.1. Riesgo de Performance y Calidad de UI con Uso de Tecnologías para Desarrollo Mobile Multiplataforma.

Componente:

Aplicación Mobile

Requerimientos:

-Las interfaces gráficas de cada plataforma deben tener los estilos de cada una.
-El tiempo que transcurre entre que un jugador realiza una acción y tiene la posibilidad de hacer otra acción, no debe superar los 3 segundos.

Resultado:

Dado los requerimientos del cliente mencionados en el punto anterior, y de la investigación de las tecnologías multiplataforma de uso libre encontradas, surge que las mismas utilizan una mezcla de javascript, css y html para crear la aplicación, lo que genera que las aplicaciones sean más lentas que con un desarrollo nativo.
Por otra parte el cliente nos comunico que en años anteriores otros grupos habían usado tecnologías multiplataforma y las interfaces gráficas de las aplicaciones creadas no quedaban con un nivel de calidad muy bueno, basicamente se veían iguales en las distintas plataformas.
De todos modos instalamos el ambiente de desarrollo de appcelerator titanium, en el cual generamos alguna pequeña prueba del estilo "hola mundo" percatando que no se trabajaba con código nativo de la plataforma, jugando como una contra a la hora de querer trabajar a un nivel más detallado de implementación.

1.2. Riesgo de Dificultad de Compresión en Uso de API GPS

Componente:

Componente encargado de implementar el desafío Usain Bolt

Requerimiento:

Desafío Usain Bolt

Resultado:

Ante las pruebas realizadas se obtuvo comprensión del servicio GPS brindado por las dos plataformas, además si validó los desafíos planteados y relacionados.
Lecturas de referencias:

- http://mobileosgeek.com/how-to-get-the-current-geo-coordinates-in-windows-phone-using-c/
- http://www.androidhive.info/2012/07/android-gps-location-manager-tutorial/

1.3. Riesgo de Dificultad de Comprensión en Uso de API Acelerómetro

Componente:

Componente encargado de implementar el desafío Bouncing game!

Requerimiento:

Desafío Bouncing game!

Resultado:

Se realizó una prueba mediante la implementación de un juego que consiste en mover una esfera en la pantalla del celular, dicho movimiento surge de la toma de datos proporcionadas del acelerómetro del celular. Dicha prueba fue satisfactoria ya que se logró el objetivo deseado al plantearse la prueba, el cual primero era lograr obtener los datos del acelerómetro y en segundo lugar interpretarlos correctamente para lograr que cuando (por ejemplo) balanceamos el celular a la derecha, la esfera en la pantalla fuera a la derecha.
Lecturas de referencias:

- http://msdn.microsoft.com/es-es/library/ff604984(v=xnagamestudio.40).aspx
- http://www.maestrosdelweb.com/editorial/curso-android-sensores-trabajar-con-acelerometro/

1.4. Riesgo de Soporte y Comprensión para Reconocimiento Facial.

Componentes:

Clases encargadas de implementar desafío “Selfie Groupie” o “Sacate una selfie”

Requerimiento:

Desafio Selfie Groupe

Resultado:

El resultado fue positivo ya que se encontró un webservices para el reconocimiento facial. A su vez se obtuvo la ventaja de que un webservice soporta la dos plataformas.

También se realizó para Android una prueba para el reconocimiento facial se haga localmente y, a pesar de que no se consiguió el resultado esperado debido al tamaño de la imagen obtenida por la cámara, se concluyó que es posible realizar, en Android, el reconocimiento facial localmente ya que se consiguió una librería que lo realiza.
Lectura de referencia:

- https://www.mashape.com/apicloud/facerect

1.5. Riesgo de Soporte Multiplataforma y Comprensión de APIs para Reconocimiento de Objetos.

Requerimiento:

Desafío "Es una botella de Coca-Cola?"

