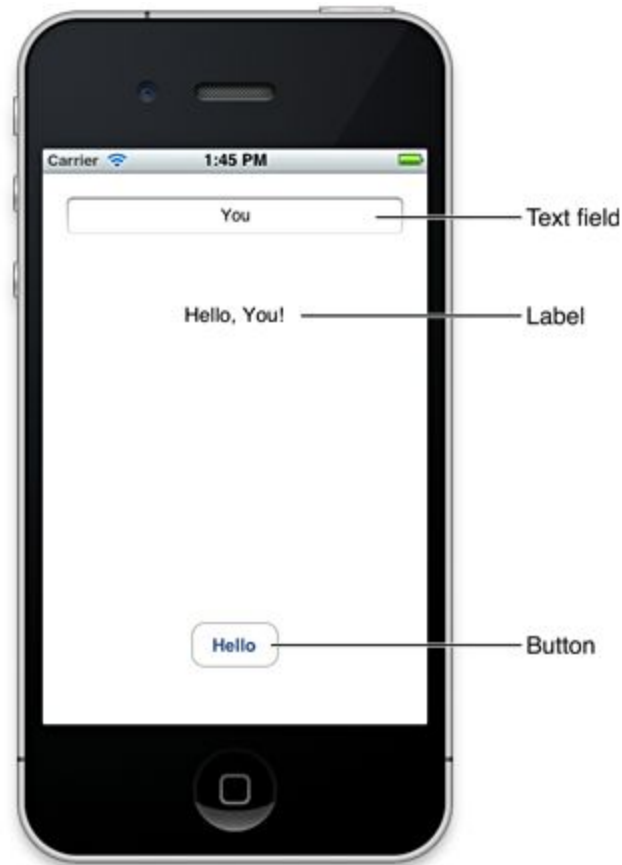


Ensimmäinen iOS Applikaatio

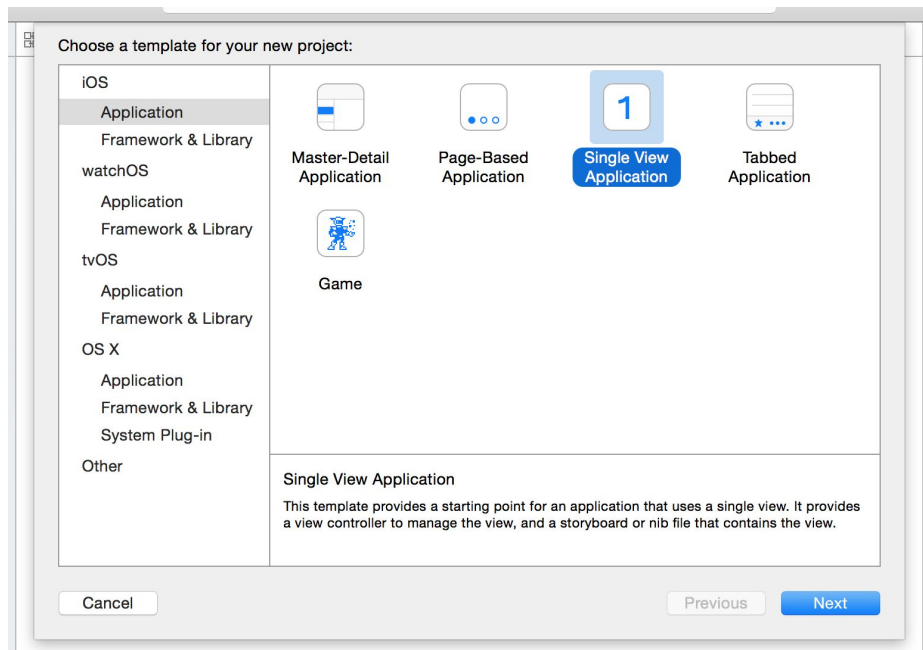
Tässä ensimmäisessä tutoriaalissa tehdään yhden näkymän (Single View) applikaation. Luodaan hieman alustavaa tuntumaa työkaluihin, teknologioihin ja menetelmiin.



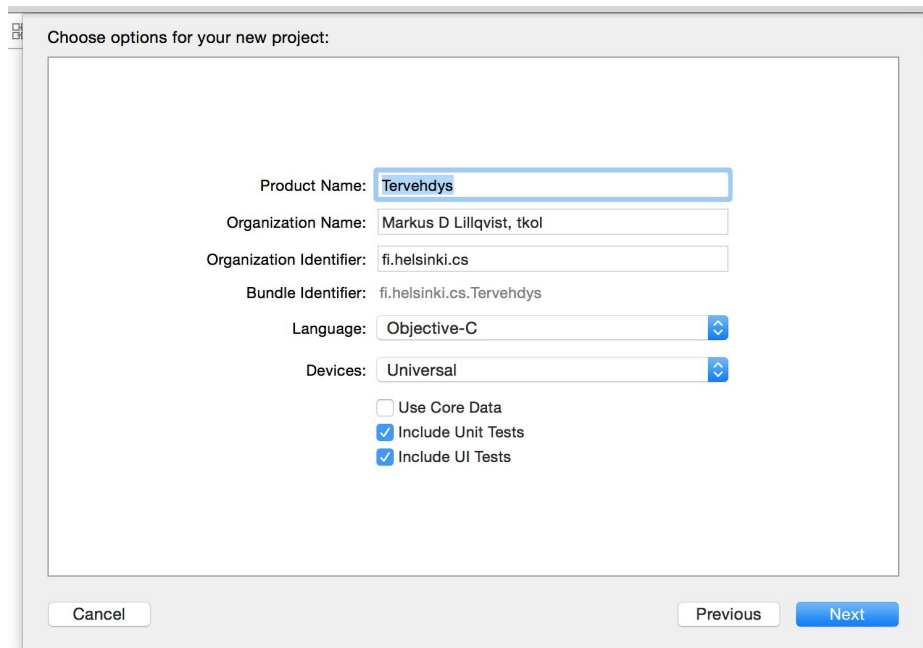
Kuva 1. Saavutamme suunnilleen tämän näköisen lopputuloksen.

Applikaatiossa lisäät tekstikenttään nimesi ja kun painat Hello-nappia, näkymässä ilmestyy tervehdys nimelläsi. Eli, tuttu HelloWorld tyyppinen alku tulossa. Jos tämä jo tuttua mene seuraavaan tutoriaaliin (Toinen iOS Applikaatio).

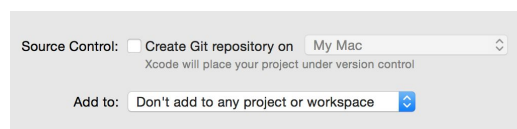
Aloitetaan



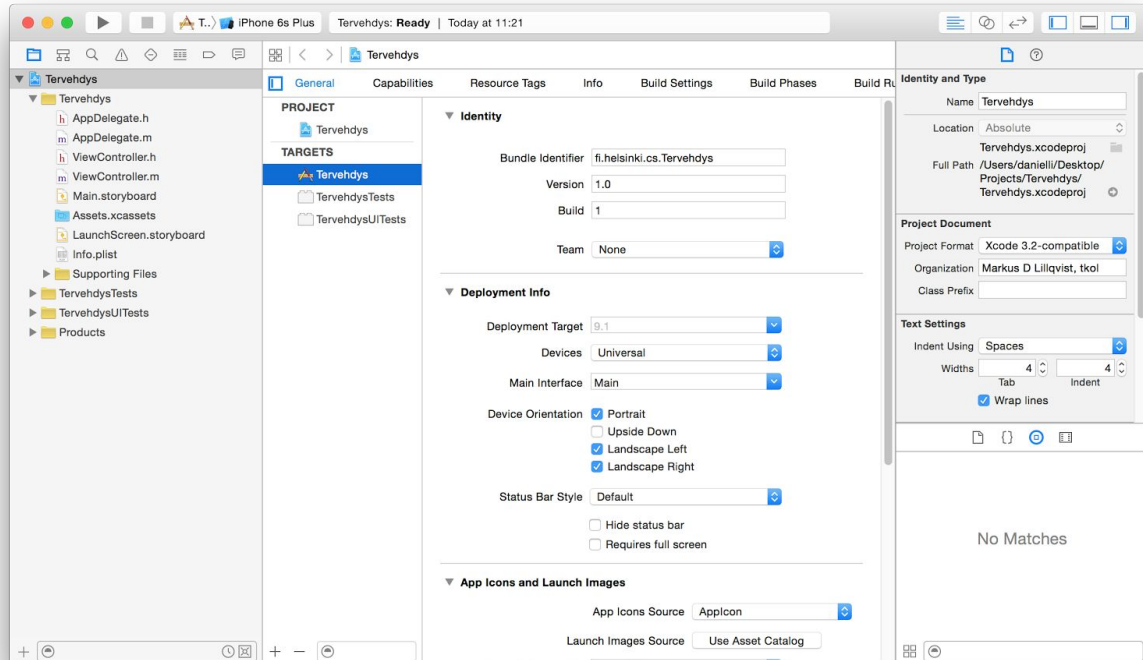
Luo uusi projekti Xcodessa (File > New > New project) valitse **Single View Application**.



