6-Scroll View, Table View, Web View

Scroll views

Le viste scorrevoli permettono agli utenti di vedere un contenuto che è più grande dei limiti della vista. L'indicatore scorrimento può essere sia verticale che orizzontale e notifica all'utente che c'è alatri da vedere.

Le viste scorrevoli sono implementante nella classe `UIScrollView`, e permette all'utente di:

- scorrere all'interno di un contenuto facendo dei gesti di scorrimento
- ingrandire/rimpicciolire da una porzione di contenuti facendo dei gesti di pizzico

Una `UIScrollView` è una vista la cui origine è aggiustabile sulla vista del contenuto.

Aggiungere del contenuto ad una scroll view

Avviene in 2 fasi:

1. aggiungere una vista ad una vista scorrevole

```
UIImagieView *imageView = [[UIImageView alloc] initWithImage:image];
[scrollView addSubview:imageView];
```

2. Impostare la dimensione del contenuto della vista scorrevole

```
scrollView.contentSize = CGSizeMake(3000, 3000);
// oppure
scrollView.contentSize = imegeView.bounds.size;
```

La proprietà `contentOffset` ritorna l'angolo in alto a sinistra della porzione corrente della vista che sta per esse mostrata sulla vista scorrevole

```
scrollView.contentOffset.x, scrollView.contentOffset.y
```

La proprietà `bounds` ritorna i rettangolo che è correntemente visibile

```
scrollView.bounds.origin.x, scrollView.bounds.origin.y
scrollView.bounds.size.width, scrollView.bounds.size.height
```

Le coordinate della sottovista nell'area visibile potrebbe sere differente da quella ritornata dalle proprietà dei bounds, a causa degli zoom

```
CGRect visibleRect = [scrollView convertRect:scrollView.bounds toView:subView];
```

Aggiungere una vista scorrevole

Il modo consigliato di usare una vista scorrevole è di aggiungerla ad una vista trascinandola dalla palette degli oggetti e dopo aggiungere la sottovista programmaticamente, mediante `addSubview:`. Oltre a questo è possibile aggiungerla nel codice con alloc/init.

Tutte le sottovista aggiunte ad una vista scorrevole sono automaticamente posizionate all'inizio (angolo in alto a sinistra) della vista scorrevole. Per cambiare la posizione bisogna impostare un nuovo frame

```
subview.frame = CGRectMake(200, 200, 100, 100);
```

> Nota
> È fondamentale impostare la proprietà `contentSize` di una vista scorrevole

Per avere una vista scorrevole della dimensione uguale a quella delle sotto viste

```
scrollView.contentSize = subview.bounds.size;
```

Lavorare con le viste scorrevoli

Le viste scorrevoli supportano nativamente lo scroll dell'utente, ma è anche possibile eseguirlo in modo programmatico

```
- (void)scrollRectToVisible:(CGRect)rect animated:(BOOL)animated
```

È possibile abilitare e disabilitare lo scroll attraverso la proprietà `scrollEnabled`. Un altra proprietà interessante è `directionalLockEnabled` che può essere utilizzata per bloccare la direzione dello scroll in base allo primo movimento fatto dall'utente.

Zoomare nelle viste scorrevoli

Le viste scorrevoli supportano lo zoom dei contenuti pizzicando dentro/fuori. Per abilitare questa funzione è necessario compiere alcune azioni preliminari:

1. Impostare la scala minima e massima della scala dello zoom

```
self.scrollView.minimumZoomScale = 0.5; // 50%
self.scrollView.maximumZoomScale = 3.0; // 300%
```

2. Dichiarare il view controller come `UIScrollViewDelegate` ed impostarlo come incaricato dello scroll

```
@interface MyViewController : UIViewController<UIScrollViewDelegate>
// or ...
@interface MyViewController ()<UIScrollViewDelegate>
self.scrollView.delegate = self;
```

3. implementare il metodo delegato per specificare la sottovista che deve essere zoomata

```
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)sender
```

UIScrollViewDelegate

Il protocollo `UIScrollViewDelegate` fornisce diversi metodi per gestire eventi di scroll, zoom e trascinamento nelle viste scorrevoli:

```
- (void)scrollViewDidScroll:(UIScrollView *)scrollView
- (void)scrollViewDidZoom:(UIScrollView *)scrollView
- (void)scrollViewWillBeginZooming:(UIScrollView *)scrollView
withView:(UIView *)view
- (void)scrollViewDidEndZooming:(UIScrollView *)scrollView
withView:(UIView *)view
atScale:(CGFloat)scale
- (void)scrollViewWillBeginDragging:(UIScrollView *)scrollView
- (void)scrollViewDidEndDragging:(UIScrollView *)sc
```

Table views

Le viste tabellari sono usate per mostrare a schermo e modificare gerarchicamente liste di informazioni.
Questo tipo di vista mostra una lista di oggetti raggruppati in un singola colonna e una o più righe.

> Nota
> Le righe possono essere raggruppate in sezioni

Dati con più dimensioni sono solitamente rappresentati combinando le viste tabulari all'interno di una navigation controller.
La classe che implementa la vista tabulare è `UITableView`, che è una sotto-classe della `UIScrollView` dove è permesso solo lo scroll verticale

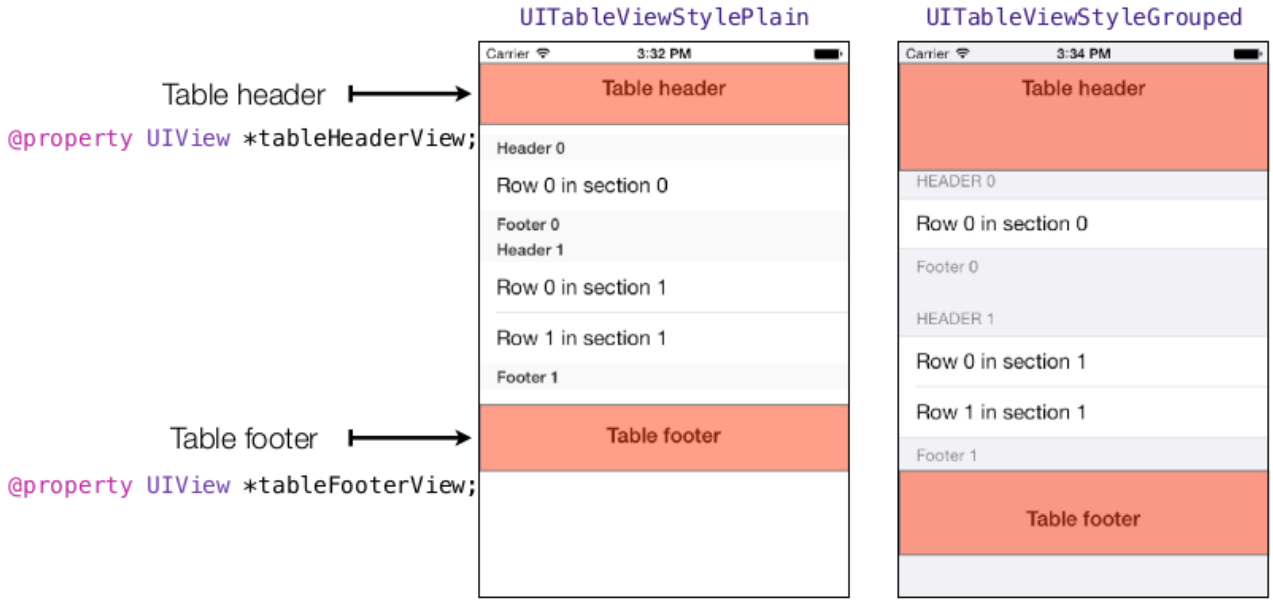
> Nota
> Le `UITableView` possono gestire in maniera efficiente grandi quantità di dati

