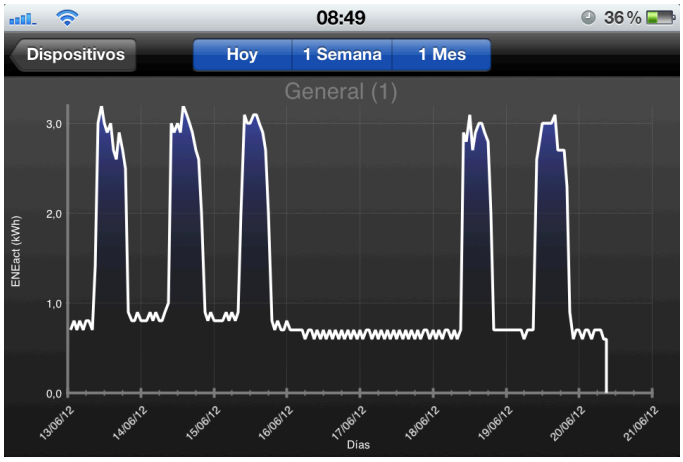
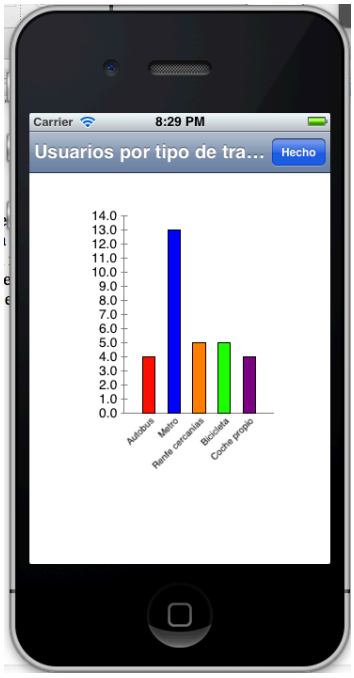
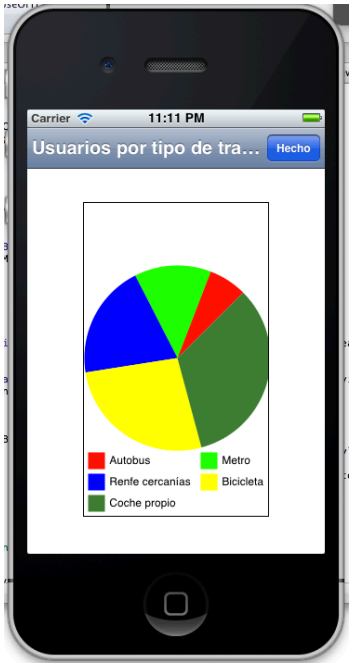
INTRODUCCIÓN A CORE_PLOT





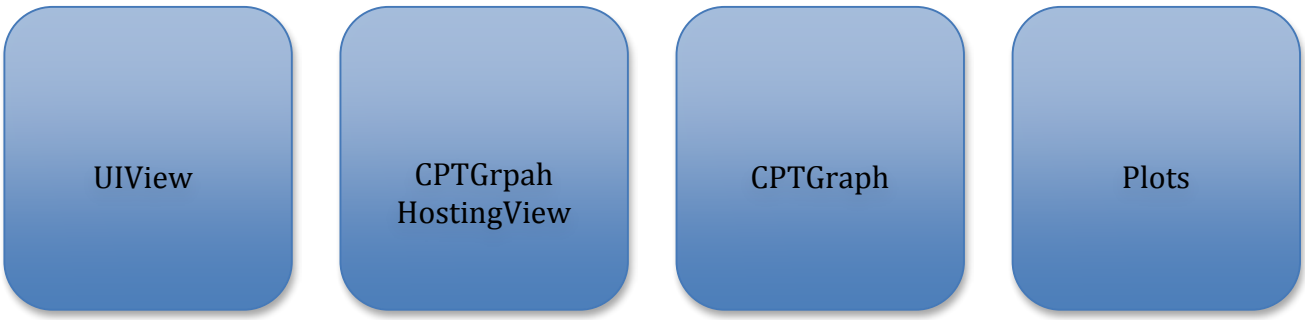


La visualización de los datos es fundamental (si el volumen de datos es grande) para ayudar al usuario a dar sentido a los datos y tomar decisiones importantes . CorePlot es una biblioteca de terceros que le permite mostrar una serie de gráficos interactivos en la aplicación. Esto incluye los gráficos de líneas, gráficos de dispersión, gráficos de barras, e incluso los gráficos circulares. Usted puede personalizar el aspecto de los gráficos e incluso permiten al usuario interactuar con ellos.

Que vamos a ver

- Cómo agregar CorePlot a una aplicación.
- Los fundamentos de la CorePlot y cómo conseguir un entorno de trabajo gráfico.
- Creación de un gráfico lineal.
- Aplicar estilos gráficos.
- Creación de un gráfico de barras.
- Creación de un gráfico circular.

ELEMENTOS BÁSICOS DE UN GRÁFICO DE CORE PLOT



El **CPTGraphHostingView** es simplemente una UIView, la cual es responsable de contener el **CPTGraph** y permitir al usuario interactual con él, lo veremos ahora.

El **CPTGraph** es una clase abstracta la cual es responsable de dibujar los elemntos del gráfico y gestionar los distintos "plots". Además es el responsable de aplicar temas, recargar los datos del gráfico y más……Puede contener uno o más plots.

EL **Plot**, es el responsable de representar los datos, cada tipo de "Plot", tiene su propio DataSource y éste, provee métodos específicos en función del tipo de Plot, el Plot puede ser:

- CPTScatterPlot. Lineal o de dispersión.
- CPTBarPlot. De barras
- CPTPieChart. Circular.

La aplicación de ejemplo

En el código fuente se encuentra la base de la aplicación que vamos a utilizar. Se trata de una aplicación que alamcena el tipo de trnasporte utilizado por los usuarios dependiendo del día de la semana.

Abre el proyecto y asegurarse de que se ejecuta. No estoy usando ARC en esta demo.

NOTA: Junto con este documento hay 2 carpetas de código fuente, una de ellas con la aplicación inicial para seguir el tutorial "Starter_Code" , y otra con el tutorial completado "Finished Project". En esta habrá que añadir el framwork **Core**_Plot en la carpeta "**ExternalLibs**" antes de ejecutarla, no se incluye por el gran tamaño de la misma

Importación de CorePlot

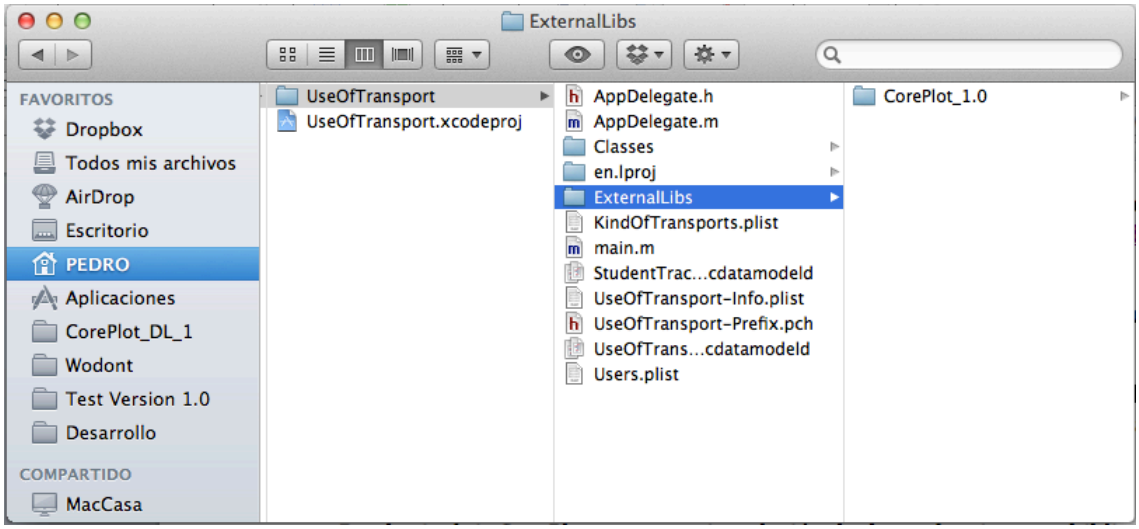
El primer paso es descargar la última versión de CorePlot. En el momento de escribir eeste docuemnto es v1.0. Visita la página principal home page y ves downloads section.

Descargue el archivo zip. La biblioteca ocupa de alrededor de 140 mb lo que puede tardar un tiempo si tienes conexión lenta.

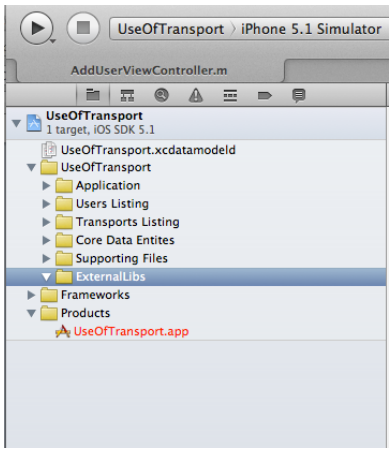
Vamos a poner los archivos de la Biblioteca CorePlot dentro de nuestra aplicación. Es una buena práctica para almacenar las libreías de terceros en una carpeta separada de los archivos de la aplicación. Crea una nueva carpeta en la carpeta "UseOfTransport" y llámala "ExternalLibs".

Puedes incluir CorePlot como una instalación de dependencia o una biblioteca estática. Vamos a hacerlo como una instalación dependiente.

1. Copie el archivo "CorePlot" en el directorio en la recién creada " ExternalLibs " carpeta dentro de la carpeta de origen UseOfTransport.

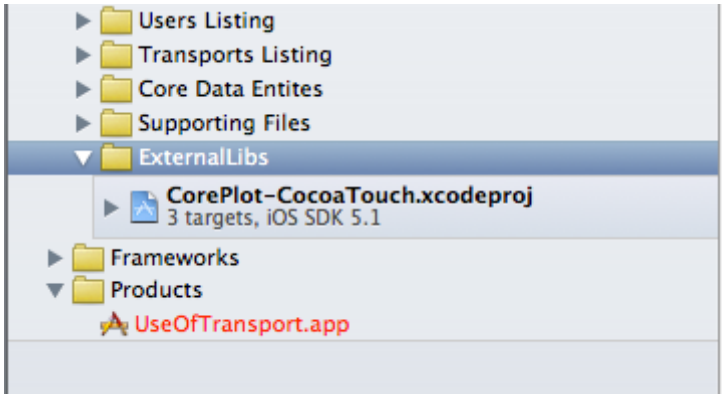


2. Crea un grupo llamada igual (“ExternalLibs”)



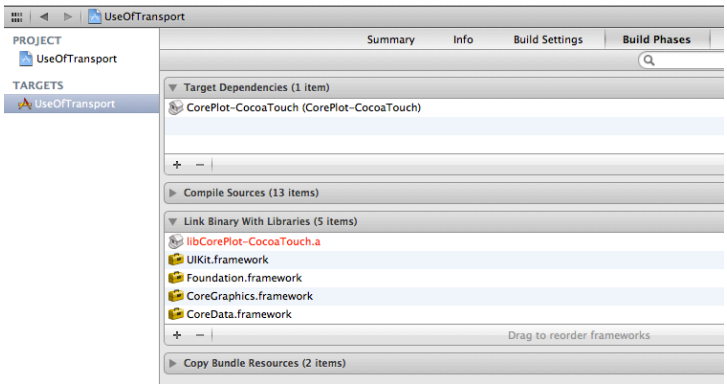
3. En Finder, localiza el proyecto “CocoaTouch.xcodeproj” (“CorePlot_1.0/Source/frameworkCocoaTouch.xcodeproj”) y arástralo hasta el grupo “ExternalLibs “ en Xcode.