Anna projektille nimi. Tallennetaan. Tälle projektille ei tarvitse versionhallintaa (älä siis ruksaa seuraavaa):



Saavutetaan seuraavaan tuloksen:



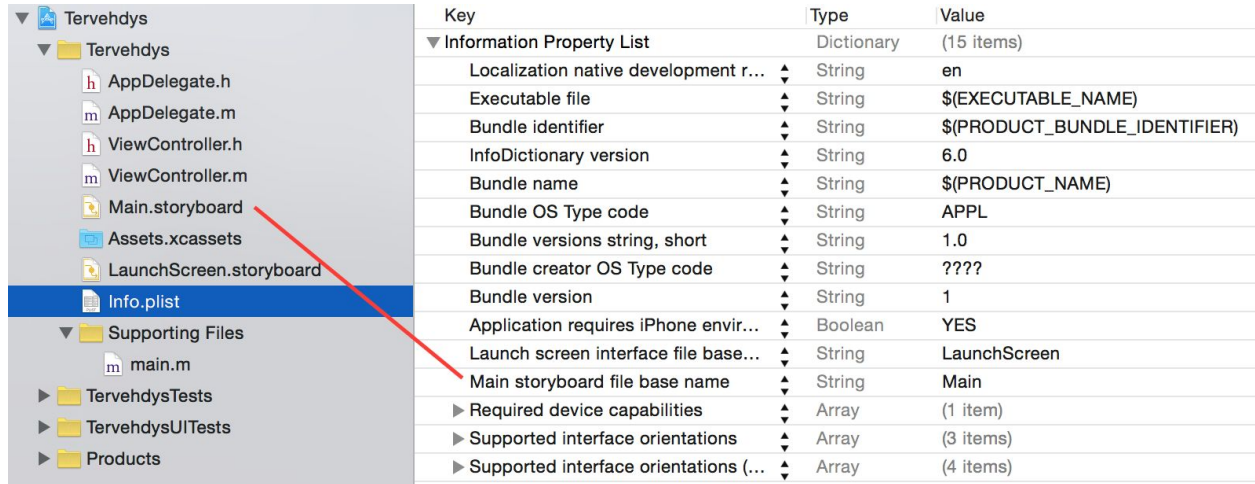
Kuten näkyy, simulaattori iPhone 6s Plus on valittu jo aluksi, joten käynnistä Play-napista (tai Product->Run tai ⌘R). Simulaattori käynnistyy omaan ikkunaan ja näyttää vain tyhjän näytön. Mene takaisin Xcodeen ja paina Stop-nappia (tai Product->Stop tai ⌘.) Sulje simulaattori.

Miten applikaatio käynnistyy? Navigaattorista löytyy projektin tiedostot ja "Supporting Files" kansion (ryhmän) alta löytyy main.m , jonka sisältö on suunnilleen seuraava:

```
1 //
2 // main.m
3 // Tervehdys
4 //
5 // Created by Markus D Lillqvist, tkol on 30/11/15.
6 // Copyright © 2015 Markus D Lillqvist, tkol. All rights reserved.
7 //
8
9 #import <UIKit/UIKit.h>
10 #import "AppDelegate.h"
11
12 int main(int argc, char * argv[]) {
13     @autoreleasepool {
14         return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
15     }
16 }
```

UIApplicationMain kutsu luo mm. AppDelegate instanssin. Jos vilkaiset projektin tiedostoja, niin löydät sieltä mm. AppDelegate tiedoston. Tämä AppDelegate tarjoaa applikaatiolle mm. ikkunan (UIWindow) ja vastaa applikaation tilamuutoksista (sulkeutuu, joutuu taustalle ym).

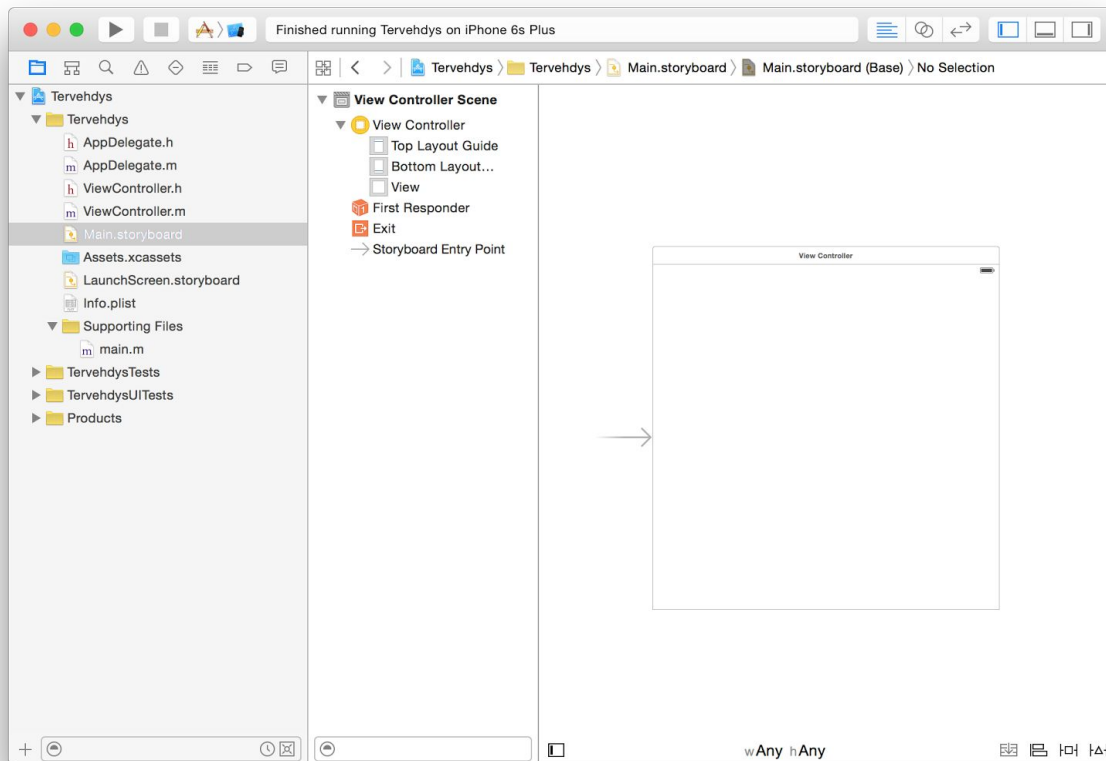
AppDelegate:n lisäksi, UIApplicationMain parsii tiedoston nimeltä Info.plist (löytyy myös projektista). Tämä tiedosto on ns. property list, joka sisältää strukturoidun joukon avain-arvo pareja. Huomaa taas, Info.plist viittaa taas tiedostoon. Tällä kertaa se on Storyboard tiedosto, joka sisältää näkymät (tällä kertaa yhden näkymän).



Key	Type	Value
Information Property List	Dictionary	(15 items)
Localization native development r...	String	en
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1
Application requires iPhone envir...	Boolean	YES
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
Required device capabilities	Array	(1 item)
Supported interface orientations	Array	(3 items)
Supported interface orientations (...)	Array	(4 items)

Kun applikaatio käynnistyy, Storyboardista ladataan ensimmäisen ViewControllerin (eli instanssi joka hallitsee yhden alueen sisällöstä). Taas näemme, että projektissamme on kyseinen ViewController tiedosto. Meidän ViewController hallitsee yhtä näkymää (eng. view), joka on yksi piirrettävä alue näytöllä. Näkymällä voi olla ali-näkymiä (eng. subviews), näin yhdellä ViewControllerilla voi olla monta näkymää ns. näkymä-hierarkiassa. Meidän applikaatioon tulee kolme ali-näkymää, tekstikenttä, etiketti/leima (eng. label) ja nappi.

Tarkastellaan Storyboardin sisältöä. Nuoli (eli Storyboard Entry Point) joka osoittaa näkymään (Storyboardissa näkymä tunnetaan nimellä Scene) osoittaa ensimmäiseen näkymään. Tämä on se joka latautuu ensimmäisenä. Tavallisesti ensimmäistä näkymää vastaa ensimmäinen ViewController.