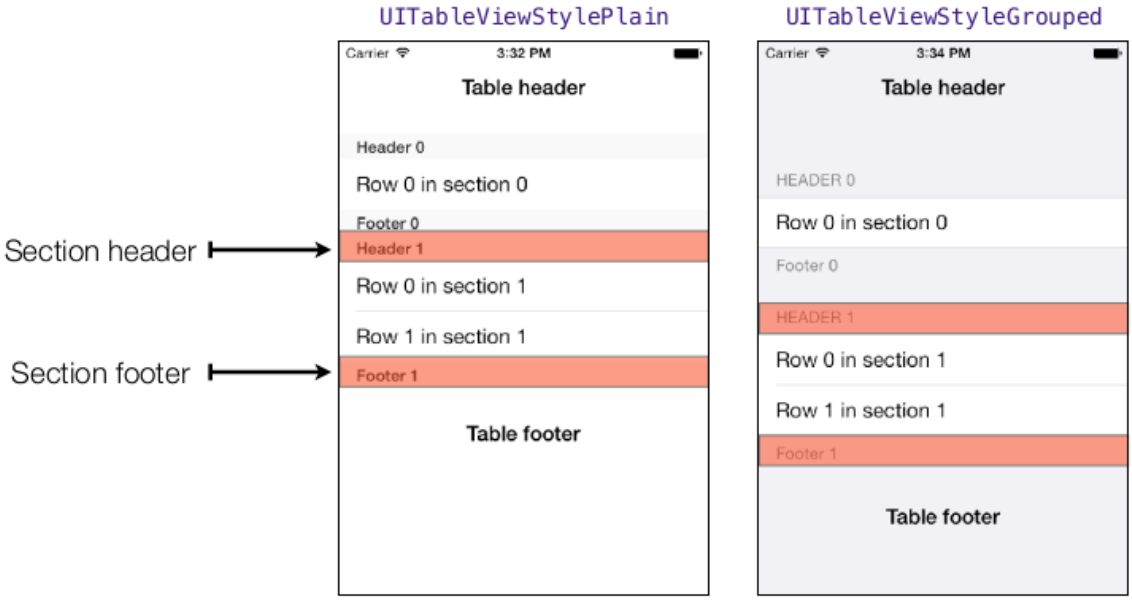
Gli stili delle viste tabellari

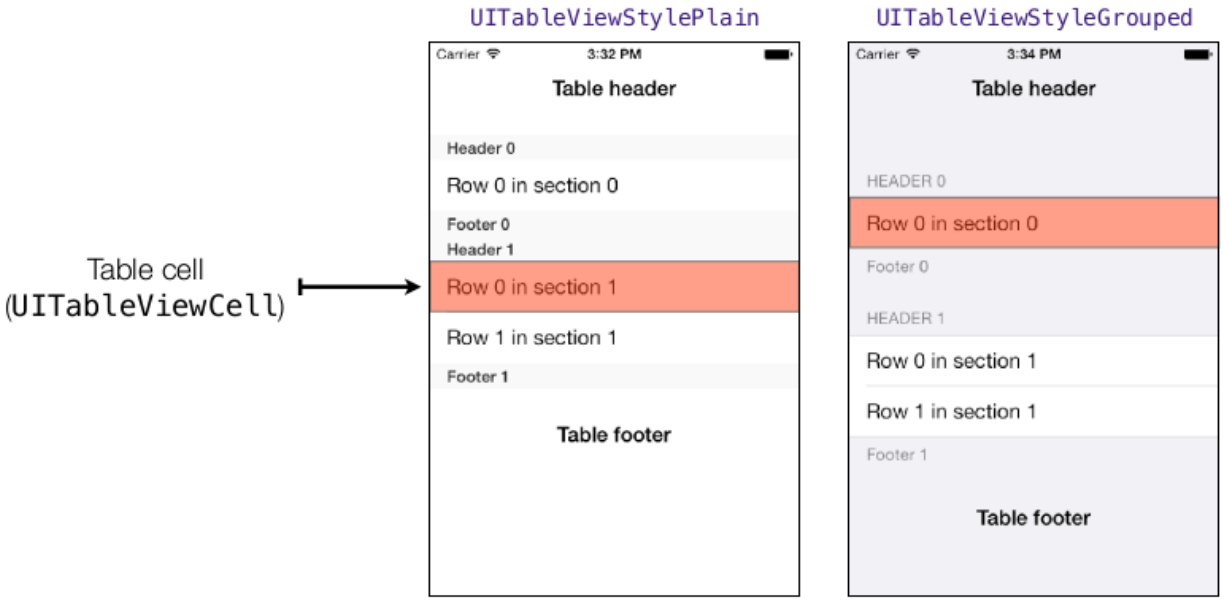
Una vista tabellare può essere:

- statica, hanno contenuti che sono mutabili e non predicibili
- dinamica, hanno un contenuto prefissato, deciso dal programmatore