4.

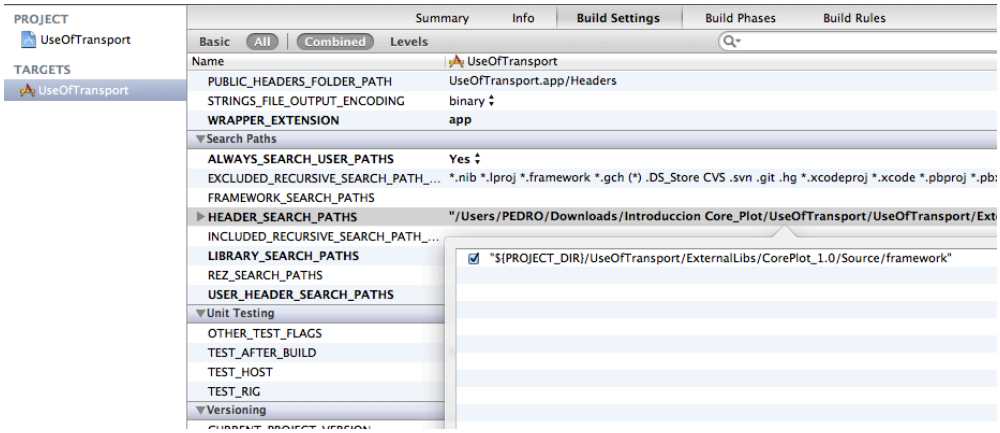


5. Selecciona el archivo de proyecto de aplicación en Xcode y haz click en "Build Phases" . A es necesario agregar la Biblioteca CorePlot-CocoaTouch en el "Target dependencies" del grupo.

6. También tendrás añadir biblioteca CorePlot. Expande "Link Binary Libraries" y añade la librería "libCorePlot-CocoaTouch.a"



5. Establece la ruta de búsqueda de cabeceras en el directorio de origen CorePlot (debe ser "${PROJECT_DIR}/UseOfTransport/ExternalLibs/CorePlot_1.0/Source/framework"). Marca la casilla situada a la derecha del texto (para indicar recursividad). También debes cambiar "Always Search User Paths" a "Sí".



6 A continuación, añade "-ObjC" a la opción "others_ldflags" .

7 CorePlot se basa en el framework QuartzCore, por lo que es necesario añadirlo al proyecto.

8 Por último, tendrás que importar el archivo "CorePlot-cocoaTouch.h" en la unidades que lo requieran. Para probrar que la integración de core-plot la hemos realizado de forma corrrecta, añadiremos “CorePlot-cocoaTouch.h” a “AppDelegate.h”

Si el proyecto se compila satisfactoriamente, entonces puedes continuar.

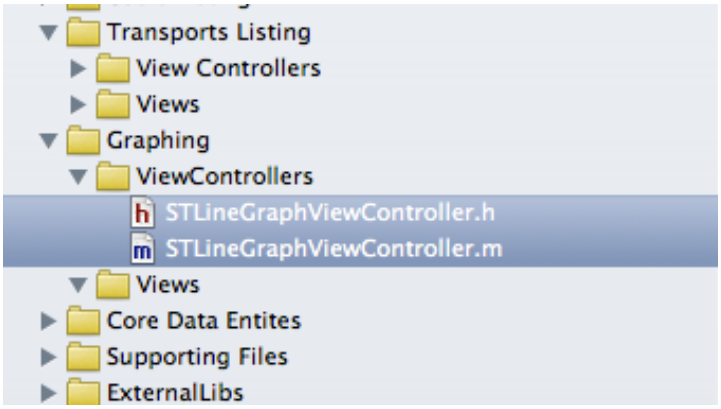
TRACK _2

Paso 1. Configurando.

Para habilitar la presentación de gráficos añadiremos un botón en la barra del navegador de la pantalla de usuarios.

Vamos a crear un nuevo grupo llamado "Graphing" . Crea un grupo Views y otro “View Controllers” dentro de este, como está en “Transports Listing”.

Crea una nueva clase llamada 'STLineGraphViewController' (subclase de UIViewController) en el grupo de los “View Controllers".



Ahora vamos a añadir el botón que permite al usuario seleccionar un gráfico para ser mostrado.

En primer lugar abre TransportListViewController.h y añade el protocolo 'UIActionSheetDelegate' y 'STLineGraphViewControllerDelegate’ , este protocolo no existe todavía, pero vamos a crearlo más tarde (También asegúrate de importar el "STLineGraphViewController.h”).

```
@interface UserListViewController : UIViewController <UITableViewDelegate,
UITableViewDataSource, AddUserViewControllerDelegate,
UIActionSheetDelegate, STLineGraphViewControllerDelegate>
```

Luego abre el fichero .m e implementa el método 'actionSheet: clickedButtonAtIndex:' con el siguiente código:

```
#pragma mark - UIActionSheetDelegateM ethods

-(void)actionSheet:(UIActionSheet *)actionSheet
clickedButtonAtIndex:(NSInteger)buttonIndex
{
    if (buttonIndex == 0)
{
        STLineGraphViewController *lineGraphVC = [[STLineGraphViewController
alloc] init];
        [lineGraphVC
setModalTransitionStyle:UIModalTransitionStyleFlipHorizontal];
        [lineGraphVC setDelegate:self];
        [lineGraphVC setManagedObjectContext:[self managedObjectContext]];

        [self presentModalViewController:lineGraphVC animated:YES];
}
}
```

Este código no necesita demasiada explicación. Simplemente estamos creando un controlador de LineGraph y presentarlo de forma modal. Creamos un delegado para que podamos saber cuándo hacer desaparecer la vista modal. También damos al view controller un 'managed object context' con lo que podremos interactuar con los datos.

A continuación vamos a crear un método para llamar al "action sheet" y añadir un UITabBarItem para llamarlo. Agrega un método llamado GraphButtonWasSelected: ':

```
@interface UserListViewController ()
@property (nonatomic, strong) NSArray *userArray;

- (void)addUser:(id)sender;
- (void)graphButtonWasSelected:(id)sender;

@end
```

A continuación, agrega su implementación:

```
- (void)graphButtonWasSelected:(id)sender
{
    UIActionSheet *graphSelectionActionSheet = [[[UIActionSheet alloc]
initWithTitle:@"Escoge un gráfico" delegate:self
cancelButtonTitle:@"Cancelar" destructiveButtonTitle:nil
otherButtonTitles:@"Uso diario de transporte", nil] autorelease];

    [graphSelectionActionSheet showInView:[[UIApplication
sharedApplication] keyWindow]];
}
```

Ahora necesitamos añadir UIBarButtonItem para que el usuario lo pulse cuando quiera ver un gráfico. Lo crearemos en el método ViewDidLoad justo debajo de donde se crea el rightBarButton.

```
[[self navigationItem] setLeftBarButtonItem:[[[UIBarButtonItem alloc]
initWithTitle:@"Graphs" style:UIBarButtonItemStylePlain target:self
action:@selector(graphButtonWasSelected:)] autorelease] animated:NO];
```

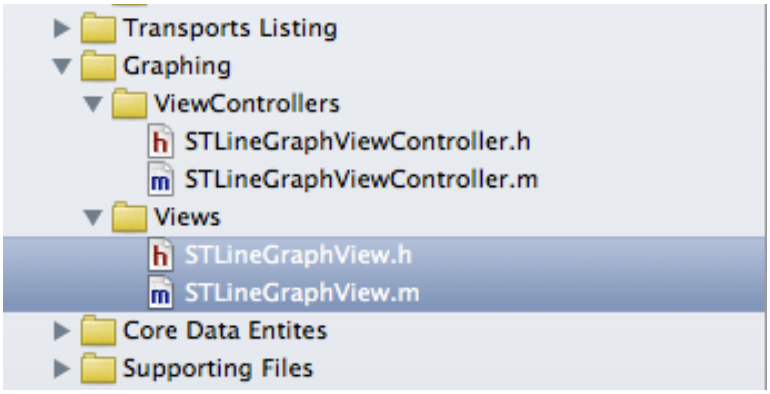
Finalmente necesitaremos implementar el protocolo de STLineGraphViewController que hará desaparecer la vista modal del gráfico.

```
- (void)doneButtonWasTapped:(id)sender
{
    [self dismissModalViewControllerAnimated:YES];
}
```

Ya está todo listo para crear el gráfico.

Paso.2 Creando el gráfico

Primero necesitamos crear un "custom view" para nuestro "LineGraphViewController". Crea una UIView llamada "STLineGraphView"".



Vamos a parametrizar la parte gráfica de la vista que hemos creado. Primero en el fichero .h importamos la unidad ""CorePlot-CocoaTouch.h" y añadimos la siguiente propiedad:

```
@property (nonatomic, strong) CPTGraphHostingView *chartHostingView;
```

El CPTGraphHostingView es simplemente una UIView, la cual es responsable de contener el gráfico y permitir al usuario interactual con él, lo veremos ahora.

Sintetiza chartHostingView y crea el `chartHostingView en el` initWithFrame método.

```
@synthesize chartHostingView = _chartHostingView;

- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        [self setChartHostingView:[[[CPTGraphHostingView alloc]
initWithFrame:CGRectZero] autorelease]];
        [self addSubview:_chartHostingView];
    }
    return self;
}
```

Hemos creado un CPTGraphHostingView y aasignado como nuestro chartHostingVIew. Entonces lo hemos añadido como subview.

Ahora necesitamos indicar las dimensiones de la vista del grafíco en el 'layout subviews' método.

```
- (void)layoutSubviews
{
    [super layoutSubviews];

    float chartHeight = self.frame.size.height - 40;
    float chartWidth = self.frame.size.width;

    [[self chartHostingView] setFrame:CGRectMake(0, 0, chartWidth,
chartHeight)];
    [[self chartHostingView] setCenter:[self center]];
}
```

La mayoría del trabajo ahora lo realizaremos en STLineGraphViewController. Abre STLineGraphViewController.h and añade las siguientes declaraciones:

```
#import "CorePlot-CocoaTouch.h"

@interface STLineGraphViewController : UIViewController
<CPTScatterPlotDataSource, CPTScatterPlotDelegate>

@property (nonatomic, strong) CPTGraph *graph;
@property (nonatomic, assign) id<STLineGraphViewControllerDelegate>
delegate;
@property (nonatomic, strong) NSManagedObjectContext
*managedObjectContext;
```

El CPTGraph es una clase abstracta la cual es responsable de dibujar los elemntos del gráfico y gestionar los distintos "plots". Además es el responsable de aplicar temas, recargar los datos del gráfico y más…… Hemos indicado que vamos a implementar los protocolos CPTScatterPlotDataSource y CPTScatterPlotDelegate.