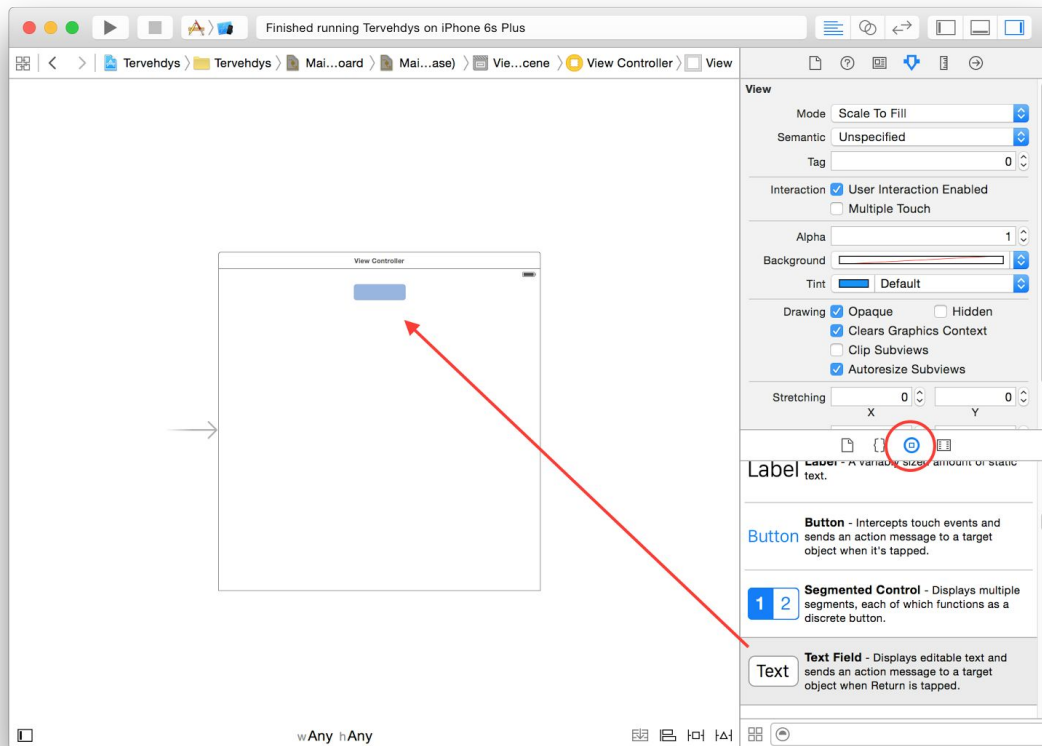


Kuten näkyy hahmottelu-näkymässä (outline view) View Controller Scene sisältää View Controllerin lisäksi, First Responder, Exit ja Storyboard Entry Point. View Controllerin alta löytyy Top ja Bottom Layout Guide, jotka avustaa mm. Auto Layout moottoria (kun kääntelee laitetta ja elementit ymmärtää löytää oikean paikkansa). Kun lisää käyttöliittymä-elementtejä kannattaa aina huomata että se kiinnittyy jonku Guiden mukaan. View on itse näkymä, eli se valkoinen mitä oli simulaattorin näytöllä kun aikasemmin käynnistettiin applikaation. First Responder on paikanpitäjä (eng. placeholder) oliolle, jolle kaikki tapahtumat pitäisi mennä ensimmäisenä. Tavallisesti kun jätetään yhden lapsi-näkymän, niin siirrytään takaisin vanhempi-näkymään, Exit on paikanpitäjä joka mahdollistaa, että siirrytään mielivaltaiseen näkymään.

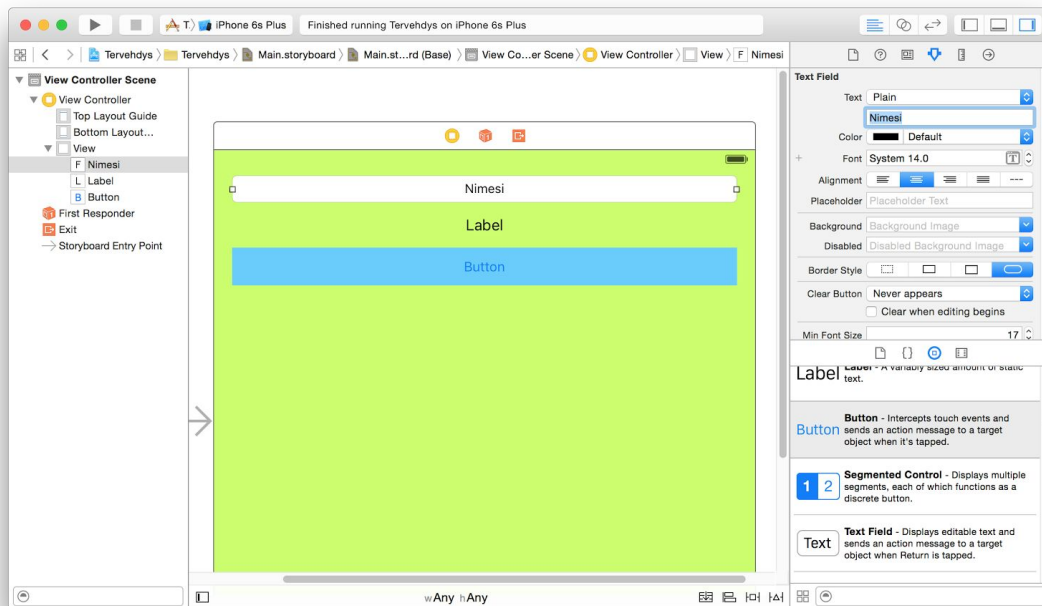
Kun Storyboard lataa näkymän, niin se luo instanssin siitä View Controllerista, joka on vastuussa näkymästä. Huomaa, että ikkuna (UIWindow) ei ikinä sijaitse Storyboardissa.

Muokataan näkymää

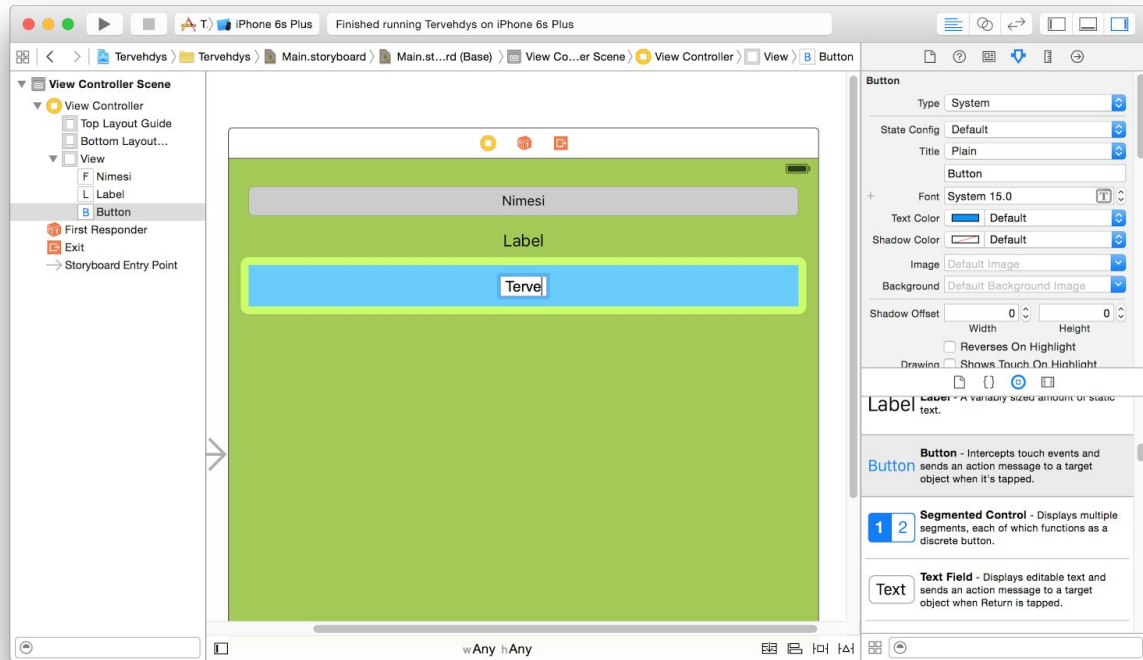
Lisätään käyttöliittymä-elementtejä vetämällä niitä näkymään Olio-kirjastosta. Aloita lisäämällä näkymään tekstikentän. Huomaa, ennen kuin asetat elementin, niin Interface Builder tarjoaa avustusta sinisellä katkoviivalla. Aseta ja muokkaa elementti haluttuun paikkaan ja kokoiseksi.