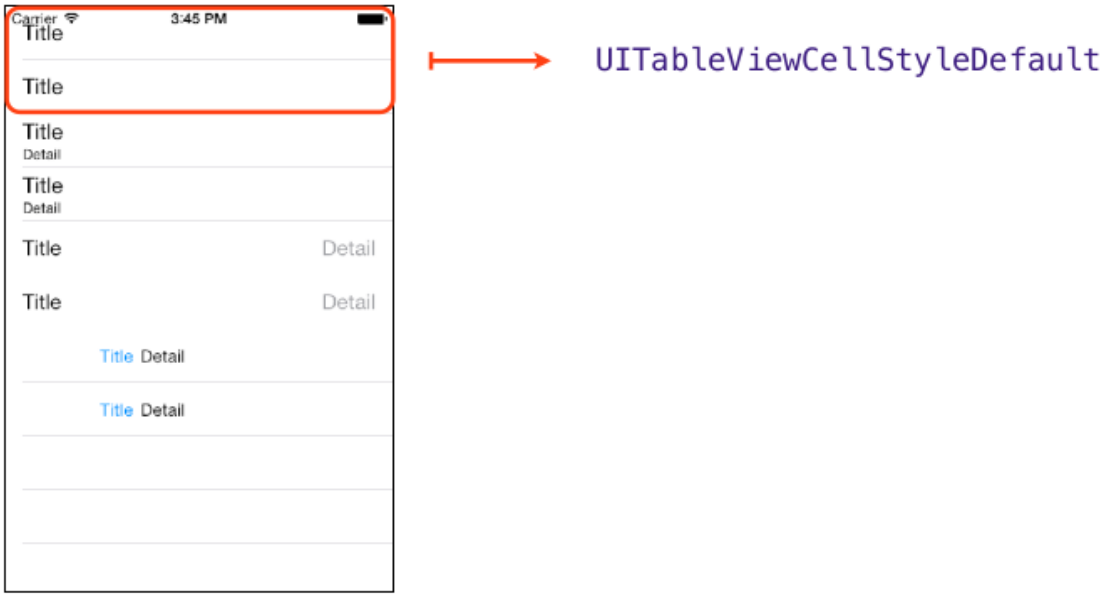
Alcune immagini

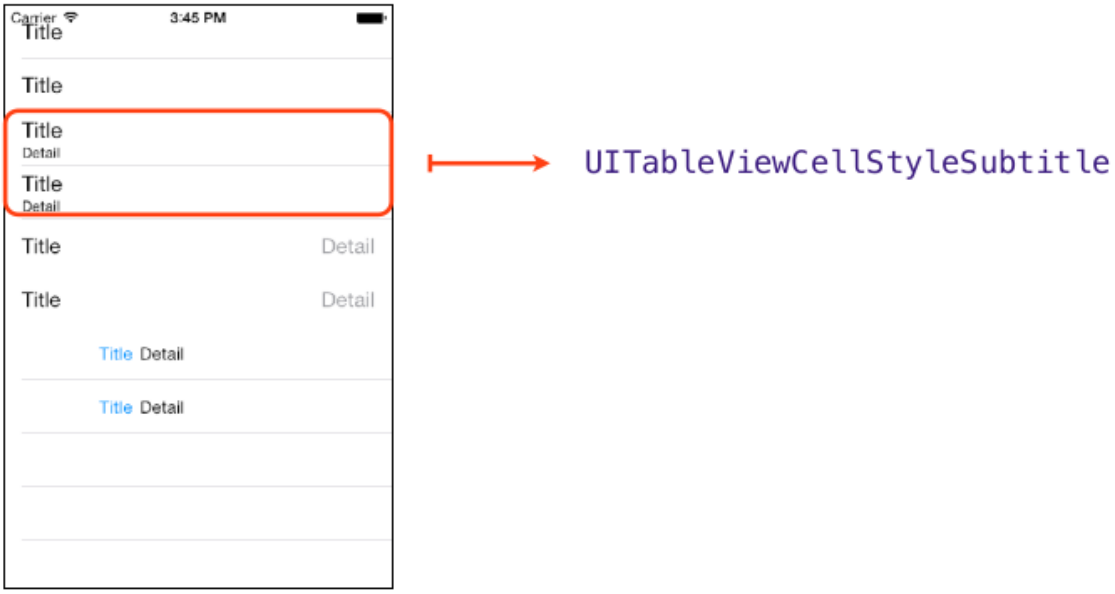


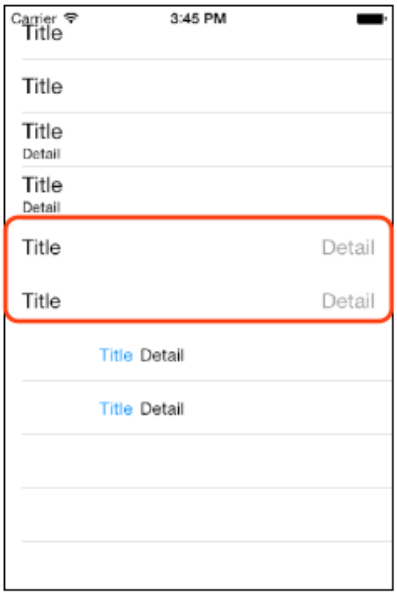




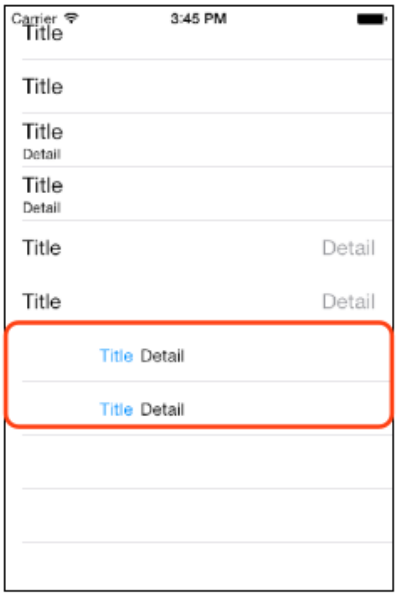
Le celle definite di default possono avere stili differenti in base a quale contenuto si è intenzionati a rappresentare. Le celle possono essere personalizzate per avere aspetto differente in base a quello che deve essere rappresentato.