Además necesitamos nuestro propio protocolo para hacer desaparecer la vista modal. Añade el siguiente código encima de la declaración “interface”:

```
@protocol STLineGraphViewControllerDelegate
@required
- (void)doneButtonWasTapped:(id)sender;

@end
```

Ya en el fichero .m, sintetiza las propiedades, luego añade el siguiente código antes del método ‘viewDidLoad’:

```
- (void)loadView
{
    [super loadView];
    [self setTitle:@"Uso de transporte"];
    [self setView:[[[STLineGraphView alloc] initWithFrame:self.view.frame]
autorelease]];

    CPTTheme *defaultTheme = [CPTTheme themeNamed:kCPTPlainWhiteTheme];
    [self setGraph:(CPTGraph *)[defaultTheme newGraph]];
}
```

En primer lugar, estamos creando y estableciendo la vista del controller como nuestro STLineGraphView. A continuación, creamos un 'CPTTheme'. El CPTTheme gestiona el estilo de un gráfico: los estilos de línea, estilo de texto, y cualquier relleno que se requiera. Una manera fácil de obtener un CPTGraph preconfigurado con una base de CPTTheme es crear el CPTTheme con uno de los nombres temáticos predefinidos y entonces con 'newGraph " obtener un gráfico con el estilo predefinido.

A continuación, vamos a poner el siguiente código en el método 'viewDidLoad':

```
[super viewDidLoad];
    STLineGraphView *graphView = (STLineGraphView *)[self view];
    [[self graph] setDelegate:self];

    [[graphView chartHostingView] setHostedGraph:[self graph]];

    CPTScatterPlot *transportScatterPlot = [[CPTScatterPlot alloc]
initWithFrame:[graph bounds]];
    [transportScatterPlot setIdentifier:@"studentEnrollment"];
    [transportScatterPlot setDelegate:self];
    [transportScatterPlot setDataSource:self];

    [[self graph] addPlot: transportScatterPlot];

    UINavigationItem *navigationItem = [[UINavigationItem alloc]
```

```
initWithTitle:self.title];
    [navigationItem setHidesBackButton:YES];

    UINavigationBar *navigationBar = [[[UINavigationBar alloc]
initWithFrame:CGRectMake(0, 0, self.view.frame.size.width, 44.0f)]
autorelease];
    [navigationBar pushNavigationItem:navigationItem animated:NO];

    [self.view addSubview:navigationBar];

    [navigationItem setRightBarButtonItem:[[[UIBarButtonItem alloc]
initWithTitle:@"Done" style:UIBarButtonItemStyleDone target:[self
delegate]

action:@selector(doneButtonWasTapped:)] autorelease] animated:NO];
```

En el código anterior estamos configurando nuestra vista y el contenedor del gráfico para nuestro gráfico. Entonces creamos un "plot" para poner en el gráfico. Usamos el 'CPTScatterPlot "cuando queremos crear un gráfico de líneas. El identificador es algo que podemos usar para identificar la “Plot” más adelante. A continuación, establecemos el delegado y el origen de datos para el controller, ya que será responsable de proporcionar los datos para el gráfico. Por último, añadimos “Plot” creado al gráfico.

Después se crea un navigation item y bar para la vista modal y poder mostrar un título y el botón que nos permitirá volver a la vista anterior.

Ahora añadiremos datos

Los Plots en CorePlot usa principalmente 2 métodos del datasource para obtener los datos

‘numberOfRecordsForPlot’ y ‘numberForPlot:field:recordIndex:’. Es parecido a a como trabajan las “TableViews”. Primero especificamos el número de registros para el “Plot”:

```
#pragma mark - CPTScatterPlotDataSource Methods

- (NSUInteger)numberOfRecordsForPlot:(CPTPlot *)plot
{
    return 8;
}
```

Estamos mostrando cuantos usuarios usan un determinado tipo de transporte en cada uno de los días de la semana. Como hay 7 posibles días tenemos 8 registros en total ya que empezamos en 0.

Ahora queremos especificar el valor para la x e y de cada registro.

```
- (NSNumber *)numberForPlot:(CPTPlot *)plot field:(NSUInteger)fieldEnum
recordIndex:(NSUInteger)index
{
    NSUInteger x = index;
    NSUInteger y = 0;

    NSError *error;
```

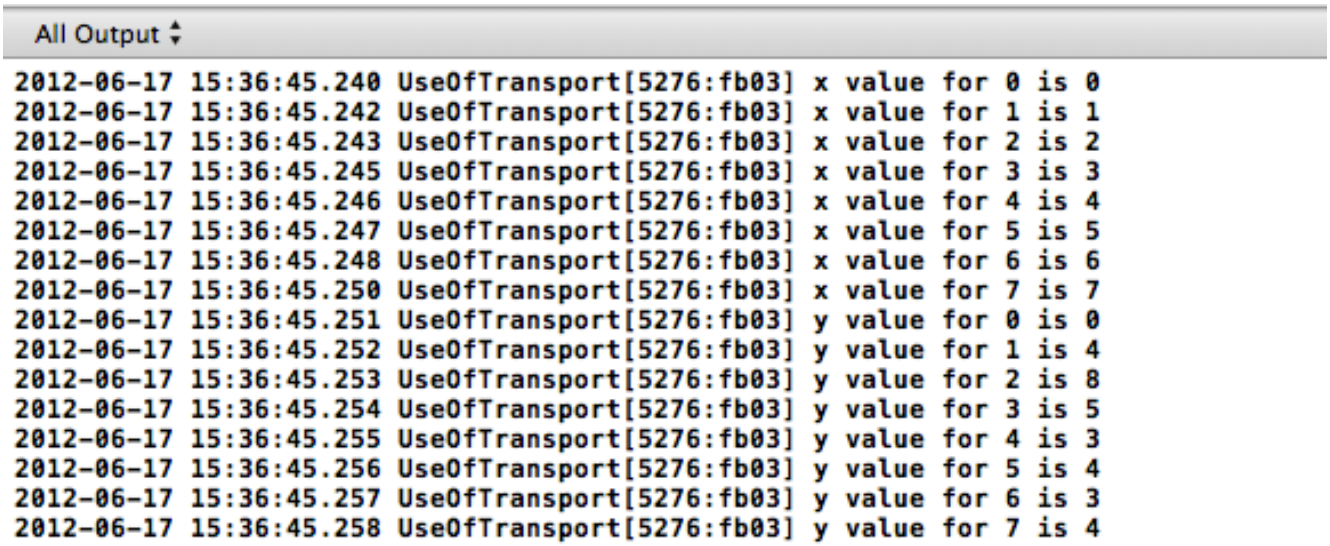
```
    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
    NSEntityDescription *entity = [NSEntityDescription
entityForName:@"User" inManagedObjectContext:managedObjectContext];
    NSPredicate *predicate = [NSPredicate predicateWithFormat:@"dayOfWeek
== %d", index];

    [fetchRequest setEntity:entity];
    [fetchRequest setPredicate:predicate];

    y = [managedObjectContext countForFetchRequest:fetchRequest
error:&error];

    [fetchRequest release];

    switch (fieldEnum)
    {
        case CPTScatterPlotFieldX:
            NSLog(@"x value for %d is %d", index, x);
            return [NSNumber numberWithInt:x];
            break;
        case CPTScatterPlotFieldY:
            NSLog(@"y value for %d is %d", index, y);
            return [NSNumber numberWithInt:y];
            break;

        default:
            break;
    }

    return nil;
}
```

Este método debe especificar un valor de x e y para un determinado índice y el valor que se devuelve se basa en 'fieldEnum' (que en nuestro caso es CPTScatterPlotFieldX o CPTScatterPlotFieldY). El parámetro “Plot” indica para cual debemos indicar el valor si tuviéramos varios en el gráfico.

CorePlot reuqiere que los valores se le devuelvan como NSNumber.

El valor de x es fácil de determinar. Como lo que muestra son los días, x es igual al índice actual. El valor y va a ser un recuento de todos los usuarios en ese día. Podemos obtener esto haciendo una consulta a la base de datos para obtener todos los registros User con un "dayOfweek' del índice.

Si guardas y ejecutas la aplicación ahora, todavía no vas ver nada en el gráfico, pero verás la siguiente salida en la consola:



Esto significa que el gráfico obtiene los valores correctos para X e Y (asegúreate de que es el mismo o similar a la salida de la imagen. Todavía no se muestra en el gráfico. Eso es porque, si nos fijamos en el gráfico, el rango que se muestra es incorrecto. estamos buscando de -1,0 a 0 en ambos ejes X e Y. Tenemos que establecer el rango para que podamos ver los puntos de datos.

El 'CPTXYPlotSpace determina cómo se vé el gráfico y cómo es formatea.

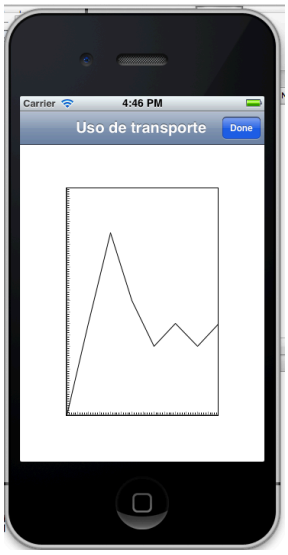
Para establecer el rango de x e y que el usuario ve, tenemos que trabajar con el objeto 'CPTXYPlotSpace'. Este objeto nos permite establecer un rango visible para el gráfico.

En el mètodo “viewDidLoad” añade el siguiente código justo debajo de donde hemos añadido el “Plot” a nuestro gráfico:

```
CPTXYPlotSpace * userPlotSpace = (CPTXYPlotSpace *)[graph
defaultPlotSpace];
    [userPlotSpace setXRange:[CPTPlotRange
plotRangeWithLocation:CPTDecimalFromInt(0) length:CPTDecimalFromInt(7)]];
    [userPlotSpace setYRange:[CPTPlotRange
plotRangeWithLocation:CPTDecimalFromInt(0) length:CPTDecimalFromInt(10)]];
```

Primero accedemos al `CPTXYPlotSpace` por defecto del gráfico. Entonces añadimos los rangos para x e y. El rango es un “CPTPlotRange” que creamos estáticamente usando el método “plotRangeWithLocation:length:”. Este método recibe NSDecimals y CorePlot posee una función que podemos usar para obtener un valor decimal a partir de un entero, la función es “CPTDecimalFromInt”, también hay una función para valores flota si fuera necesario.

Ahora ya podremos ver el inicio de lo que será nuestro gráfico:



TRACK 3

Ahora vamos a ver cómo hacer que el gráfico sea más útil para el usuario mediante la especificación de incrementos de los ejes. Vamos a ver diferentes maneras en que puede personalizar la apariencia del gráfico. Además veremos la posibilidad de trabjar con más de un "Plot" en el mismo gráfico.