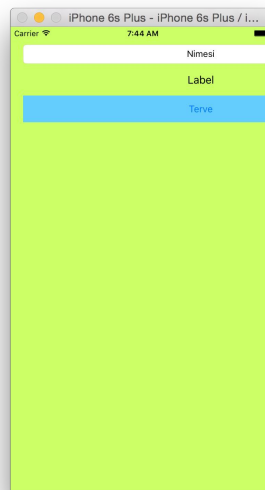
Lisää vielä leiman (eli label) ja napin (vaihda myös näkymän taustaväriä jos huvittaa).



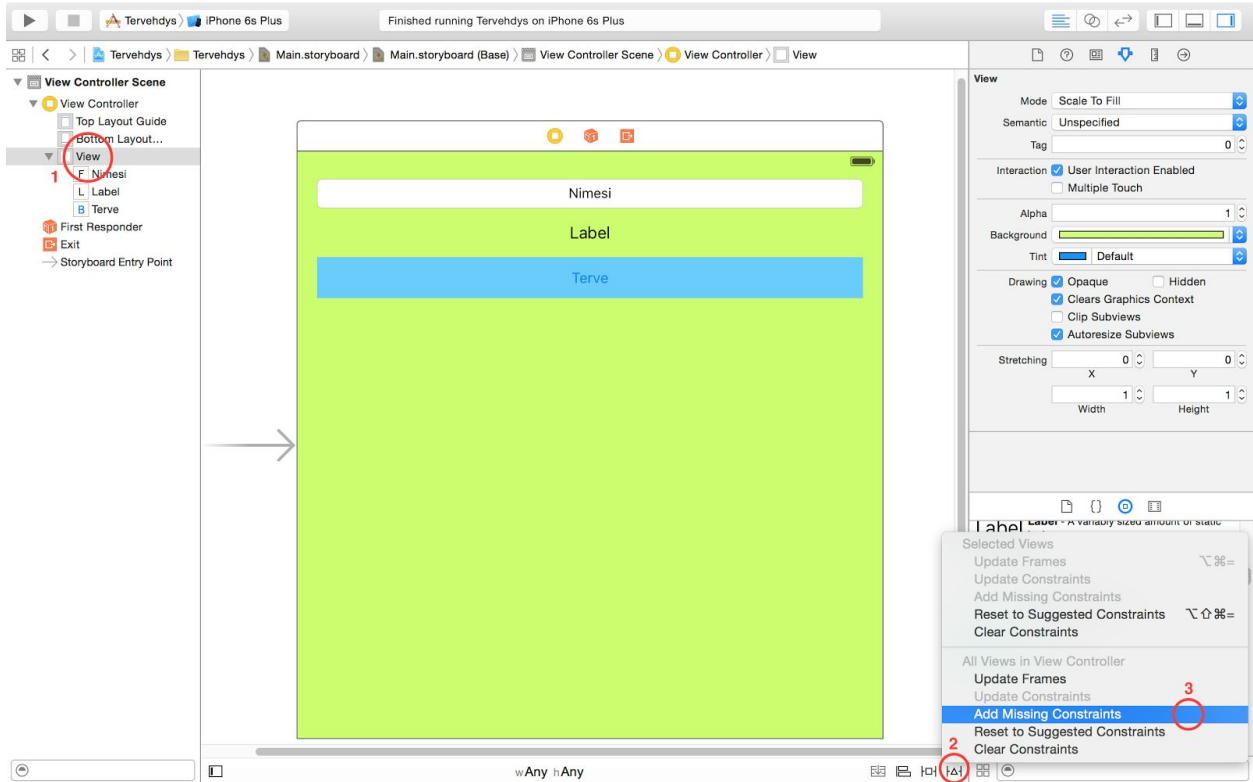
Olemme lisänneet halutut elementit. Muokataan vähän sisältöä, antamalla tekstikentälle paikanpitäjä-tekstin (esimerkissä nähtävänä "Nimesi") ja valitaan tekstille keskitetty "Alignment". Muokataan vielä muiden elementtien sisältöä, napin tekstin voi muokata näpdyttämällä sitä kahdesti.



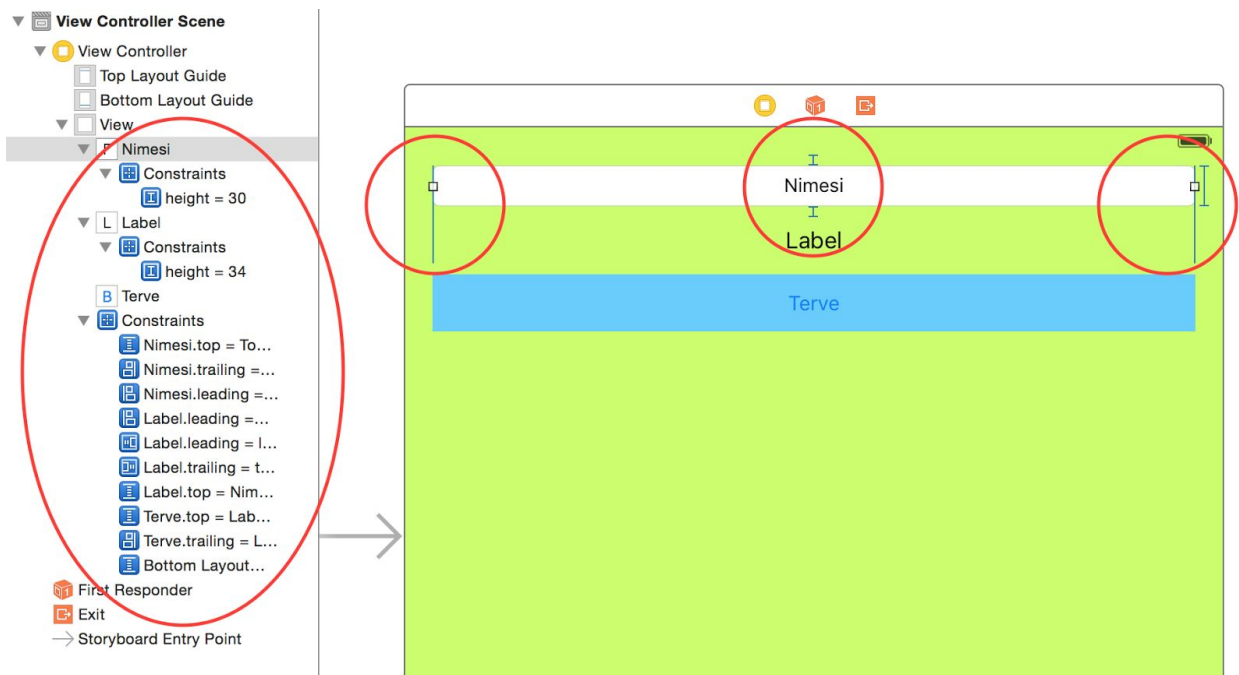
Tarkistetaan tuloksen käynnistämällä simulaattorin. Tulos ei oikein vastaa odotuksia.



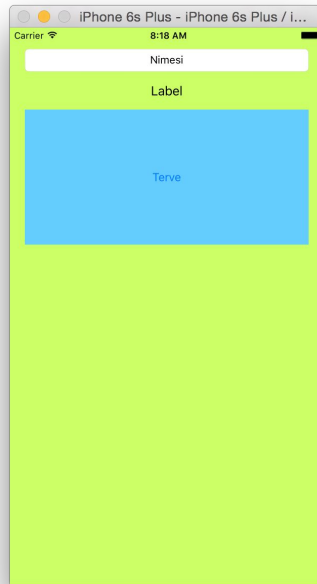
Meidän täytyy lisätä näkymälle rajoitteita. Lisätään ne seuraavasti: valitse näkymä (1), napsauta “Resolve Auto Layout Issues” (2) ja valitse “Add Missing Constraints” (3).



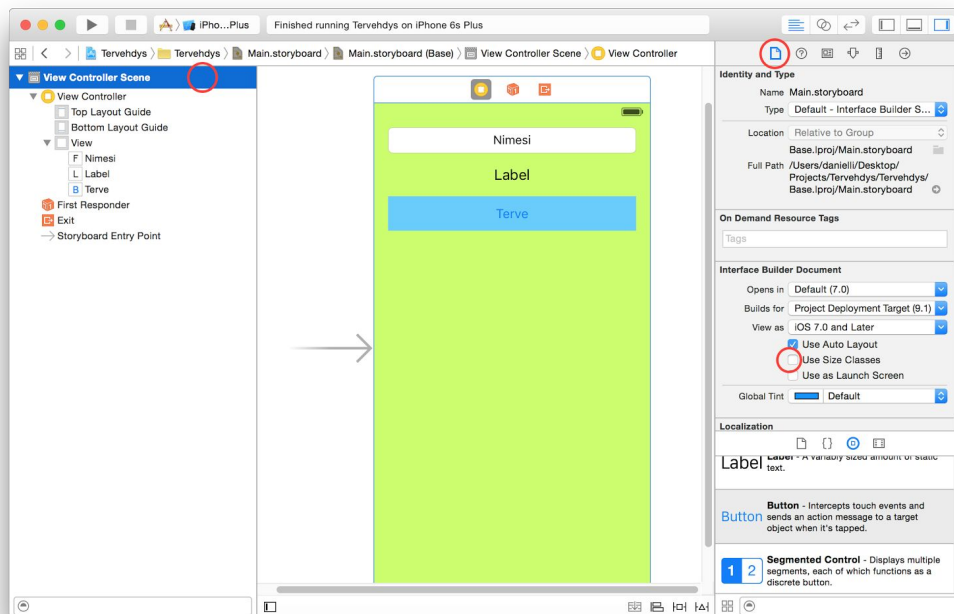
Tulos on seuraava. Rajoitteet näkyvät hahmottelu-näkymässä ja näkymässä. Jokaista voi hienosäätää jos tarvetta.