`UITableViewCellStyleValue1`



`UITableViewCellStyleValue2`

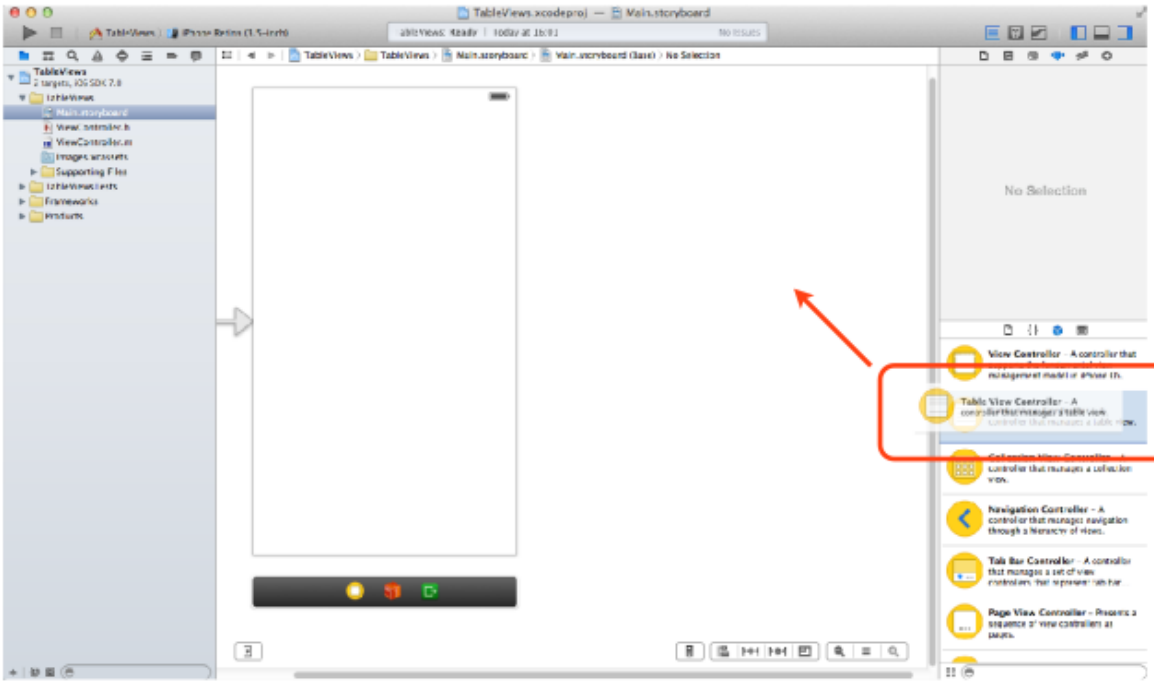
Controller delle viste tabulari

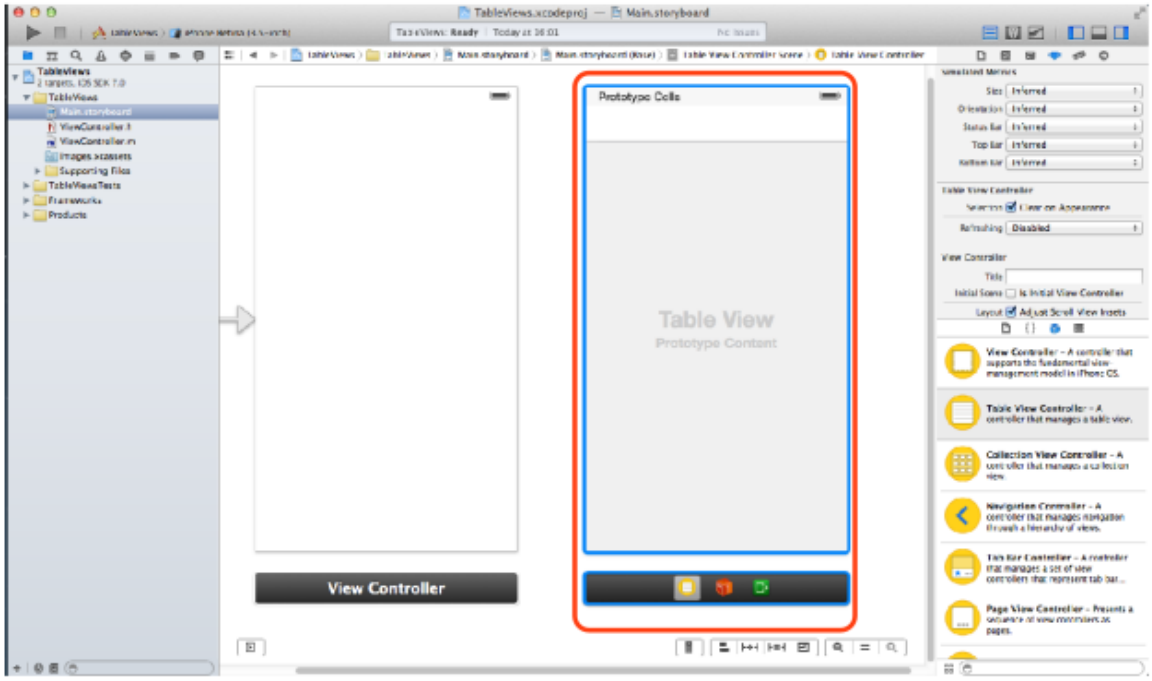
Una vista tabulare è solitamente gestita da un view controller specializzato chiamato `UITableViewController`.

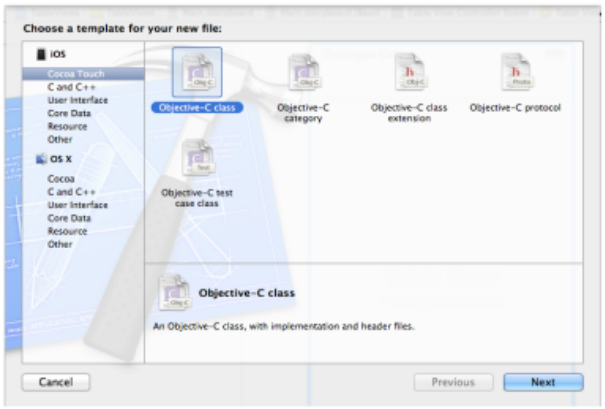
> Nota
> Normalmente le viste tabulari occupano l'intera vista

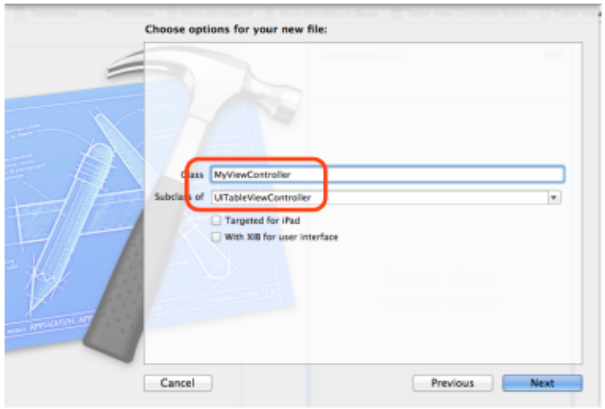
Per aggiungere table view controller alla storyboard basta semplicemente trascinare e rilasciare il controller dalla palette degli oggetti; il controller è una sottoclasse di `UITableViewController` e la vista è una `UITableView`.
Dopo avere trascinato il controller è necessario creare una nuova classe Objective-C che sarà una sottoclasse di `UITableViewController` ed impostarla come la classe della vista nell'ispettore d'identità nello storyboard.