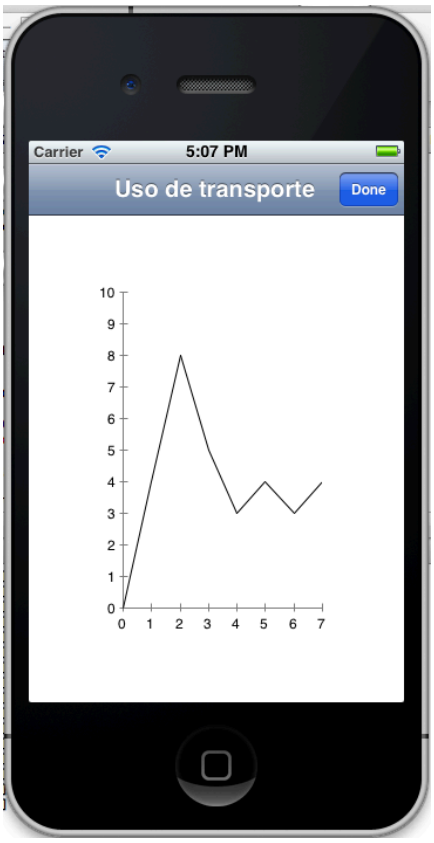
Paso 1: Configuración de incrementos del eje
Para modificar las propiedades de un eje X e Y se trabaja con el 'CPTXYAxisSet' y CPTXAxis. Abre el archivo STLineGraphViewController.m y continúa con el método viewDidLoad. Justo debajo de donde trabajamos con "CPTXYPlotSpace" introduzca el siguiente código:

```
[[graph plotAreaFrame] setPaddingLeft:20.0f];
[[graph plotAreaFrame] setPaddingTop:10.0f];
[[graph plotAreaFrame] setPaddingBottom:20.0f];
[[graph plotAreaFrame] setPaddingRight:10.0f];
[[graph plotAreaFrame] setBorderLineStyle:nil];

NSNumberFormatter *axisFormatter = [[NSNumberFormatter alloc] init];
[axisFormatter setMinimumIntegerDigits:1];
[axisFormatter setMaximumFractionDigits:0];

CPTMutableTextStyle *textStyle = [CPTMutableTextStyle textStyle];
[textStyle setFontSize:12.0f];

CPTXYAxisSet *axisSet = (CPTXYAxisSet *)[graph axisSet];

CPTXYAxis *xAxis = [axisSet xAxis];
[xAxis setMajorIntervalLength:CPTDecimalFromInt(1)];
[xAxis setMinorTickLineStyle:nil];
[xAxis setLabelingPolicy:CPTAxisLabelingPolicyFixedInterval];
```

```
    [xAxis setLabelTextStyle:textStyle];
    [xAxis setLabelFormatter:axisFormatter];

    CPTXYAxis *yAxis = [axisSet yAxis];
    [yAxis setMajorIntervalLength:CPTDecimalFromInt(1)];
    [yAxis setMinorTickLineStyle:nil];
    [yAxis setLabelingPolicy:CPTAxisLabelingPolicyFixedInterval];
    [yAxis setLabelTextStyle:textStyle];
    [yAxis setLabelFormatter:axisFormatter];
```

Vamos a repasar todo lo anterior. En primer lugar, trabajmos con la propiedad "plotAreaFrame" 'de "CPTGraph". Con esto estamos en condiciones de establecer el espacio donde está realmente el gráfico representado y nos permite ver las etiquetas de los ejes (que antes estaban ocultas). A continuación, establecemos el estilo del borde para hacer desaparecer la línea que lo encerraba Creamos un NSNumber que utilizamos para dar formato a las etiquetas de los ejes. También creamos algo que se llama un "CPTMutableTextStyle '. Al aplicar formato a las líneas, rellenar una sección y texto, utilizamos objetos como CPTMutableTextStyle para hacerlo. Nosotros solo ajustaremos el tamaño de la fuente, pero se puede establecer el tipo de fuente y el color también.

Obtenemos el CPTXYAxisSet de nuestro gráfico. Este contiene un axisSet ejeX y un yaxis (los dos objetos de tipo 'CPTXYAxis'). A continuación, establecemos una serie de propiedades en cada eje. La longitud del mayor intervalo, lo que establece el intervalo en cada tick principal. Queremos eliminar los ticks intermedios por lo que establece el estilo de línea a cero. Hemos establecido el labellingPolicy a intervalos fijos. A continuación, establecemos el estilo de texto para el CPTMutableTextStyle que hemos creado antes.



Cambiando el look al gráfico

Primero cambiaremos la línea. Debajo de la líenas donde hemos configurado los ejes, introduce el siguiente código:

```
[mainPlotLineStyle setLineWidth:2.0f];
    [mainPlotLineStyle setLineColor:[CPTColor colorWithCGColor:[[UIColor
blueColor] CGColor]]];

[transportScatterPlot setDataLineStyle:mainPlotLineStyle];
```

Esto hará que la línea de nuestro gráfico se muestre azul y que aumente su grosor. Si eres más creativo puedes hacer que el color sea m menos rígido, pero ten encuenta que CorePlot requiere que el color sea expresado como CPTColor. Aunque no podemos obetener un CPTColor a partir de un UIColor, si que lo podemos obtener a partir de un CGColor.

Además podemos cambiar el estilo de los ejes, introduce el siguiente código debajo de donde hemos indicado el "dataLineStyle" del "Plot".

```
CPTMutableLineStyle *axisLineStyle = [CPTMutableLineStyle lineStyle];
    [axisLineStyle setLineWidth:1];
    [axisLineStyle setLineColor:[CPTColor colorWithCGColor:[[UIColor
grayColor] CGColor]]];

    [xAxis setAxisLineStyle:axisLineStyle];
    [xAxis setMajorTickLineStyle:axisLineStyle];
    [yAxis setAxisLineStyle:axisLineStyle];
    [yAxis setMajorTickLineStyle:axisLineStyle];
```

Esto indica el estilo para los ticks principales de ambos ejes. Además puedes seleccionar el estilo de texto como gris si quieres.

Puedes añadir una gradiente de relleno a la zona entre la linea del gráfico y los ejes, para hacer esto crea un "CPTFill", el cual es asignable al Plot:

```
CPTColor *areaColor = [CPTColor blueColor];
    CPTGradient *areaGradient = [CPTGradient
gradientWithBeginningColor:areaColor endingColor:[CPTColor clearColor]];
    [areaGradient setAngle:-90.0f];
    CPTFill *areaGradientFill = [CPTFill fillWithGradient:areaGradient];
    [transportScatterPlot setAreaFill:areaGradientFill];

[transportScatterPlot setAreaBaseValue:CPTDecimalFromInt(0)];
```

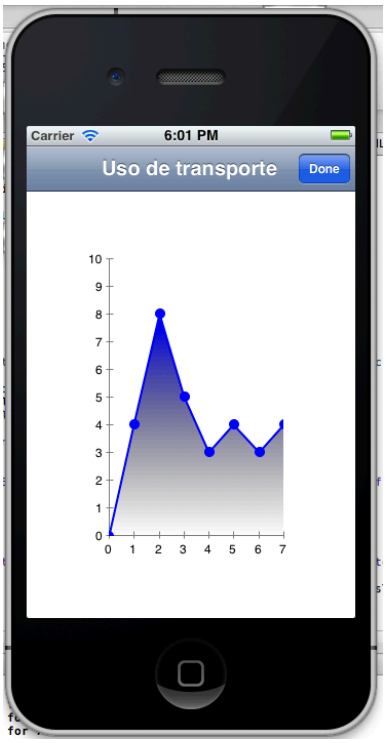
Esto crea un relleno para la zona entre la linea del gráfico y los ejes, en este caso el gradiente em pieza con axul oscuro hasta llegr a ser transparente. El ángulo indica la dirección del gradiente y el área don empieza el ardiente. Como queremos empezar en el inicio del gráfico hemos indicado el valor 0.

Por último, a veces es una buena idea tener los puntos a partir de los cuáles se trazan las líneas del gráfico. Para ello tenemos que utilizar un método del DataSource de "CPTScatterPlot" llamado 'symbolForScatterPlot: recordIndex':

```
- (CPTPlotSymbol *)symbolForScatterPlot:(CPTScatterPlot *)aPlot
recordIndex:(NSUInteger)index
{
    CPTPlotSymbol *plotSymbol = [CPTPlotSymbol ellipsePlotSymbol];
    [plotSymbol setSize:CGSizeMake(10, 10)];
    [plotSymbol setFill:[CPTFill fillWithColor:[CPTColor blueColor]]];
    [plotSymbol setLineStyle:nil];
    [aPlot setPlotSymbol:plotSymbol];

    return plotSymbol;
}
```

El código de arriba crea y devuelve un objeto CPTPlotSymbol. Podemos hacer que parezca todo tipo de cosas, pero nuestro gráfico utilizará una elipse (círculo) de color azul con un tamaño de 10 por 10.

Ahora tu gráfico será más o menos como este::



Varios Plot en el mismo gráfico.

Estamos mostrando los usuarios que viajan a lo largo de la semana, pero si queremos ver el uso de un tipo de trnasporte en el mismo gráfico?

Un CPTGraph puede contener varios “Plots”. Creamos uno nuevo “Plot” igual que hemos hecho antes y lo añadimos al granfico.. Gracias a los métodos del DataSource podemos identificar de forma única cada uno de los “Plot” y de esa manera mostrar los datos correctos.

Vamos a crear un Plot que muestre el uso del metro. Debajo del código donde se crea el “`userPlotSpace`“(en el método viewDidLoad), agrega el siguiente:

```
CPTScatterPlot *metroScatterPlot = [[CPTScatterPlot alloc]
initWithFrame:[graph bounds]];
    [metroScatterPlot setIdentifier:@"useOfMetro"];
    [metroScatterPlot setDelegate:self];
    [metroScatterPlot setDataSource:self];

    [[self graph] addPlot:transportScatterPlot];
    [[self graph] addPlot:metroScatterPlot];
```

Vamos a asignarle algunas propiedades de estilo. Debajo de donde indicamos el “dataLineStyle” for the “`userPlotSpace`”, añade el siguiento código:

```
[transportScatterPlot setDataLineStyle:mainPlotLineStyle];

    [mainPlotLineStyle setLineColor:[CPTColor greenColor]];
    [metroScatterPlot setDataLineStyle:mainPlotLineStyle];
```

Debajo de donde indicamos el relleno del “Plot” userTransport añade el siguiente código:

```
areaColor = [CPTColor greenColor];
    areaGradient = [CPTGradient gradientWithBeginningColor:areaColor
endingColor:[CPTColor clearColor]];
    [areaGradient setAngle:-90.0f];
    areaGradientFill = [CPTFill fillWithGradient:areaGradient];
    [metroScatterPlot setAreaFill:areaGradientFill];
    [metroScatterPlot setAreaBaseValue:CPTDecimalFromInt(0)];
```

Básicamente hemos repetido lo mismo que para el primer gráfico . Ahora debemos modificar los métodos del “DataSource” del gráfico para puedan devolver los datos de forma correcta para cada uno de los “Plots”. Realmente solo tenemos que modificar el método “numberForPlot: field: recordIndex:” ya que los días de la semana son los mismos. Localiza donde creamos el predicado y modificalo para que quede así:

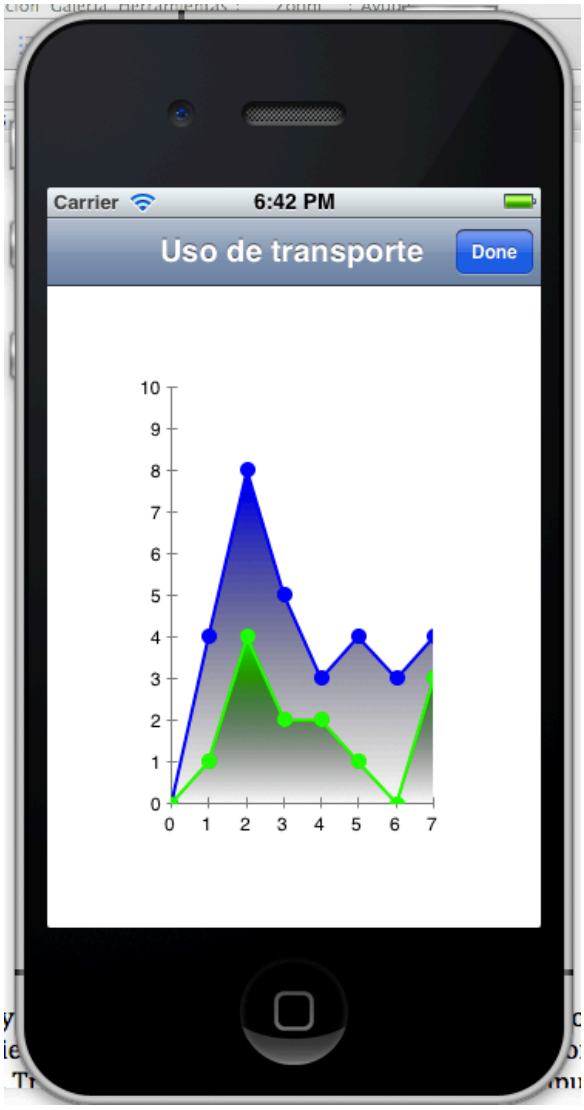
```
NSPredicate *predicate = nil;

NSPredicate *predicate = nil;
```

```
    if ([[plot identifier] isEqual:@"userTravel"])
    {
        predicate = [NSPredicate predicateWithFormat:@"dayOfWeek == %d",
index];
    }
    else if ([[plot identifier] isEqual:@"useOfMetro"])
    {
        predicate = [NSPredicate predicateWithFormat:@"dayOfWeek == %d AND
transportID == %d", index, 1];  // Coidgo del transporte Metro
    }
```

Esto construye el predicado basándose en el "Plot" que está solicitando los datos, ya solo nos queda identificar los puntos del gráfico:

```
- (CPTPlotSymbol *)symbolForScatterPlot:(CPTScatterPlot *)aPlot
recordIndex:(NSUInteger)index
{
    CPTPlotSymbol *plotSymbol = [CPTPlotSymbol ellipsePlotSymbol];
    [plotSymbol setSize:CGSizeMake(10, 10)];

    if ([[aPlot identifier] isEqual:@"userTravel"])
    {
        [plotSymbol setFill:[CPTFill fillWithColor:[CPTColor blueColor]]];
    }
    else if ([[aPlot identifier] isEqual:@"useOfMetro"])
    {   [plotSymbol setFill:[CPTFill fillWithColor:[CPTColor
greenColor]]];
    }

    [plotSymbol setLineStyle:nil];
    [aPlot setPlotSymbol:plotSymbol];

    return plotSymbol;
}
```

Nuestro gráfico ahora debería ser algo como:

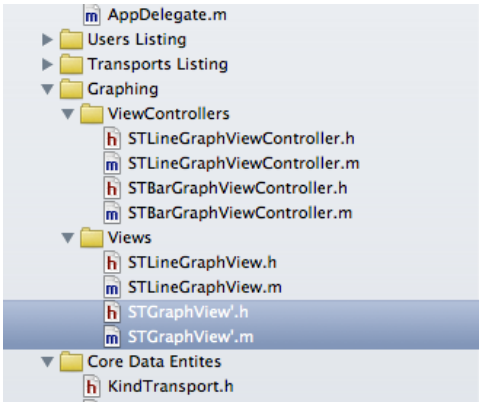


TRACK 4

Ahora veremos la creación de un gráfico de barras que muestra el número total de usuarios de cada tipo de trnasporte, cada barra representará una tipo de transporte. También veremos cómo personalizar la apariencia de la gráfica.

Paso 1

En primer lugar tenemos que añadir las clases correspondientes a nuestro proyecto. Vamos a crear una ViewController llamado 'STBarGraphViewController "y un" STGraphView'.



Necesitamos añadir un nuevo botón al "Action sheet" para lanzar el nuevo gráfico . Pero antes de nada vamos a abrir " UserListViewController.h" e importaremos STBarGraphViewController. Añadir STBarGraphViewControllerDelegate a la lista de protocolos registrados (el protocolo de hecho no existe todavía, pero vamos a crear más adelante):

```
@interface UserListViewController : UIViewController <UITableViewDelegate,
UITableViewDataSource, AddUserViewControllerDelegate,
UIActionSheetDelegate, STLineGraphViewControllerDelegate,
STBarGraphViewControllerDelegate>
```

A continuación en el fichero .m localiza el metodo 'graphButtonWasSelected:'. Añade 'Usuarios por tipo de trnasporte" a la lista de 'otherButtonTitles:':

```
UIActionSheet *graphSelectionActionSheet = [[[UIActionSheet alloc]
initWithTitle:@"Escoge un gráfico" delegate:self
cancelButtonTitle:@"Cancela" destructiveButtonTitle:nil
otherButtonTitles:@"Uso diario de transporte", @"Usuarios por tipo de
trnasporte", nil] autorelease];
```

Localiza el método 'actionSheet:clickedButtonAtIndex:' y modifícalo para que contemple el buttonIndex 1:

```
-(void)actionSheet:(UIActionSheet *)actionSheet
clickedButtonAtIndex:(NSInteger)buttonIndex
{
    if (buttonIndex == 0)
    {
        STLineGraphViewController *graphVC = [[STLineGraphViewController
alloc] init];
        [graphVC
setModalTransitionStyle:UIModalTransitionStyleFlipHorizontal];
        [graphVC setModalPresentationStyle:UIModalPresentationFullScreen];
        [graphVC setDelegate:self];
        [graphVC setManagedObjectContext:[self managedObjectContext]];

        [self presentModalViewController:graphVC animated:YES];

      //  [graphVC release];

    }
    else if (buttonIndex == 1)
    {
        STBarGraphViewController *graphVC = [[STBarGraphViewController
alloc] init];
        [graphVC
setModalTransitionStyle:UIModalTransitionStyleFlipHorizontal];
        [graphVC setModalPresentationStyle:UIModalPresentationFullScreen];
        [graphVC setDelegate:self];
        [graphVC setManagedObjectContext:[self managedObjectContext]];

        [self presentModalViewController:graphVC animated:YES];

        [graphVC release];
    }
}
```

Esto nos mostrará algunos warnings porque no hemos implementado el “delegate” ni las propiedades managedObjectContext y STBarGraphViewController todavía: Vamos a solucionarlo, para ello ves a STBarGraphViewController.h. importa CorePlot-CocoaTouch.h y añade la siguiente declaración:

```
@protocol STBarGraphViewControllerDelegate
@required
- (void)doneButtonWasTapped:(id)sender;

@end
```

Y las properties:

```
@property (nonatomic, strong) CPTGraph *graph;
@property (nonatomic, assign) id<STBarGraphViewControllerDelegate>
```

```
delegate;
@property (nonatomic, strong) NSManagedObjectContext
*managedObjectContext;
```

Finalmente indicaremos que la clase implementa los siguientes prococlos:

```
@interface STBarGraphViewController : UIViewController
<CPTBarPlotDelegate, CPTBarPlotDataSource>
```

Ahora en el fichero .m sintetiza las propiedades y libéralas en el método dealloc:

```
@synthesize delegate;
@synthesize managedObjectContext;
@synthesize graph;
```

Ahora vamos a añadir la vista al controlador, importa las clase “STBarGraphView.h” y añade el siguiente método:

```
- (void)loadView
{
    [super loadView];

    [self setTitle:@"Usuarios por tipo de transporte"];
    [self setView:[[[STGraphView alloc] initWithFrame:self.view.frame]
autorelease]];

    CPTTheme *defaultTheme = [CPTTheme themeNamed:kCPTPlainWhiteTheme];

    [self setGraph:(CPTGraph *)[defaultTheme newGraph]];
}
```

Ahor ya podemos trabajar en la vista STGraphView.h, importa (CorePlot-CocoaTouch.h), y añade las siguientes propiedad:

```
@property (nonatomic, strong) CPTGraphHostingView *chartHostingView;
```

En el fichero .m, sintetiza la propiedad y libérala en el método dealloc, and release the property in the dealloc. Entonces crea el CPTGraphHostingView en el método ‘initWithFrame:’ method:

```
- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self)
    {

        [self setChartHostingView:[[[CPTGraphHostingView alloc]
initWithFrame:CGRectZero] autorelease]];
        [chartHostingView setBackgroundColor:[UIColor purpleColor]];
```

```
        [self addSubview:chartHostingView];

    }
    return self;
}
```

Finalmente crea el método “layoutSubviews” con el siguiente código:

```
- (void)layoutSubviews
{
    [super layoutSubviews];

    float chartHeight = self.frame.size.height;
    float chartWidth = self.frame.size.width;

    [[self chartHostingView] setFrame:CGRectMake(0, 0, chartWidth,
chartHeight)];
    [[self chartHostingView] setCenter:[self center]];
}
```

Elcódigo es el mismo que que el del gráfico lineal, por lo que podríamos usar esta clase como base para todos los gráficos, se duplica con el fin de reforzar las partes de un gráfico creado con Core_Plot.

Ya en la unidad “STBarGraphViewController.m” en el método “ViewDisLoad”, vamos a crear nuestro “Plot” y añadírselo a nuestro gráfico.

```
STGraphView *graphView = (STGraphView *)[self view];
    [[self graph] setDelegate:self];

    [[graphView chartHostingView] setHostedGraph:[self graph]];

    CPTBarPlot *subjectBarPlot = [[CPTBarPlot alloc] initWithFrame:[graph
bounds]];
    [subjectBarPlot setIdentifier:@"UssTransportDayly"];
    [subjectBarPlot setDelegate:self];
    [subjectBarPlot setDataSource:self];

    [[self graph] addPlot:subjectBarPlot];
```

Añadiremos un navigation con un “Hecho” botón, para que el usuario pueda volver atrás:

```
UINavigationItem *navigationItem = [[[UINavigationItem alloc]
initWithTitle:self.title] autorelease];
    [navigationItem setHidesBackButton:YES];

    UINavigationBar *navigationBar = [[[UINavigationBar alloc]
initWithFrame:CGRectMake(0, 0, self.view.frame.size.width, 44.0f)]
autorelease];
    [navigationBar pushNavigationItem:navigationItem animated:NO];

    [self.view addSubview:navigationBar];

    [navigationItem setRightBarButtonItem:[[[UIBarButtonItem alloc]
initWithTitle:@"Hecho" style:UIBarButtonItemStyleDone target:[self
delegate] action:@selector(doneButtonWasTapped:)] autorelease]
animated:NO];
```

Paso 2. Datos del gráfico

El proceso va a ser similar a la manera en que hemos creado el gráfico de líneas, pero vamos a hacer las cosas un poco más dinámicas. Vamos a crear métodos personalizados que proporcionan un espacio para el “Plot” adeacuado. También vamos a añadir algunos métodos adicionales del “DataSource” para ofrecer un color específico para cada barra, así como los títulos de eje-x.