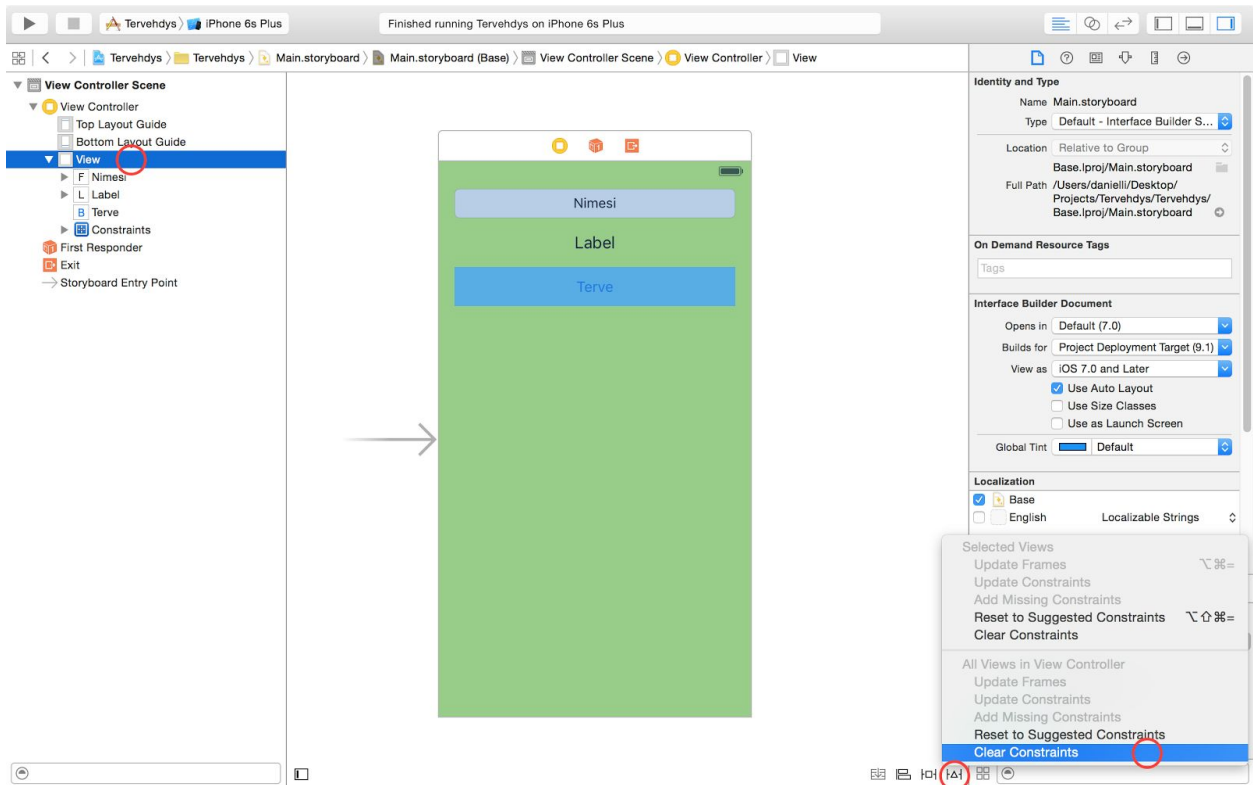
Vilkaistaan taas tulos simulaattorissa. Aika hyvä paitsi tuo jättimäinen nappi. Korjataan se.



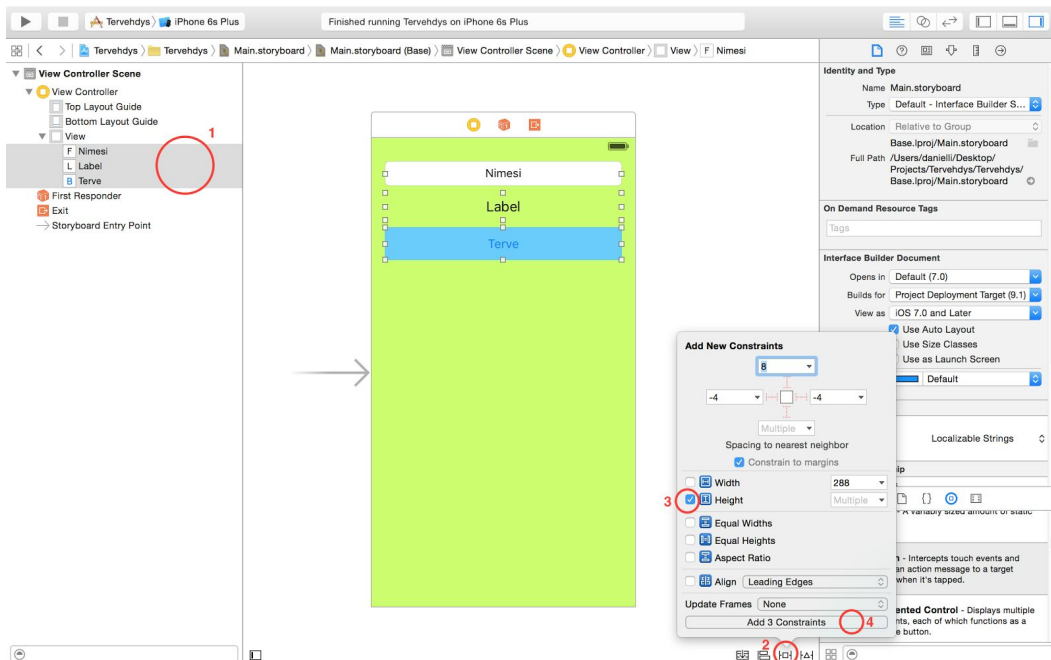
Storyboardissa on oletuksena käytössä ns. Size Classes, joka mahdollistaa että käytetään samaa Storyboard tai XIB tiedostoa kaikille laitteille (monta eri kokoista näyttöä). Jos napsauttaa pois Size Classes täytyy valita tekeekö Storyboardin rajoitteet ja määrytykset toimimaan iPhoneille tai iPadille. Tehdään muutos ja napsautetaan pois Size Classes toimimasta tälle projektille. Size Classes asetus löytyy näin:



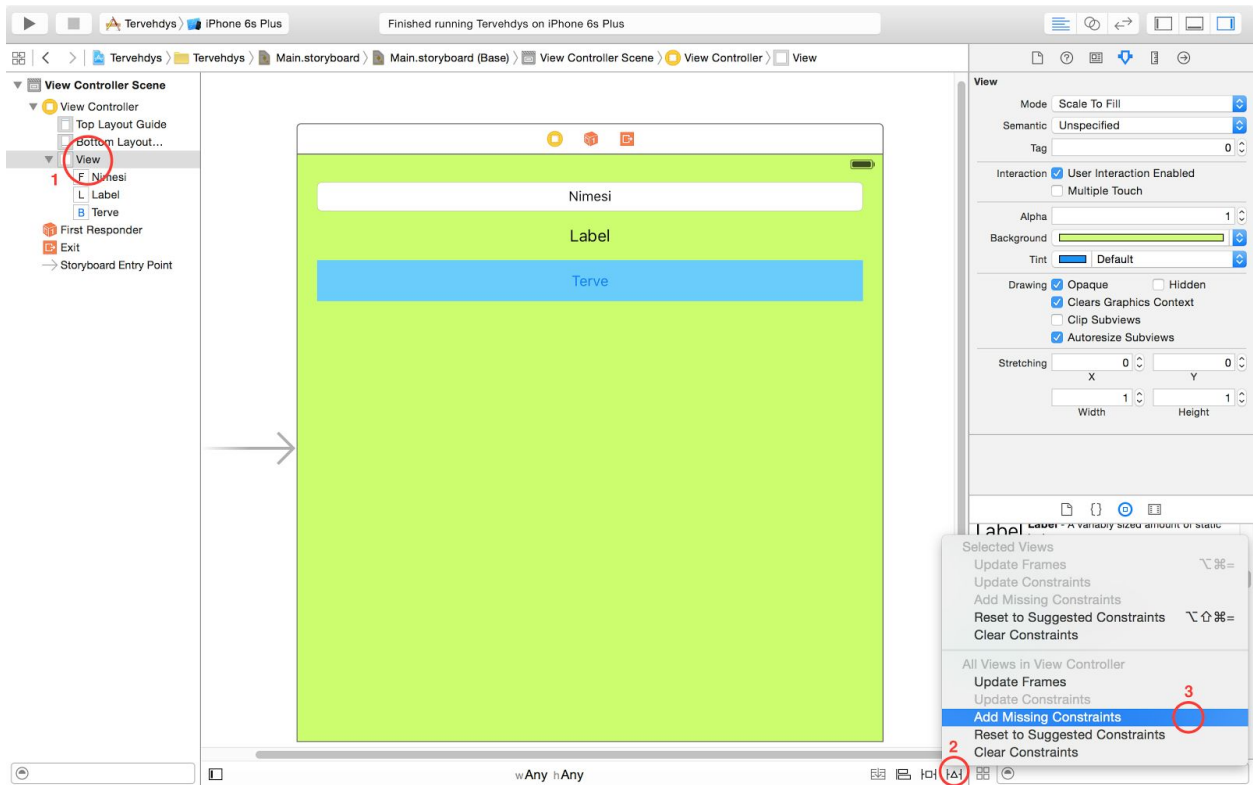
Korjataan se iso nappi. Aloitetaan poistamalla rajoitteet.



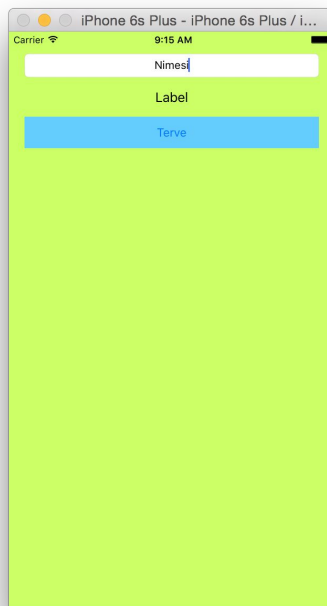
Lisätään meidän elementeille korkeus-rajoitteet. Valitse ensin kaikki elementit (1), napsauta "Pin" (2), valitse "Height" (3) ja lopuksi lisää rajoitteet "Add 3 Constraints" (4).



Nyt lisätään vielä muut puuttuvat rajoitteet, jotka otimme pois aikasemmin. Eli lisää ne taas “Resolve Auto Layout Issues” (2) ja valitse “Add Missing Constraints” (3).

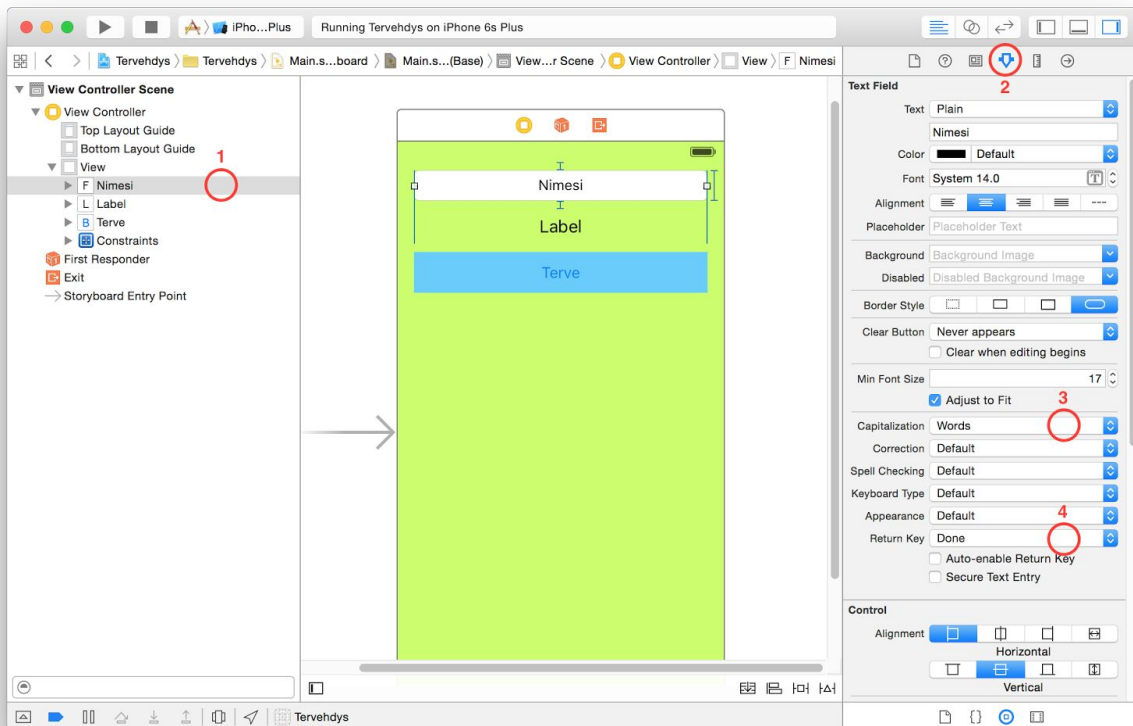


Nyt kun ajetaan saamme halutun näköisen käyttöliittymän.



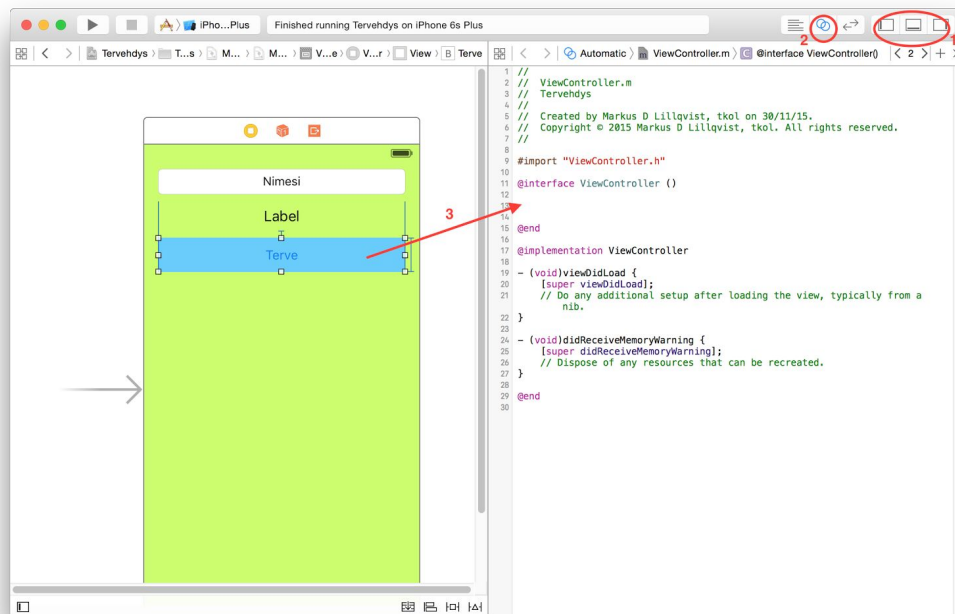
Auto Layoutista on valtavasti hyötyä, jos ongelmia niin sen seikkoihin voi tutustua esimerkiksi [täältä](#). Toinen osa samaa materiaalia käsittelee [rajoitteet](#).

Muokataan vielä tekstikenttää seuraavasti. Valitse tekstikenttä (1), “Attributes Inspector” (2) alta löytyy hieno määrä asetuksia. Muuta “Capitalization”:"Words" (3) ja “Return Key”:"Done" (4).