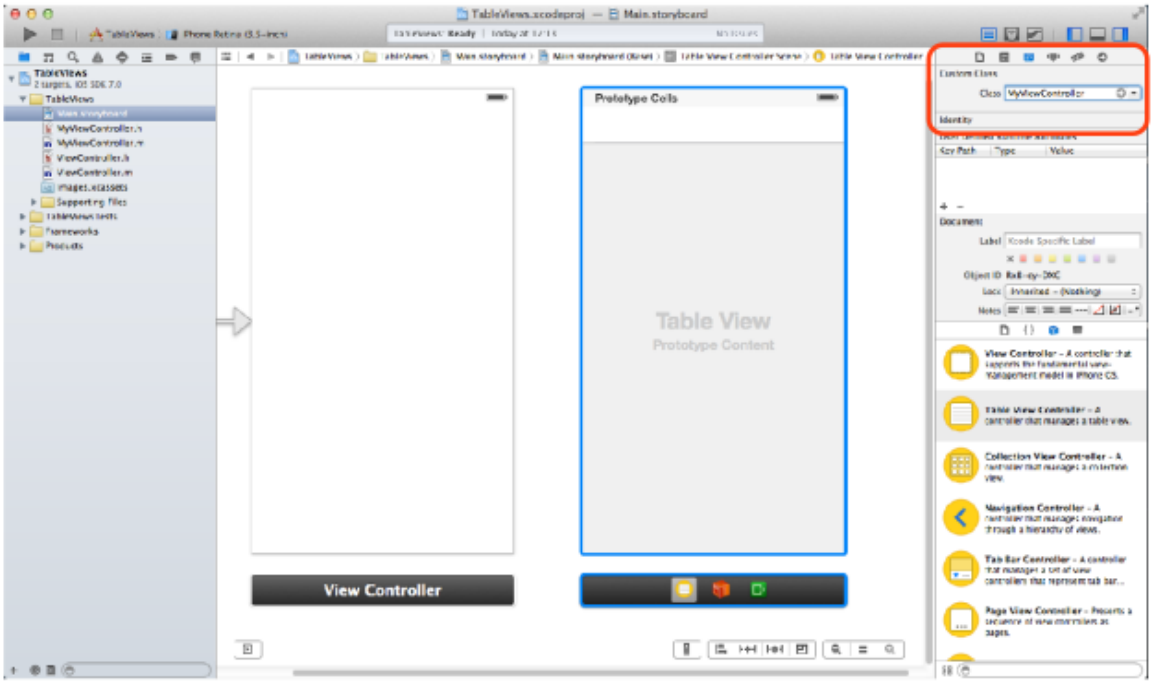
> Nota
> Le viste tabulari possono essere configurate per impostare il loro stile ed il loro tipo di contenuto (statico/dinamico)

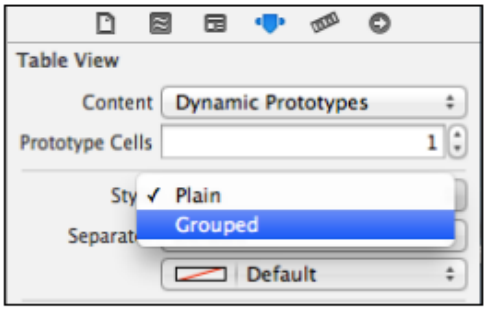


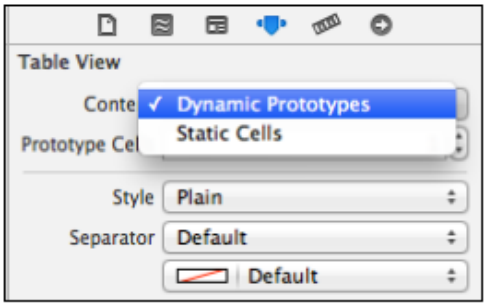

Prototype Cells
Table View
Prototype Content
View Controller


Choose a template for your new file:
iOS
Cocoa Touch
C and C++
User Interface
Core Data
Resource
Other
OS X
Cocoa
C and C++
User Interface
Core Data
Resource
Other
Objective-C class
Objective-C category
Objective-C class extension
Objective-C protocol
Objective-C test case class
Objective-C class
An Objective-C class, with implementation and header files.
Cancel
Previous
Next

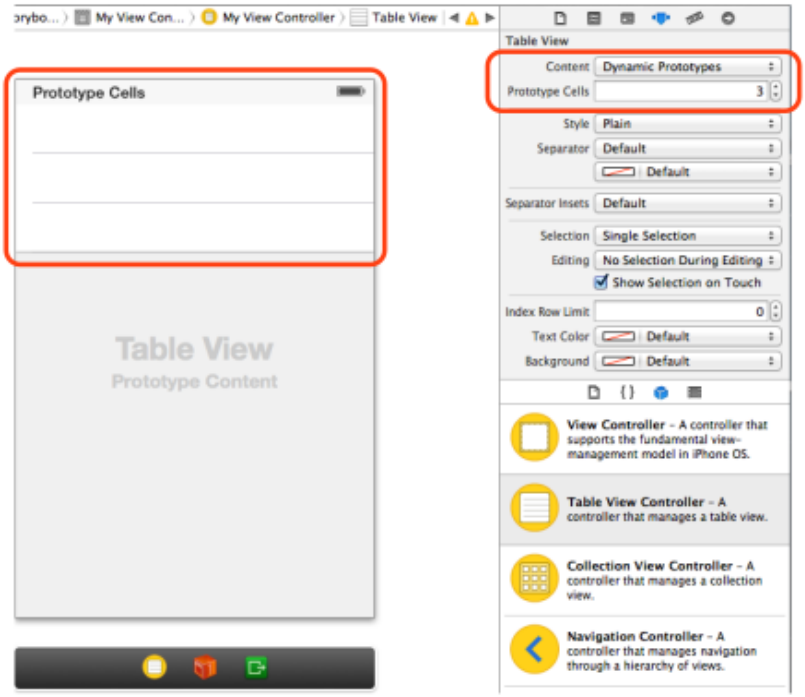

Choose options for your new file:
Class
MyViewController
Subclass of
UITableViewController
Targeted for iPad
With XIB for user interface
Cancel
Previous
Next