Resultado:

La prueba se dividió en dos partes:

1. Comprender como capturar imagenes desde los dispositivos.
2. Investigar APIs locales o webservices para el reconocimiento de objetos.

La primera parte se realizó con éxito, en base a documentación disponible en internet no surgieron problemas.
Para la segunda se consiguieron APIs de reconocimientos pero implicaba limitaciones de presupuesto, y para las libres requería un nivel de implementación más elevado por lo que implicaba un riesgo de curva de aprendizaje más prolongado para validar este desafío.

1.6. Riesgo de Soporte Multiplataforma y Comprensión de APIs para Reconocimiento de Sonidos.

Requerimiento:

Desafíos "Alcanzame una nota !" y "Seguime el ritmo"

Resultado:

La prueba se dividió en dos partes:

1. Investigación de APIs locales o webservices para el procesamiento de sonidos.
2. Comprender cómo capturar sonidos desde los dispositivos para luego procesarlos.

Para el primer punto no tuvimos éxito dado que no se encontraron en internet APIs o webservices gratis que realizará el procesamiento de sonido tal como lo necesitábamos. Debido a esto nos propusimos investigar como hacerlo nosotros mismos (punto 2).

Según lo investigado en internet para el procesamiento de sonido lo que se hace es grabar el sonido por medio de la clase AudioRecord que utiliza el micrófono del dispositivo. Luego se realiza un estudio del archivo grabado. El archivo da las frecuencias del sonido capturado por el micrófono, por medio de las cuales se debe distinguir a qué notas pertenecen.

Dado que para llevar a cabo el desafío se necesitaba tener conocimiento teórico de sonido, se requiere una curva de aprendizaje demasiado alta para el tiempo que tenemos se decidió descartar estos dos desafíos.

1.7 Riesgo de Soporte en Android y Comprensión de APIs para el envío de mensajes de texto.

Requerimiento:

Desafío "Exprésate!"

Resultado:

Se realizó una pequeña aplicación en la cual se le pide al usuario que escoja un contacto de su agenda para enviarle un mensaje predeterminado por la aplicación.
El resultado fue positivo ya que se encontró una librería que permite el acceso a los contactos del usuario (y obtener el número telefónico) y otra librería con la cual es posible enviar mensajes de texto.
Se encontró una forma de realizar el envío pasando por la aplicación de mensajes del teléfono, pudiéndose editar el mensaje y el contacto elegido, y también sin pasar por la misma.

2. Riesgos Arquitectura

2.1. Riesgo Tecnología WebServer

Componente:

Servidor Web

Requerimientos:

- Utilizar la plataforma en la nube Microsoft Azure.
- El tiempo que transcurre entre que un jugador realiza una acción y tiene la posibilidad de hacer otra acción, no debe superar los 3 segundos.

Resultado:

Se investigaron distintas opciones para la implementación del servidor web, concluyendo que NodeJS era teóricamente útil y sencillo para la lógica que pretendemos implementar en el servidor, dado ese escenario se procedió a realizar las pruebas con este servidor en Azure para comprobarlo prácticamente.

Se probaron dos instalaciones distintas de este servidor, primero utilizando una Máquina Virtual con Centos alojado en Azure y luego utilizando el servicio Web Sites.

La instalación fue muy sencilla teniendo un "Hola Mundo" rápidamente en ambas instalaciones.

Código del servidor

```
var http = require('http')
var port = process.env.PORT || 1337;
http.createServer(function(req, res) {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello World PIS');
}).listen(port);
```

El sitio de pruebas utilizado fue http://trynodejs.azurewebsites.net/ creando por nosotros.

Posteriormente se vinculo un repositorio de github a la instalación en el servicio WebSites, esto nos brinda la posibilidad de actualizar automáticamente el servidor en cada "push" del repositorio, lo que acelera el desarrollo en lo que respecta al servidor.