Primero vamos a configurar el eje y los rangos visibles. Para ello necesitamos dos métodos que calculan la x máxima y los valores máximos y. Declara los métodos siguientes en la parte superior del archivo m, a continuación del último “import”:

```
@interface STBarGraphViewController ()

- (float)getTotalTransports;
- (float)getMaxUsers;

@end
```

Ahora implementalos, como sigue:

```
#pragma mark - Private Methods

- (float)getTotalTransports
{
    NSError *error = nil;
    NSFetchRequest *fetchRequest = [[[NSFetchRequest alloc] init]
autorelease];
    NSEntityDescription *entity = [NSEntityDescription
entityForName:@"KindTransport"
inManagedObjectContext:managedObjectContext];
    [fetchRequest setEntity:entity];

    return [managedObjectContext countForFetchRequest:fetchRequest
error:&error];
}

- (float)getMaxUsers
{
    float maxEnrolled = 0;

    NSError *error = nil;

    for (int i = 0; i < [self getTotalTransports]; i++)
    {
        NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
        NSEntityDescription *entity = [NSEntityDescription
entityForName:@"User" inManagedObjectContext:managedObjectContext];

        NSPredicate *predicate = [NSPredicate
predicateWithFormat:@"transportID == %d", i];
        [fetchRequest setEntity:entity];
        [fetchRequest setPredicate:predicate];

        float subjectMax = [managedObjectContext
countForFetchRequest:fetchRequest error:&error];
        NSLog(@"users for Transport %d is %f", i, subjectMax);
        [fetchRequest release];

        if (subjectMax > maxEnrolled)
        {
            maxEnrolled = subjectMax;
        }
    }

    return maxEnrolled;
}
```

`getTotalTransports` simplemente devuelve un recuento de todos los tipos de transporte en la base de datos, `getMaxUsers` recorre todos los tipos de transporte y busca el mayor número de usuarios en cada uno de ellos y devuelve el valor más alto.

Con estos métodos, podemos volver al método 'viewDidLoad' y agregar el siguiente código debajo de donde hemos añadido el Plot al gráfico:

```
CPTXYPlotSpace *userPlotSpace = (CPTXYPlotSpace *)[graph
defaultPlotSpace];
    [userPlotSpace setXRange:[CPTPlotRange
plotRangeWithLocation:CPTDecimalFromInt(0) length:CPTDecimalFromInt([self
getTotalTransports] + 1)]];
    [userPlotSpace setYRange:[CPTPlotRange
plotRangeWithLocation:CPTDecimalFromInt(0) length:CPTDecimalFromInt([self
getMaxUsers] + 1)]];

    [[graph plotAreaFrame] setPaddingLeft:40.0f];
    [[graph plotAreaFrame] setPaddingTop:10.0f];
    [[graph plotAreaFrame] setPaddingBottom:120.0f];
    [[graph plotAreaFrame] setPaddingRight:0.0f];
    [[graph plotAreaFrame] setBorderLineStyle:nil];

    CPTMutableTextStyle *textStyle = [CPTMutableTextStyle textStyle];
    [textStyle setFontSize:12.0f];
    [textStyle setColor:[CPTColor colorWithCGColor:[[UIColor grayColor]
CGColor]]];

    CPTXYAxisSet *axisSet = (CPTXYAxisSet *)[graph axisSet];

    CPTXYAxis *xAxis = [axisSet xAxis];
    [xAxis setMajorIntervalLength:CPTDecimalFromInt(1)];
    [xAxis setMinorTickLineStyle:nil];
    [xAxis setLabelingPolicy:CPTAxisLabelingPolicyFixedInterval];
    [xAxis setLabelTextStyle:textStyle];

    CPTXYAxis *yAxis = [axisSet yAxis];
    [yAxis setMajorIntervalLength:CPTDecimalFromInt(1)];
    [yAxis setMinorTickLineStyle:nil];
    [yAxis setLabelingPolicy:CPTAxisLabelingPolicyFixedInterval];
```

Ahora necesitamos añadir datos al gráfico, mediante los métodos del “DataSource”, Calcular el nº de registros es fácil:

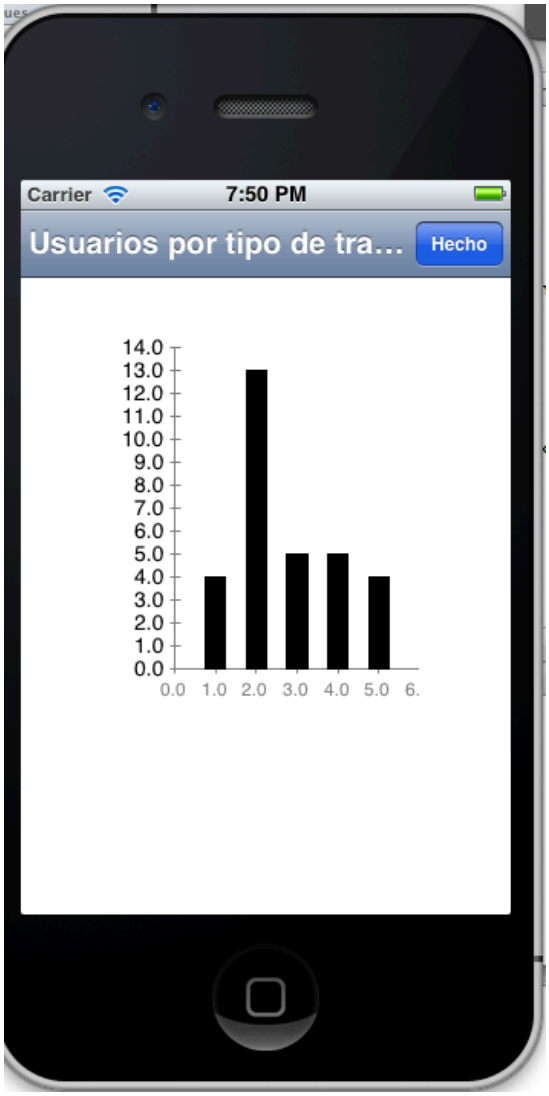
```
#pragma mark - CPTBarPlotDataSource Methods

- (NSUInteger)numberOfRecordsForPlot:(CPTPlot *)plot
{
    return [self getTotalTransports];
}
```

Ahora los valores para lax x e y:

```
- (NSNumber *)numberForPlot:(CPTPlot *)plot field:(NSUInteger)fieldEnum
recordIndex:(NSUInteger)index
{
    int x = index + 1;
    int y = 0;

    NSError *error;

    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
    NSEntityDescription *entity = [NSEntityDescription
entityForName:@"User" inManagedObjectContext:managedObjectContext];

    NSPredicate *predicate = [NSPredicate
predicateWithFormat:@"transportID == %d", index];

    [fetchRequest setEntity:entity];
    [fetchRequest setPredicate:predicate];

    y = [managedObjectContext countForFetchRequest:fetchRequest
error:&error];

    [fetchRequest release];

    switch (fieldEnum)
    {
        case CPTScatterPlotFieldX:
            return [NSNumber numberWithInt:x];
            break;
        case CPTScatterPlotFieldY:
            return [NSNumber numberWithInt:y];
            break;

        default:
            break;
    }

    return nil;
}
```

El método anterior es muy similar a cómo proporcionar datos al gráfico lineal. Cambiamos la a la base de datos por lo que tenemos un recuento de todos los usuarios que usan un determinado tipo de trnasporte. Además asignamos el valor a x +1. Esto porque la barra comienza en el '1 ' y proporciona un espacio entre la primera barra y el eje y.

Ya deberíamos poder ver el gráfico……!

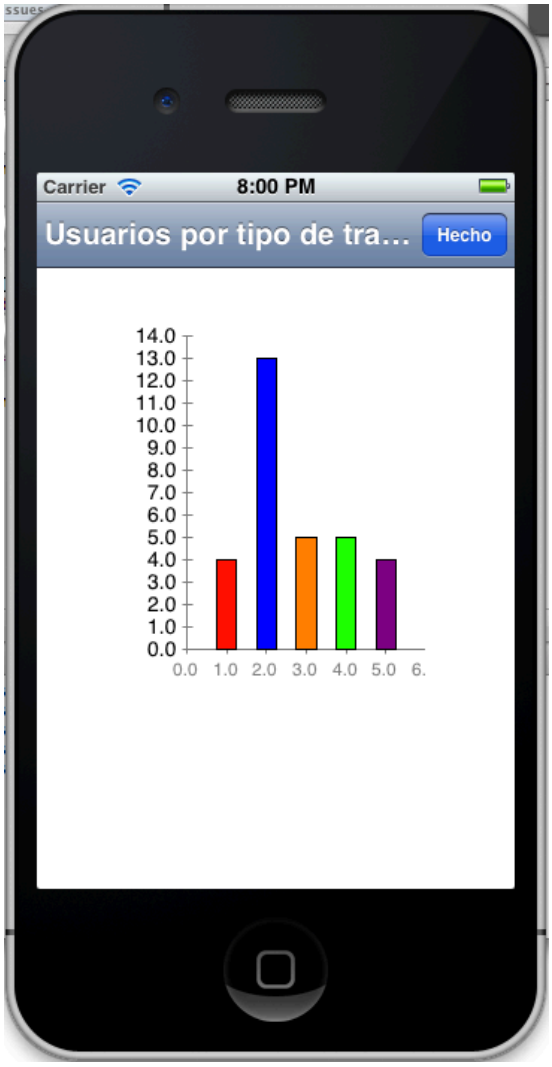

Carrier
7:50 PM
Usuarios por tipo de tra...
Hecho
14.0
13.0
12.0
11.0
10.0
9.0
8.0
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0.0
0.0 1.0 2.0 3.0 4.0 5.0 6.

Paso 3 Toques finales.

Podemos hacer algo para que sea más facil de ver. Queremos dar a cada barra un color diferente y proporcionar el nombre transporte como etiqueta del eje x en lugar de un número. Lo primero podemos hacerlo el primero con un método del CPTBarPlotDataSource llamado "barFillForBarPlot: recordIndex":

```
-(CPTFill *)barFillForBarPlot:(CPTBarPlot *)barPlot
recordIndex:(NSUInteger)index
{
    CPTColor *areaColor = nil;

    switch (index)
    {
        case 0:
            areaColor = [CPTColor redColor];
            break;

        case 1:
            areaColor = [CPTColor blueColor];
            break;

        case 2:
            areaColor = [CPTColor orangeColor];
            break;

        case 3:
            areaColor = [CPTColor greenColor];
            break;

        default:
            areaColor = [CPTColor purpleColor];
            break;
    }

    CPTFill *barFill = [CPTFill fillWithColor:areaColor];

    return barFill;
}
```

Esto devolverá un color diferente para cada barra del gráfico. Si quieres que sea aún más elegante, en realidad se podría hacer un degradado, como en los gráficos lineales.


Carrier
8:00 PM
Usuarios por tipo de tra...
Hecho
14.0
13.0
12.0
11.0
10.0
9.0
8.0
7.0
6.0
5.0
4.0
3.0
2.0
1.0
0.0
0.0 1.0 2.0 3.0 4.0 5.0 6.