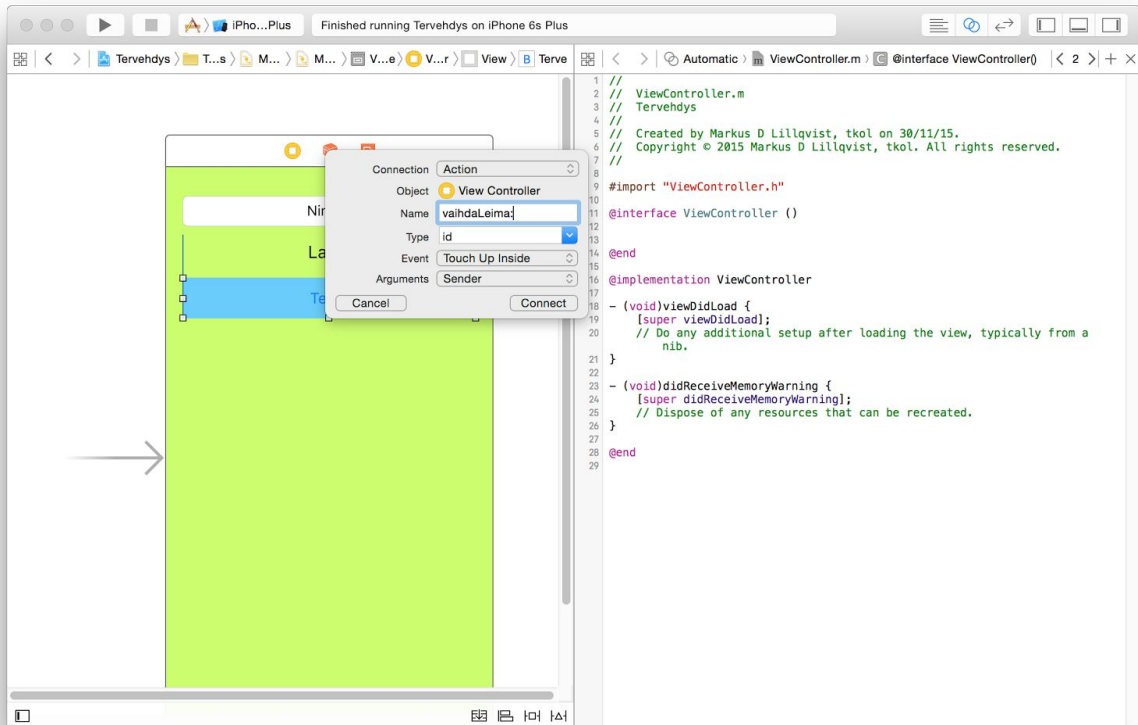


Tapahtumaa napille

Olemme valmiita tutustumaan ViewController-tiedoston sisältöön. Piilota muita palikoita (1), valitse "Assistant Editor" (2), niin saat seuraavan näköisen näytön. Paina nyt Terve-nappia Control painike pohjassa ja vedä @interface osaan koodia.



Saat seuraavaan valikoiman esille. Muuta vaihtoehdot samaksi kuin alla, eli "Connection": "Action", "Name": "vaihdaLeima:" ja "Type": "id", paina Connect.

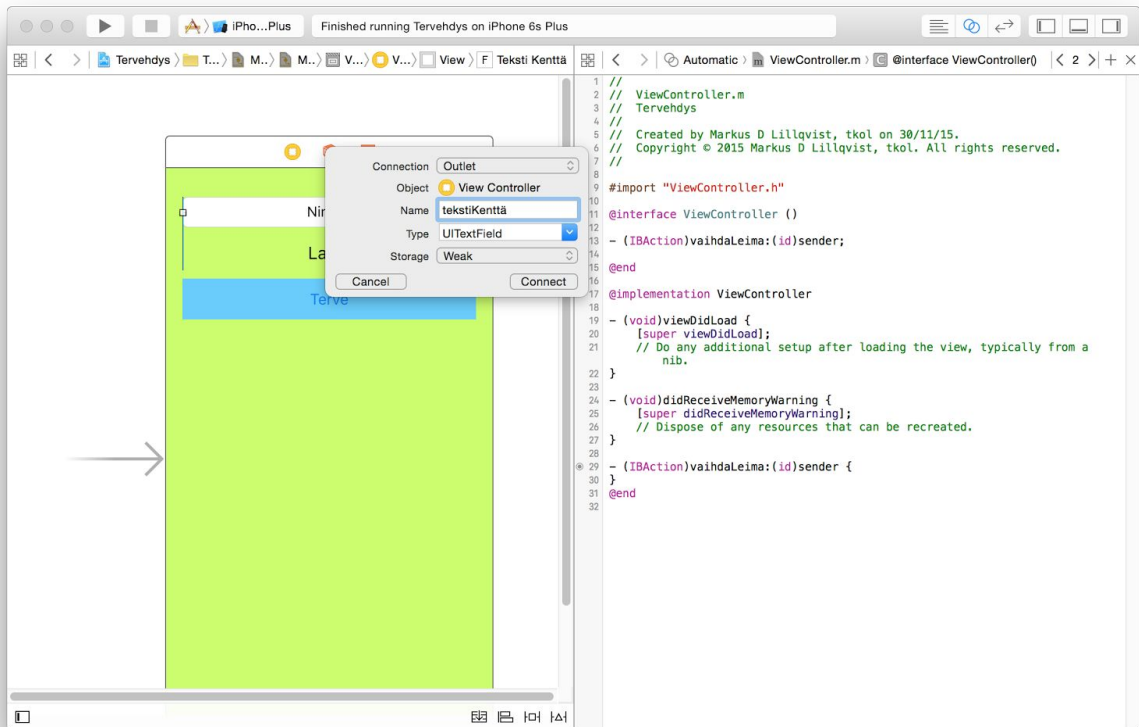


Kaksi lisäystä ilmestyi koodiin. @interface puolelle lisättiin metodin signatuuri ja @implementation puolelle metodeille lyhkäinen implementaatio-pätkä.

```
9 #import "ViewController.h"
10
11 @interface ViewController ()
12
13 - (IBAction)vaihdaLeima:(id)sender;
14
15 @end
16
17 @implementation ViewController
18
19 - (void)viewDidLoad {
20     [super viewDidLoad];
21     // Do any additional setup after loading the view, typically from a
22     // nib.
23 }
24
25 - (void)didReceiveMemoryWarning {
26     [super didReceiveMemoryWarning];
27     // Dispose of any resources that can be recreated.
28 }
29
30 - (IBAction)vaihdaLeima:(id)sender {
31 }
32 @end
```

Yhteyksiä leimalle ja tekstikentälle

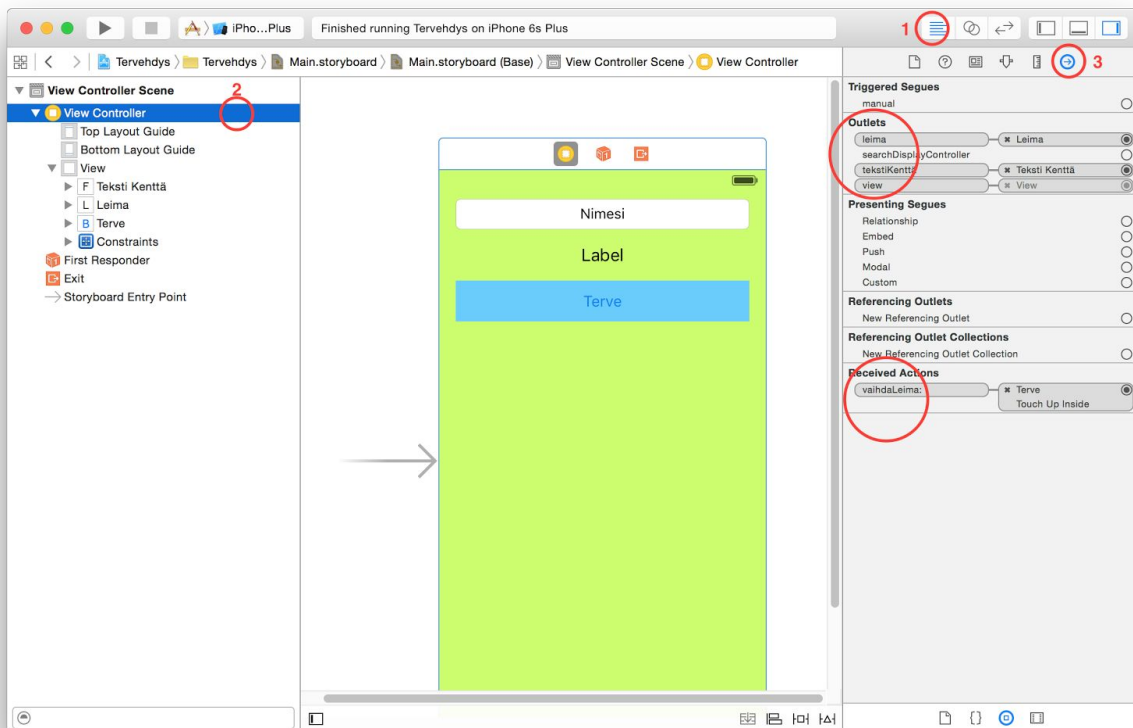
Samalla tavalla kun luotiin yhteys napista koodiin, luodaan tekstikentälle ja leimalle ns. Outletit.



Olemme nyt luonut yhteykset meidän elementeille.