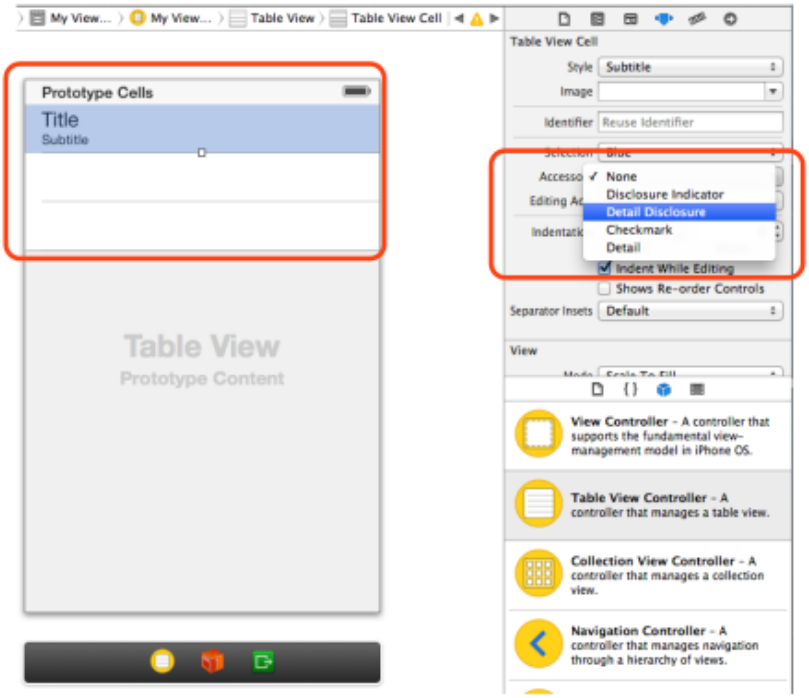


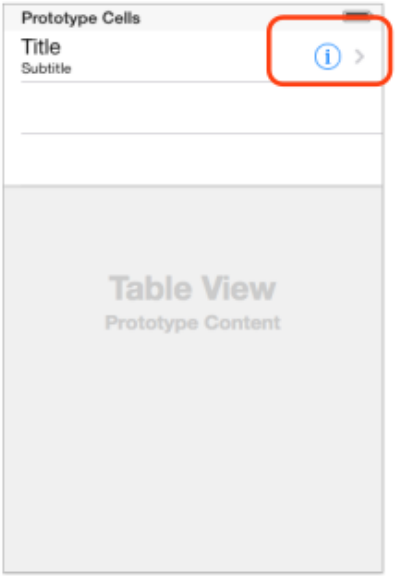


Quando è selezionato il prototipo dinamico è possibile configurare i template per le celle che verranno mostrate.



Per ogni cella è possibile impostare lo stile e gli accessori che rappresenteranno un suggerimento quando la riga sarà selezionata.





È necessario specificare un identificatore per le celle così da poterci fare riferimento all'interno del codice.

NSIndexPath

La classe `NSIndexPath` è usata per rappresentare un percorso per uno specifico nodo in un albero di collezioni di array nidificati. In iOS la `UIKit` fornisce alcune funzionalità per le istanze di `NSIndexPath` quindi queste possono essere utilizzate per riferirsi alla posizione di un cella in una vista tabulare.
Ci sono 2 proprietà che sono usate per questo scopo:

- `section`, un numero che funge da indice ed identifica una determinata sezione in una vista tabulare
- `row`, un numero che funge da indice che identifica una determinata riga di una sezione

Un istanza di `NSIndexPath` è passata come argomento al metodo che fa riferimento a quella specifica cella nella vista tabulare, così da identificare la cella in modo univoco.

Table view content

Una vista tabulare si basa su un oggetto esterno, chiamato sorgente dei dati (data source) per inserire il contenuto nelle righe. Questa sorgente si trova tra il modello e la vista tabulare, infatti è possibile che il controller sia la sorgente dei dati per la vista che gestisce.

Generalmente le sorgenti forniscono il contenuto alle tavole rispondendo alle seguenti domande:

- Quante sezioni deve avere la vista?
- Quante righe ci saranno nella sezione i?
- Quale contenuto dovrà essere mostrato (`UITableViewCell`) nella cella nella sezione i e riga j ?

- Quale titolo dovrà essere mostrato per le sezioni "header" e "footer"?
- La cella nella sezione i e riga j è moficabile?

Una sorgente di dati deve essere conforme ai protocolli della `UITableVIewDataSource`.

> Nota
> Le tabelle statiche non necessitano di implementare i protocolli della `UITableVIewDataSource`, perché avverrà in automatico. Diversamente le tabelle dinamiche dovranno implementare i protocolli, perché ciò che la sorgente fornirà corrisponde a ciò che verrà mostrato

UITableViewDataSource

Il protocollo `UITableViewDataSource` definisce i seguenti metodi che devono essere implementati da una sorgente di dati per fornire abbastanza informazioni alla `UITableView` per mostrare i sui contenuti.

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView

- (NSInteger)tableView:(UITableView *)tableView
                              numberOfRowsInSection:(NSInteger)section
- (UITableViewCell *)tableView:(UITableView *)tableView
                              cellForRowAtIndexPath:(NSIndexPath *)indexPath

- (NSString *)tableView:(UITableView *)tableView
                              titleForHeaderInSection:(NSInteger)section
- (NSString *)tableView:(UITableView *)tableView
                              titleForFooterInSection:(NSInteger)section

- (BOOL)tableView:(UITableView *)tableView
                              canEditRowAtIndexPath:(NSIndexPath *)indexPath

- (void)tableView:(UITableView *)tableView
                              commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
                              forRowAtIndexPath:(NSIndexPath *)indexPath
```

esempi di implementazioni

```
// Quante sezioni deve avere la vista?
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView{
        // Ritorna il numero di sezioni
        return <NUMBER OF SECTIONS>;
}
// Quante righe ci saranno nella sezione i?
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section{
        // Ritorna il numero di righe in una data sezione
        return <NUMBER OF ROWS FOR SECTION section>;
}

// Quale contenuto dovrà essere mostrato (`UITableViewCell`) nella cella nella sezione $i$ e
riga $j$ ?
- (UITableViewCell *)tableView:(UITableView *)tableView
        cellForRowAtIndexPath:(NSIndexPath *)indexPath{
        // Identificatore della cella impostato nell'ispettore degli attributi
        static NSString *CellIdentifier = @"CellIdentifier";
        // La cella deve essere riutilizzata perché la creazione di un altra cella è onerosa
        UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier
                                                                forIndexPath:indexPath];