Ahora vamos a ñadir los títulos al eje x. Para esto tenomos que trabajar con el "CPTXYAxis", en el método "ViewDidLoad" localiza donde lo hemos parametrizado y modifícalo para que quede así:

Finally, let's add some custom x-axis titles. To do this, we need to work with the x-axis. Locate where we set up the x-axis and modify the code to do the following:

```
CPTXYAxis *xAxis = [axisSet xAxis];
    [xAxis setMajorIntervalLength:CPTDecimalFromInt(1)];
    [xAxis setMinorTickLineStyle:nil];
    [xAxis setLabelingPolicy:CPTAxisLabelingPolicyNone];
    [xAxis setLabelTextStyle:textStyle];
    [xAxis setLabelRotation:M_PI/4];

    NSArray *transportsArray = [self getTransportTitlesAsArray];

    [xAxis setAxisLabels:[NSSet setWithArray:transportsArray]];
```

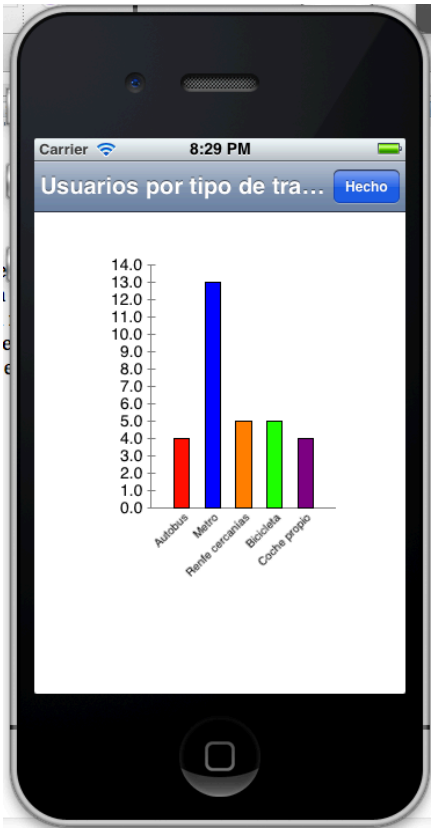
Hay algunos cambios en el código anterior. En primer lugar, estamos cambiando la política de etiquetado a ninguno. Esto asegurará que CorePlot no imprime la etiqueta en sí. A continuación, se establece la rotación de etiqueta, de modo que se alinee con la gráfica mejor. Por último, la propiedad "`AxisLabels`" toma como valor un NSSet de valores NSString. Creamos el NSSet utilizando un NSArray creado por `getTransportTitlesAsArray` . Este método aún no existe, así que lo vamos a crear, agregaremos la declarción en el "interface" del fichero .m y la siguiente implementación:

```
- (NSArray *)getTransportTitlesAsArray
{
    NSError *error = nil;

    NSFetchRequest *request = [[NSFetchRequest alloc] init];
    NSSortDescriptor *sortDescriptor = [NSSortDescriptor
sortDescriptorWithKey:@"transportID" ascending:YES];
    NSEntityDescription *entity = [NSEntityDescription
entityForName:@"KindTransport"
inManagedObjectContext:managedObjectContext];
    [request setEntity:entity];
    [request setSortDescriptors:[NSArray arrayWithObject:sortDescriptor]];
    [request setResultType:NSDictionaryResultType];
    [request setReturnsDistinctResults:NO];
    [request setPropertiesToFetch :[NSArray
arrayWithObject:@"transportName"]];

    NSArray *titleStrings = [managedObjectContext
executeFetchRequest:request error:&error];
    NSMutableArray *labelArray = [NSMutableArray array];

    CPTMutableTextStyle *textStyle = [CPTMutableTextStyle textStyle];
    [textStyle setFontSize:10];

    for (int i = 0; i < [titleStrings count]; i++)
    {
        NSDictionary *dict = [titleStrings objectAtIndex:i];
```

```
        CPTAxisLabel *axisLabel = [[CPTAxisLabel alloc] initWithText:[dict
objectForKey:@"transportName"] textStyle:textStyle];
        [axisLabel setTickLocation:CPTDecimalFromInt(i + 1)];
        [axisLabel setRotation:M_PI/4];
        [axisLabel setOffset:0.1];
        [labelArray addObject:axisLabel];
        [axisLabel release];
    }

    return [NSArray arrayWithArray:labelArray];
}
```

A fin de dar al gráfico las etiquetas personalizadas, necesitamos pasar un objeto que contiene un NSSet de 'CPTAxisLabel'. En primer lugar, se obtiene un array de todos los nombres de los tipos de transporte ordenados por su ID por lo que será en el mismo orden que en el gráfico. Por cada nombre recibido creamos un CPTAxisLabel con nombre del transporte y un estilo de texto predefinido. El tickLocation es el tick donde aparecerá el título por o que le sumamos 1 al igual que lo hicimos con las barras.

Este es el aspecto final del gráfico:

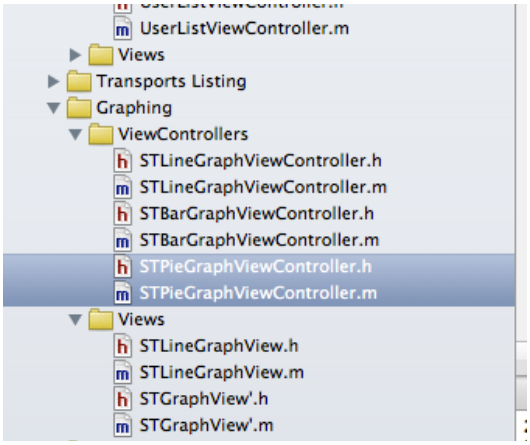


Track 5

Ahora vamos a ver como podemos mostrar los mismos datos en un gráfico circular:

Paso 1. Configuración

En primer lugar tenemos que añadir las clases correspondientes a nuestro proyecto. Vamos a crear una ViewController llamado "STPieGraphViewController'" esta vez no crearemos la vista y reutilizaremos la anterior.



Antes de seguir ves a "`UserListViewController`.h" y añade STPieGraphViewController.h. También necesitamos añadir el protocolo STPieGraphViewControllerDelegate (el cual crearemos después):

```
@interface UserListViewController : UIViewController <UITableViewDelegate,
UITableViewDataSource, AddUserViewControllerDelegate,
UIActionSheetDelegate, STLineGraphViewControllerDelegate,
STBarGraphViewControllerDelegate, STPieGraphViewControllerDelegate>
```

Ves al fichero .m. Necesitamos añadir un botón al action sheet. Localiza el método graphButtonWasSelected:. Y modifícalo para que quede así:

```
- (void)graphButtonWasSelected:(id)sender
{
    UIActionSheet *graphSelectionActionSheet = [[[UIActionSheet alloc]
initWithTitle:@"Escoge un gráfico" delegate:self
cancelButtonTitle:@"Cancelar" destructiveButtonTitle:nil
otherButtonTitles:@"Uso diario de transporte", @"Usuarios por tipo de
trnasporte –Bar", @"Usuarios por tipo de trnasporte –Pie" nil]
autorelease];

    [graphSelectionActionSheet showInView:[[UIApplication
sharedApplication] keyWindow]];
}
```

Ahora en el método "actionSheet:clickedButtonAtIndex: ç2 añadiremos la clausula para el buttonIndex == 2:

```
else if (buttonIndex == 2)
    {
        STPieGraphViewController *graphVC = [[STPieGraphViewController
alloc] init];
        [graphVC
setModalTransitionStyle:UIModalTransitionStyleFlipHorizontal];
        [graphVC setModalPresentationStyle:UIModalPresentationFullScreen];
        [graphVC setDelegate:self];
        [graphVC setManagedObjectContext:[self managedObjectContext]];

        [self presentModalViewController:graphVC animated:YES];

    }
```

Como antes, tenemos warnings porque no tiene delegate ni la propiedad managedObjectContext

En "STPieGraphViewController.h." Importa 'CorePlot-CocoaTouch.h' y añade ls siguientes propiedades y declaraciñón de protocolos:

```
#import "CorePlot-CocoaTouch.h"

@protocol STPieGraphViewControllerDelegate
@required
- (void)doneButtonWasTapped:(id)sender;

@end

@interface STPieGraphViewController : UIViewController
<CPTPieChartDelegate>

@property (nonatomic, strong) CPTGraph *graph;
@property (nonatomic, assign) id<STPieGraphViewControllerDelegate>
delegate;
@property (nonatomic, strong) NSManagedObjectContext
*managedObjectContext;
```

Para obtenr los datos de este gráfico vamos a reutilizar los métodos implementados en el gráfico de barras, para ellos abstraeremos la lógica en una clase separada. Antes de hacerlo, sin embargo, vamos a terminar de configurar todo. En el archivo. m, importar 'STGraphView.h', sintetiza las propiedades, y y realiza el correspondiente “release” en el método dealloc.
Por último, establecermos los métodos loadView y viewDidLoad como sigue:

```
- (void)loadView
{
    [super loadView];
    [self setTitle:@"Usuarios por tipo de transporte"];
    [self setView:[[[STGraphView alloc] initWithFrame:self.view.frame]
autorelease]];

    CPTTheme *defaultTheme = [CPTTheme themeNamed:kCPTPlainWhiteTheme];

    [self setGraph:(CPTGraph *)[defaultTheme newGraph]];
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    STGraphView *graphView = (STGraphView *)[self view];

    [[graphView chartHostingView] setHostedGraph:[self graph]];

    //Allow user to go back
    UINavigationItem *navigationItem = [[[UINavigationItem alloc]
initWithTitle:self.title] autorelease];
    [navigationItem setHidesBackButton:YES];

    UINavigationBar *navigationBar = [[[UINavigationBar alloc]
initWithFrame:CGRectMake(0, 0, self.view.frame.size.width, 44.0f)]
autorelease];
    [navigationBar pushNavigationItem:navigationItem animated:NO];

    [self.view addSubview:navigationBar];

    [navigationItem setRightBarButtonItem:[[[UIBarButtonItem alloc]
initWithTitle:@"Hecho"

style:UIBarButtonItemStyleDone

target:[self delegate]

action:@selector(doneButtonWasTapped:)] autorelease] animated:NO];
}
```

El código de arriba ya nos debe ser familiar, vamos a abstraer la lógica de datos.

Paso2. Separando la lógica del gráfico

Ya hemos escrito la lógica para la obtención de datos sobre el número total de usuarios de todos los tipos de transporte, por lo queno queremos tener que escribirla de nuevo, por suerte, todos los datasource de los gráficos heredan de "CPTPlotDataSource ', y es este protocolo el que contiene los métodos numberOfRecordsForPlot: y numberForPlot: fieldEnum: recordIndex .

Crea una clase llamada "STAbstractTransportDataSource.h '(que hereda de la NSObject) en un nuevo grupo llamado' DataSource' dentro de 'Graphing'. Importa 'CorePlot-CocoaTouch.h "y crea las siguientes propiedades y declaraciones de métodos:

```
@interface STAbstractSubjectEnrollementDataSource : NSObject
<CPTPlotDataSource>

@property (nonatomic, strong) NSManagedObjectContext
*managedObjectContext;

- (id)initWithManagedObjectContext:(NSManagedObjectContext
*)aManagedObjectContext;

- (float)getTotalTransports;
- (float)getMaxUsers;
- (NSArray *)getTransportTitlesAsArray;
@end
```

Hemos añadido el protocolo 'CPTPlotDataSource'. Creamos un método personalizado de inicio que recibe un managedObjectContext con lo que puede tener acceso a la base de datos. Por último, hay tres métodos de ayuda que para la obtención de información acerca de los usuarios y los transportes. Estos son los mismos métodos que existen actualmente en STBarGraphViewController. Vamos a moverlos a la unidad de Data source que acabamos de crear
.
Aparte del método de inicio, el archivo. m no contiene ningún código nuevo que no hayamos visto antes. Los métodos que se mueven son los siguientes:

```
- (float)getTotalTransports;
- (float)getMaxUsers;
- (NSArray *)getTransportTitlesAsArray;
- (NSUInteger)numberOfRecordsForPlot:(CPTPlot *)Plot;
- (NSNumber *)numberForPlot:(CPTPlot *)plot field:(NSUInteger)fieldEnum
recordIndex:(NSUInteger)index;
{
```

asegúrate de añadir el método de inicialización:

```
- (id)initWithManagedObjectContext:(NSManagedObjectContext
*)aManagedObjectContext
{
    self = [super init];
    if (self)
    {
        [self setManagedObjectContext:aManagedObjectContext];
    }

    return self;
}
```