Se realizo dicha prueba siguiendo el tutorial http://azure.microsoft.com/en-us/documentation/articles/web-sites-nodejs-develop-deploy-mac/

Código del servidor

```
var http = require('http')
var port = process.env.PORT || 1337;
http.createServer(function(req, res) {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello World PIS, probando deploy desde git.........\n');
}).listen(port);
```

2.2. Riesgo Tecnología Base de Datos

Componente:

DataStorage

Requerimientos:

- Utilizar la plataforma en la nube Microsoft Azure.
- El tiempo que transcurre entre que un jugador realiza una acción y tiene la posibilidad de hacer otra acción, no debe superar los 3 segundos.

Resultado:

Se optó por probar una base de datos NoSQL luego de investigar la bondades de la misma en cuanto a los tiempos de respuesta, lo "natural" de la conexión con un servidor NodeJS y como es la definición del modelo.

Se utilizó un tutorial donde se presenta la conexión entre un Web Site de Azure con NodeJS y MongoDB
http://azure.microsoft.com/en-us/documentation/articles/store-mongolab-web-sites-nodejs-store-data-mongodb/

Definición de un modelo en NodeJS para MongoDB

```
var mongoose = require('mongoose')
  , Schema = mongoose.Schema;

var TaskSchema = new Schema({
          itemName            : String
        , itemCategory        : String
        , itemCompleted       : { type: Boolean, default: false }
        , itemDate            : { type: Date, default: Date.now }
});

module.exports = mongoose.model('TaskModel', TaskSchema);
```

2.3. Riesgo Tecnología Servicios Rest

Componente:

BeatIt! API

Requerimientos:

- Utilizar la plataforma en la nube Microsoft Azure.
- El tiempo que transcurre entre que un jugador realiza una acción y tiene la posibilidad de hacer otra acción, no debe superar los 3 segundos.

Resultado:

Se investigaron distintas opciones para presentar los servicios desde el servidor a los clientes, entre los mismos podemos mencionar algunas como WFC, ASP.Net Web API, SOAP, REST

Se optó por útilizar una API RESTfull, debido a la extensa documentación en lo que respecta a la interacción con dispositivos móviles, la velocidad de respuesta por ser una implementación liviana en comparación a las otras tecnologías y por interoperabilidad con las otras herramientas elegidas.

En este caso las pruebas no fueron tan sencillas, dado que existen varias maneras de implementar estos servicios y mucha de la documentación en la web utiliza versiones ya no soportadas. De igual manera se tuvo éxito utilizando como base el tutorial y modificandolo http://coenraets.org/blog/2012/10/creating-a-rest-api-using-node-js-express-and-mongodb/

Fracción de código que "publica" los servicios y los asocia a funciones que los implementa

```
app.get('/wines', wine.findAll);
app.get('/wines/:id', wine.findById);
app.post('/wines', wine.addWine);
app.put('/wines/:id', wine.updateWine);
app.delete('/wines/:id', wine.deleteWine);
```

Fracción de código que se conecta a una base MongoDB y carga datos en la misma

```
var mongo = require('mongodb');

var Server = mongo.Server,
    Db = mongo.Db,
    BSON = mongo.BSONPure;

var server = new Server('localhost', 27017, {auto_reconnect: true});
db = new Db('winedb', server);

db.open(function(err, db) {
    if(!err) {
        console.log("Connected to 'winedb' database");
        db.collection('wines', {strict:true}, function(err, collection) {
            if (err) {
                console.log("The 'wines' collection doesn't exist. Creating it with sample
data...");
                populateDB();
            }
        });
```

```
    }
});
```

Fracción de código que muestra la implementación de uno de los servicios listados anteriormente

```
exports.findById = function(req, res) {
    var id = req.params.id;
    console.log('Retrieving wine: ' + id);
    db.collection('wines', function(err, collection) {
        collection.findOne({'_id':new BSON.ObjectID(id)}, function(err, item) {
            res.send(item);
        });
    });
};
```