        // Configura la cella
        cell.textLabel.text = [NSString stringWithFormat:@"cell at row [%d]", indexPath.row];

        return cell;
}


// Quale titolo dovrà essere mostrato per le sezioni "header" e "footer"?
- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section{
        return [NSString stringWithFormat:@"Header %d",section];
}
- (NSString *)tableView:(UITableView *)tableView titleForFooterInSection:(NSInteger)section{
        return [NSString stringWithFormat:@"Footer %d",section];
}
```

Il comportamento delle viste tabulari

Una vista tabulare sfrutta un oggetto esterno, chiamato delegato, per specificare il suo aspetto e comportamento. Il delegato è necessario per gestire selezioni, configurare sezioni headers/footers, aiuta a eliminare/riordinare celle e molto altro.
È comune che il controller è il delegato per la vista tabulare che gestisce. Generalmente il delegato implementa i metodi che rispondono alle domande:

- Cosa dovrebbe essere fatto se una cella è prossima ad essere/è stata selezionata/deselezionata? (comportamento)
- Cosa dovrebbe essere fatto se un bottone accessorio di una cella viene premuto? (comportamento)
- Quale altezza dovrebbe avere una cella? (stile)
- Quale vista dovrebbe essere mostrata per header/footer di una determinata sezione? (stile)

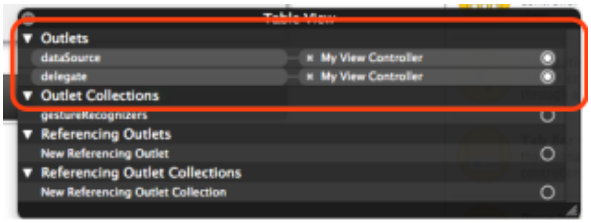
Un delegato deve implementare i protocolli conformi a `UITableViewDelegate` :

```
- (NSIndexPath *)tableView:(UITableView *)tableView
                            willSelectRowAtIndexPath:(NSIndexPath *)indexPath
- (NSIndexPath *)tableView:(UITableView *)tableView
                            didSelectRowAtIndexPath:(NSIndexPath *)indexPath
- (NSIndexPath *)tableView:(UITableView *)tableView
                            willDeselectRowAtIndexPath:(NSIndexPath *)indexPath
- (NSIndexPath *)tableView:(UITableView *)tableView
                            didDeselectRowAtIndexPath:(NSIndexPath *)indexPath
- (void)tableView:(UITableView *)tableView
                            accessoryButtonTappedForRowWithIndexPath:(NSIndexPath
*)indexPath
- (CGFloat)tableView:(UITableView *)tableView
                            heightForRowAtIndexPath:(NSIndexPath *)indexPath
- (UIView *)tableView:(UITableView *)tableView
                            viewForHeaderInSection:(NSInteger)sectionection
- (UIView *)tableView:(UITableView *)tableView
                            viewForFooterInSection:(NSInteger)sectionection
```

UITableViewController

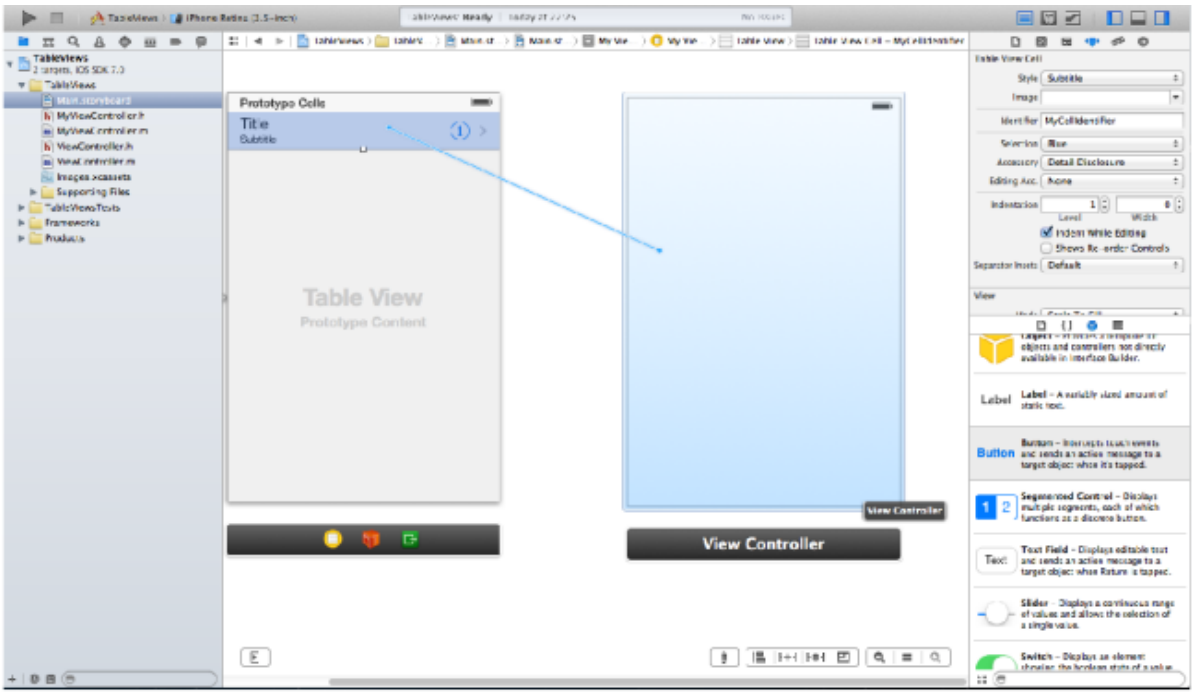
Un `UITableViewController` imposta da solo la sorgente dei dati ed il delegato per la vista che gestisce. Ha una proprietà, `tableView`, che può essere usata per riferirsi alla vista tabulare `@property UITableView *tableView;`.

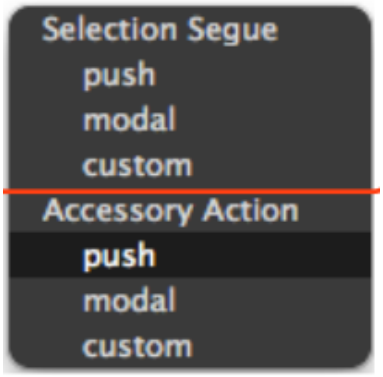
> Nota
> Per un `UITableViewController` è presente la seguente proprietà `self.view == self.tableView;`



Quando una sottoclasse di `UITableViewController` è creata, Xcode fornisce un implementazione basilare dei metodi più importanti che devono essere implementato per essere conformi al protocollo `UITableViewDataSource`.