Ahora tenemos un objeto “DataSource” que puede proporcionar los datos básicos, tanto para el gráfico circular como para gráfico de barras. Sin embargo necesitamos el método barFillForBarPlot: recordIndex y es parte del CPTBarPlotDataSource, por lo que no podemos incluir lo en nuestra nueva clase. Vamos a tener que ampliarla extenderla para los gráficos de barras.
Crea un nuevo objeto en el grupo DataSource llamado “STBarTransportDataSource" que extiende “STAbstractTransportDataSource”. En la cabecera añade el protocolo CPTBarPlotDataSource:

```
@interface STBarTransportDataSource : STAbstractTransportDataSource
<CPTBarPlotDataSource>
```

Y en el fichero .m , implementa el método barFillForBarPlot:

```
-(CPTFill *)barFillForBarPlot:(CPTBarPlot *)barPlot
recordIndex:(NSUInteger)index
{
    CPTColor *areaColor = nil;

    switch (index)
    {
        case 0:
            areaColor = [CPTColor redColor];
            break;

        case 1:
            areaColor = [CPTColor blueColor];
            break;

        case 2:
            areaColor = [CPTColor orangeColor];
            break;

        case 3:
            areaColor = [CPTColor greenColor];
            break;

        default:
```

```
            areaColor = [CPTColor purpleColor];
            break;
    }

    CPTFill *barFill = [CPTFill fillWithColor:areaColor];

    return barFill;
}
```

Ahora en STBarGraphViewControllers.h e importa el nuevo DataSource para el gráfico de barras. Ahora puedemos eliminar el 'CPTBarPlotDataSource'. En el fichero .m elimina todos los métodos a excepción de loadView, viewDidLoad y dealloc.
Tenemos que mantener un puntero al datasource y luego liberarlo cunado la vista ya no lo necesite. En la interface privado, declarar la propiedad y luego sintetízala :

```
@interface STBarGraphViewController ()

@property (nonatomic, retain) STBarTransportDataSource
*barGraphDataSource;

@end


@implementation STBarGraphViewController

@synthesize delegate;
@synthesize managedObjectContext;
@synthesize graph;
@synthesize barGraphDataSource;
```

Asegurte de liberar la nueva propiedad en el método dealloc. Crea una nueva instancia y asignala en el método loadview:

```
[self setBarGraphDataSource:[[[STBarTransportDataSource alloc]
initWithManagedObjectContext:[self managedObjectContext]] autorelease]];
```

Ahora tenemos que hace unas modificaciones en el método para usar los métodos para crear las etiquetas del nuevo DataSource:

```
[subjectBarPlot setDataSource:[self barGraphDataSource]];

CPTXYPlotSpace *userPlotSpace = (CPTXYPlotSpace *)[graph
defaultPlotSpace];
      [userPlotSpace setXRange:[CPTPlotRange
plotRangeWithLocation:CPTDecimalFromInt(0) length:CPTDecimalFromInt([[self
barGraphDataSource] getTotalTransports] + 1)]];
    [userPlotSpace setYRange:[CPTPlotRange
plotRangeWithLocation:CPTDecimalFromInt(0) length:CPTDecimalFromInt([[self
barGraphDataSource] getMaxUsers] + 1)]];
```

```
NSArray *transportsArray = [[self barGraphDataSource]
getTransportTitlesAsArray];
```

Ahora todo debería funcionar como antes, vamos a crear ahora el gráfico circular

Paso 3 Crear el gráfico.

Tendremos que crear un DataSource específico para el gráfico circular tal y como hicimos para el gráfico de barras por si lo necesitemos para implementar algún método específico del DataSource de gráficos circulares.

Crea una clase llamada "STPieGraphTransportDataSource 'que hereda de' STAbstractTrnasportDataSource '. En el archivo de cabecera, añade el protocolo 'CPTPieChartDataSource'.

Ahora que tenemos la fuentes de datos, la creación del gráfico circular es muy sencillo! Ir a la STBPieGraphViewController.m e importar el nuevo data source creado. Declarar como una propiedad en el archivo de m, como lo hicimos la última vez.:

```
@interface STPieGraphViewController ()

@property (nonatomic, strong) STPieGraphTransportDataSource
*pieChartDataSource;

@end

@implementation STPieGraphViewController

@synthesize managedObjectContext;
@synthesize delegate;
@synthesize graph;
@synthesize pieChartDataSource;
```

Crealo y asignalo en el LoadView:

```
[self setPieChartDataSource:[[[STPieGraphTransportDataSource alloc]
initWithManagedObjectContext:[self managedObjectContext]] autorelease]];
```

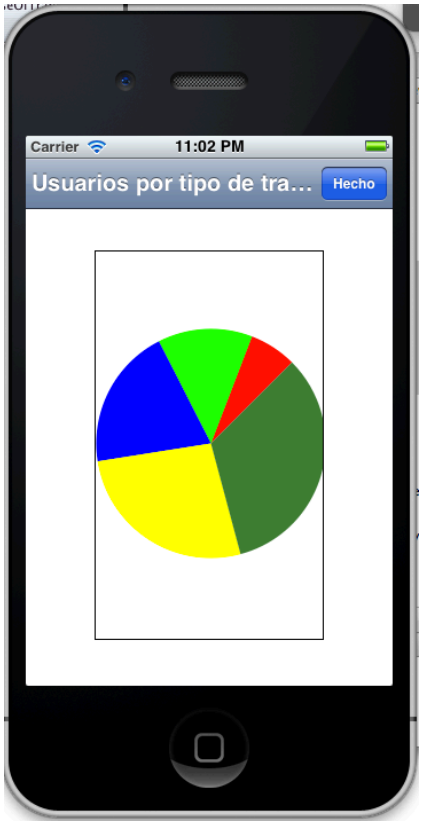
Finally, in the viewDidLoad method we need to create our pie chart, add it to our GraphView and remove the standard axis:
Finalmente em¡n el método ViewDidLoad necesitamos crear nuestro gráfico y añadirlo al GraphView, asi como quitar las ejes estandar:

```
CPTPieChart *pieChart = [[CPTPieChart alloc] initWithFrame:[graph
bounds]];
    [pieChart setPieRadius:100.00];
```

```
[pieChart setIdentifier:@"Transport"];
[pieChart setStartAngle:M_PI_4];
[pieChart setSliceDirection:CPTPieDirectionCounterClockwise];
[pieChart setDataSource:pieChartDataSource];
[graph addPlot:pieChart];

[graph setAxisSet:nil];
[graph setBorderLineStyle:nil];
```

La mayor parte de lo anterior debería resultar familiar. Ten en cuenta que no se llama explícitamente a un "Plot" porque no se basa en un eje de abscisas, pero todavía nos queda mucho tratamiento. Hay algunas cosas del gráfico de sectores específicos que hacemos aquí. Creamos un radio de pastel y el ángulo de partida. También establecemos una dirección de la portión. Por último ponemos el 'axisSet' de la gráfica a cero, para que no nos muestre la x e y.

Podríamos hacer con algún tipo de indicación en cuanto a lo que cada color representa. Una buena manera de hacer esto es utilizar leyendas. Para ello creamos un 'CPTLegend' objeto que añadimos a nuestro gráfico y creamos el método del delegado que devuelve el título para la leyenda.
Vamos a crear el objeto CPTLegend primero. En nuestro método viewDidLoad introduzca el siguiente código debajo de donde creamos nuestro gráfico circular:

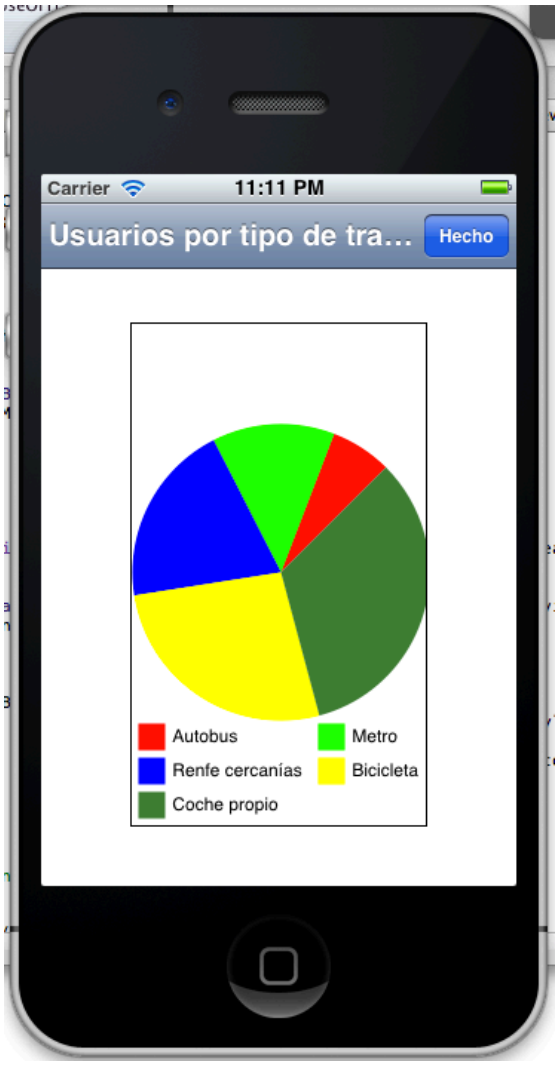


```
CPTLegend *theLegend = [CPTLegend legendWithGraph:[self graph]];
    [theLegend setNumberOfColumns:2];
    [[self graph] setLegend:theLegend];
    [[self graph] setLegendAnchor:CPTRectAnchorBottom];
    [[self graph] setLegendDisplacement:CGPointMake(0.0, 30.0)];
```

Esto crea una leyenda y lo añade a nuestro objeto gráfico. El número de columnas determina cómo se va a exponer los títulos de la leyenda. A continuación, establecemos algunos atributos en el gráfico que determina el lugar donde se coloca laleyenda y un desplazamiento para asegurarsnos de que demuestra plenamente en la vista.
Todavía tenemos que dar los títulos a la leyenda. Hay un método específico en CPTPieChartDataSource que nos permite hacer esto. Añadir al datasource de gráficos circulares el siguiente código:

```
#pragma mark - CPTPieChartDataSource
-(NSString *)legendTitleForPieChart:(CPTPieChart *)pieChart
recordIndex:(NSUInteger)index
{
    NSError *error = nil;

    NSFetchRequest *request = [[NSFetchRequest alloc] init];
    NSEntityDescription *entity = [NSEntityDescription
entityForName:@"KindTransport" inManagedObjectContext:[self
managedObjectContext]];
    NSPredicate *predicate = [NSPredicate
predicateWithFormat:@"transportID == %d", index];
    [request setEntity:entity];
    [request setResultType:NSDictionaryResultType];
    [request setPredicate:predicate];
    [request setReturnsDistinctResults:NO];
    [request setPropertiesToFetch:[NSArray
arrayWithObject:@"transportName"]];

    NSArray *titleStrings = [[self managedObjectContext]
executeFetchRequest:request error:&error];

    NSDictionary *fetchedSubject = [titleStrings objectAtIndex:0];

    return [fetchedSubject objectForKey:@"transportName"];
}
```

Este método obtiene el índice de la leyenda y el título desde la base y lo devuelve.
Ejecuta el gráfico…!


Carrier
11:11 PM
Usuarios por tipo de tra...
Hecho
Autobus
Metro
Renfe cercanías
Bicicleta
Coche propio

Gracias!