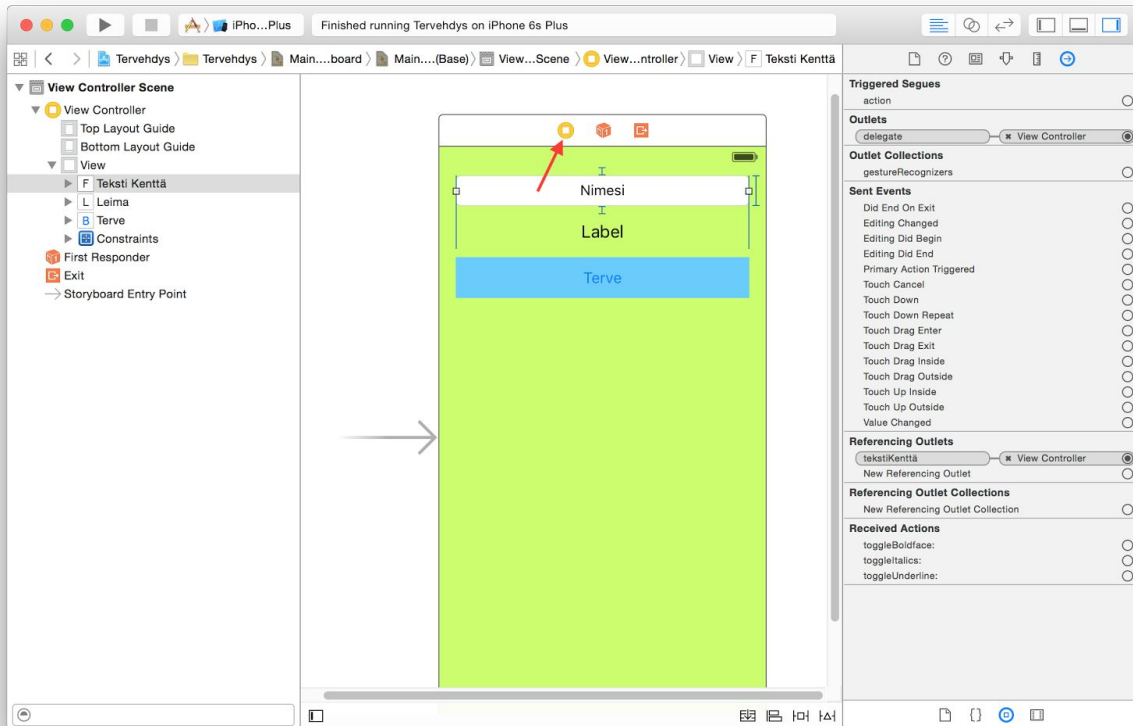
```
1 //
2 // ViewController.m
3 // Tervehdys
4 //
5 // Created by Markus D Lillqvist, tkol on 30/11/15.
6 // Copyright © 2015 Markus D Lillqvist, tkol. All rights reserved.
7 //
8
9 #import "ViewController.h"
10
11 @interface ViewController ()
12
13 @property (weak, nonatomic) IBOutlet UITextField *tekstiKenttä;
14 @property (weak, nonatomic) IBOutlet UILabel *leima;
15 - (IBAction)vaihdaLeima:(id)sender;
16
17 @end
18
19 @implementation ViewController
20
21 - (void)viewDidLoad {
22     [super viewDidLoad];
23     // Do any additional setup after loading the view, typically from a
24     // nib.
25 }
26
27 - (void)didReceiveMemoryWarning {
28     [super didReceiveMemoryWarning];
29     // Dispose of any resources that can be recreated.
30 }
31
32 - (IBAction)vaihdaLeima:(id)sender {
33 }
34 @end
```

Vilkaistaan vielä tulosta Interface Builderistä. Voit sulkea “Assistant editor” valitsemalla “Standard editor” (1). Valitse View Controller (2) ja “Connections inspector” (3).

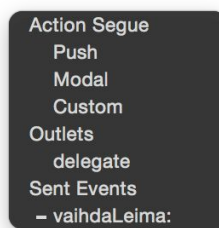


Kuten näet, Outletit ja Actionit ovat näkyvissä ja muokattavissa myös täältä.

Seuraavaksi, lisätään delegaatti tekstikentälle, koska kun painetaan Done näppäimistöä, niin kenttä lähettää delegaatile viestin. Meidän delegaatti on ViewController. Eli, Control-vedä taas tekstikentästä keltaiseen ikoniin (joka on meidän ViewController).



Seuraava valikko ilmestyy:



Valitse delegate ja yhteys on luotu (voit tarkistaa taas Connections Inspector).

Käynnistä taas simulaattori. Näet, että jos painat tekstikenttää, niin näppäimistö tulee esille, mut ei mene minnekään vaikka painat Done-nappia. Samoin itse Terve-nappi ei vielä tee mitään järkevää. Nyt on aika implementoida tapahtumat.

Implementoidaan ViewController

Osat mitkä pitää implementoida on mm. joku talletus tila (ominaisuus meidän luokassa) nimelle, vaihdaLeima: -metodi ja varmistaa, että näppäimistö menee piiloon kun painaa Done.

Lisätään nimi luokan ominaisuudeksi (ViewController.h tiedostoon):

```
@property (copy, nonatomic) NSString *nimi;
```

```
1 //
2 // ViewController.h
3 // Tervehdys
4 //
5 // Created by Markus D Lillqvist, tkol on 30/11/15.
6 // Copyright © 2015 Markus D Lillqvist, tkol. All rights reserved.
7 //
8
9 #import <UIKit/UIKit.h>
10
11 @interface ViewController : UIViewController
12
13 @property (copy, nonatomic) NSString *nimi;
14
15 @end
```

Tässä tapahtuu taustalla ainakin seuraavat asiat. Ominaisuudelle luodaan automaattisesti get ja set metodit. Vaikka niitä ei ole näkyvässä missään on seuraavat metodit käytettävissä:

- (NSString *)nimi;
- (void)setNimi:(NSString *)newNimi;

@property luo nimelle instanssi muuttujan nimellä `_nimi`

Nyt, lisätään implementaatioon (ViewController.m tiedostoon), vaihdaLeima: metodin:

```
- (IBAction)vaihdaLeima:(id)sender {
    self.nimi = self.tekstiKenttä.text;
    NSString *nimiLeima = self.nimi;
    if ([nimiLeima length] == 0)
        nimiLeima = @"Maailma";
    NSString *tervehdys = [[NSString alloc] initWithFormat:@"Terve, %@", nimiLeima];
    self.leima.text = tervehdys;
}
```

Tässä pätkässä otetaan tekstin tekstikentästä ja laitetaan se äsken luodun nimi-ominaisuuden arvoksi. Seuraavaksi luodaan nimiLeima muuttuja (tyyppiä NSString), joka viittaa nimeen. Jos leiman pituus on nolla, annetaan nimiLeima viittaa @"Maailma" NSString literaaliin. Luodaan

uusi NSString nimellä tervehdys, jonka alustetaan omalla formaatilla ja nimiLeima-muuttujan sisällöllä. Lopuksi lisätään tervehdys leimaan.

Vaikka nyt ajetaan applikaation simulaattorissa, niin näppäimistö ei vieläkään suostu poistumaan vaikka napsautetaan Done.

Korjaan vielä tämä. Lisätään ensin meidän ViewController (.h) noudattamaan tekstikentän protokollaa kun se on meidän delegaatti. Rivi 11 lisäys: <UITextFieldDelegate>

```
1 //
2 // ViewController.h
3 // Tervehdys
4 //
5 // Created by Markus D Lillqvist, tkol on 30/11/15.
6 // Copyright © 2015 Markus D Lillqvist, tkol. All rights reserved.
7 //
8
9 #import <UIKit/UIKit.h>
10
11 @interface ViewController : UIViewController <UITextFieldDelegate>
12
13 @property (copy, nonatomic) NSString *nimi;
14
15 @end
```

Implementoidaan lopuksi vielä kyseinen metodi, joka vastaa Return (Done, ym) napin tapahtumiin. resignFirstResponder kutsu antaa pois tekstikentän First Responder statuksen ja näppäimistö menee piiloon.

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField {
    if (textField == self.tekstiKenttä)
        [textField resignFirstResponder];
    return YES;
}
```

Löytyikö korjattavaa? Ota yhteyttä ohjaajaan.

This material is reworked from Apple iOS legacy documentation made current for Xcode 7.x and iOS 9.x
https://developer.apple.com/legacy/library/reference/library/GettingStarted/RoadMapiOS-Legacy/chapters/RM_YourFirstApp_iOS/Articles/00_Introduction.html#/apple_ref/doc/uid/TP40011343-TP40012323-CH1-SW1
Copyright © 2015 Apple Inc. All Rights Reserved