```
#pragma mark - Table view data source
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView{
        // Ritorna il numero di sezuioni
        return 1;
}
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section{
        // Ritorna il numero di righe in una sezioni
        return 0;
}
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath{
        static NSString *CellIdentifier = @"CellIdentifier";
        UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier
forIndexPath:indexPath];
        // Configura la cella...
        return cell;
}
```





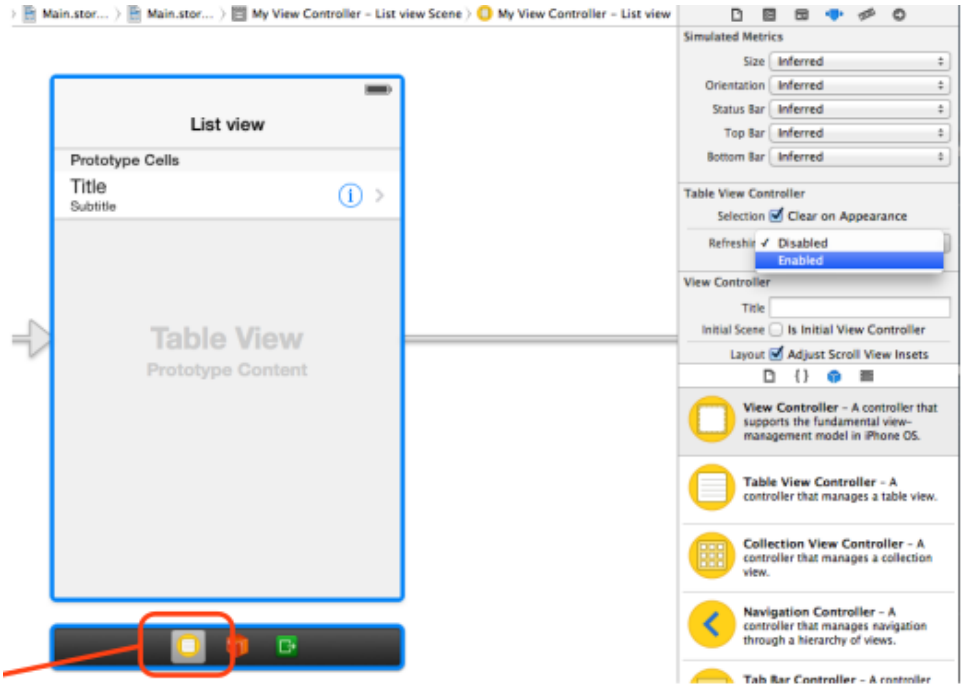
Preparare un successore

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender{
        // Get the new view controller using [segue destinationViewController].
        // Pass the selected object to the new view controller.
        NSIndexPath *indexPath = [self.tableView indexPathForCell:sender];
        if([segue.destinationViewController isKindOfClass:[SomeViewController class]]){
                SomeViewController *vc = (SomeViewController
*)segue.destinationViewController;
                vc.title = [NSString stringWithFormat:@"Detail %d",indexPath.row];
        }
}
```

> Nota
> È buona pratica controllare che il sender sia un `UITableViewCell` utilizzando un `isKindOfClass`

Aggiornare

È possibile impostare il refresh mediante la proprietà `@property UIRefreshControl *refreshControl;`




```
- (IBAction)refreshTableView:(UIRefreshControl *)sender {
        [sender beginRefreshing];
        // ... reload data
        [self.tableView reloadData];
        [sender endRefreshing];
}
```

Web Views

Una vista web è utilizzata per inserire dei contenuti provenienti dal web, caricati mediante URL, all'interno dell'applicazione. Questo tipo di vista supporta la navigazione e muovendosi avanti/indietro nella storia.
La classe che le implementa è `UIWebView`, un oggetto di questa classe può essere trascinato nel controllore della vista mediante storyboard o creato programmaticamente ed aggiunto alla vista. Inoltre può successivamente essere collegato, mediante ctrl-dragging, da un `UIWebView` ad una interfaccia di dichiarazione del controllore.

È possibile configurare la `UIWebView` per ridimensionare automaticamente i contenuti web per riempire lo schermo impostando il proprietà `scalesPagesToFit`.

Caricare i contenuti

loadRequest

```
NSURL *url = [NSURL URLWithString:@"url"];
NSURLRequest *request = [NSURLRequest requestWithURL:url];
[self.webView loadRequest:request];
```

laodHTMLString:baseURL:

```
NSString *html = @"<html><body>Hello world!</body></html>";
[self.webView loadHTMLString:html baseURL:url];
```

UIWebViewDelegate

Una vista web può avere un delegato che può essere utilizzato per tracciare il caricamento dei contenuti web. L'oggetto che funge da delegato deve essere conforme ai protocolli della `UIWebViewDelegate`.

> Nota
> È possibile impostare un vista web come delegato `self.webView.delegate = self;`

La classe di un delegato deve dichiarare di conformarsi al protocollo `UIWebViewDelegate` e implementare i metodi facoltativi

```
@interface WebViewViewController()<UIWebViewDelegate>
```

Se un delegato è stato impostato per una visualizzazione Web, deve essere impostato su nil prima di eliminare l'istanza della visualizzazione Web:

```
- (void)dealloc{
        _webView.delegate = nil;
}
```

Metodi

```
- (BOOL)webView:(UIWebView *)webView
               shouldStartLoadWithRequest:(NSURLRequest *)request
               navigationType:(UIWebViewNavigationType)navigationType
```

Questo metodo viene chiamato prima che una visualizzazione Web inizi a caricare un frame e può essere utilizzato per "intrappolare" il caricamento. Solo se il metodo ritorna `YES` il contenuto è caricato.

```
- (void)webViewDidStartLoad:(UIWebView *)webView
```

Questo metodo viene chiamato dopo che una visualizzazione Web ha avviato il caricamento di un frame.

```
- (void)webViewDidFinishLoad:(UIWebView *)webView
```

Questo metodo viene chiamato dopo che una visualizzazione Web ha terminato il caricamento di un frame.

```
- (void)webView:(UIWebView *)webView
               didFailLoadWithError:(NSError *)error
```

Questo metodo viene chiamato se una visualizzazione Web ha fallito il caricamento