

Praktikum iOS-Entwicklung

Wintersemester 2015/16

Prof. Dr. Linnhoff-Popien

Florian Dorfmeister, Marco Maier



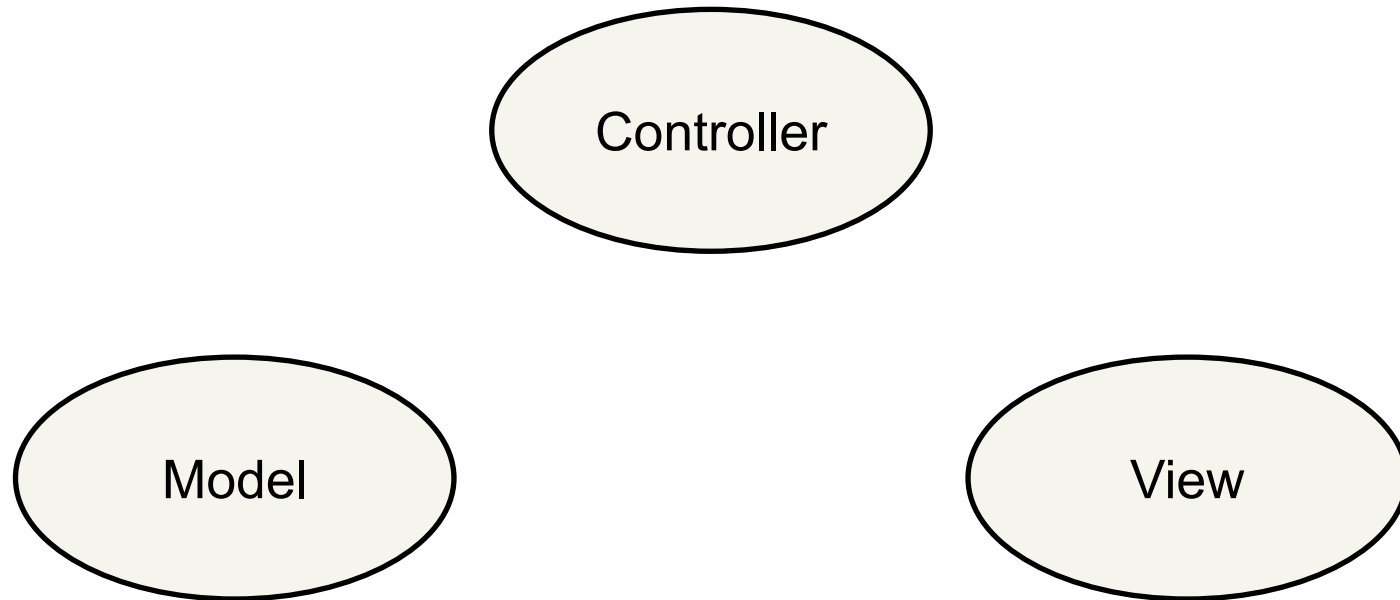
Gemeinsames Themen-Brainstorming am 2.12.!

Wir suchen Ideen für die Praxisphase!

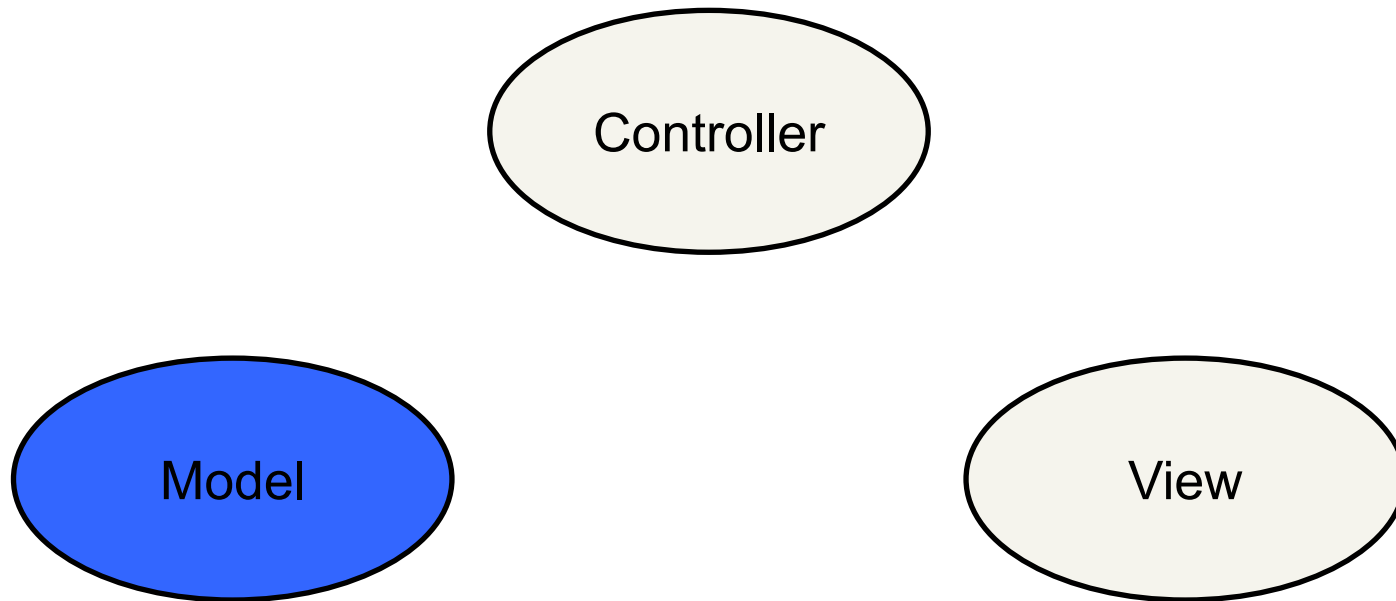
Das heißt:

- Eure Ideen sind gefragt!
- Vorstellen der Ideen in 5-minütigen Präsentationen
- Vergabe der Themen mit Beginn der Programmierphase

MODEL-VIEW-CONTROLLER

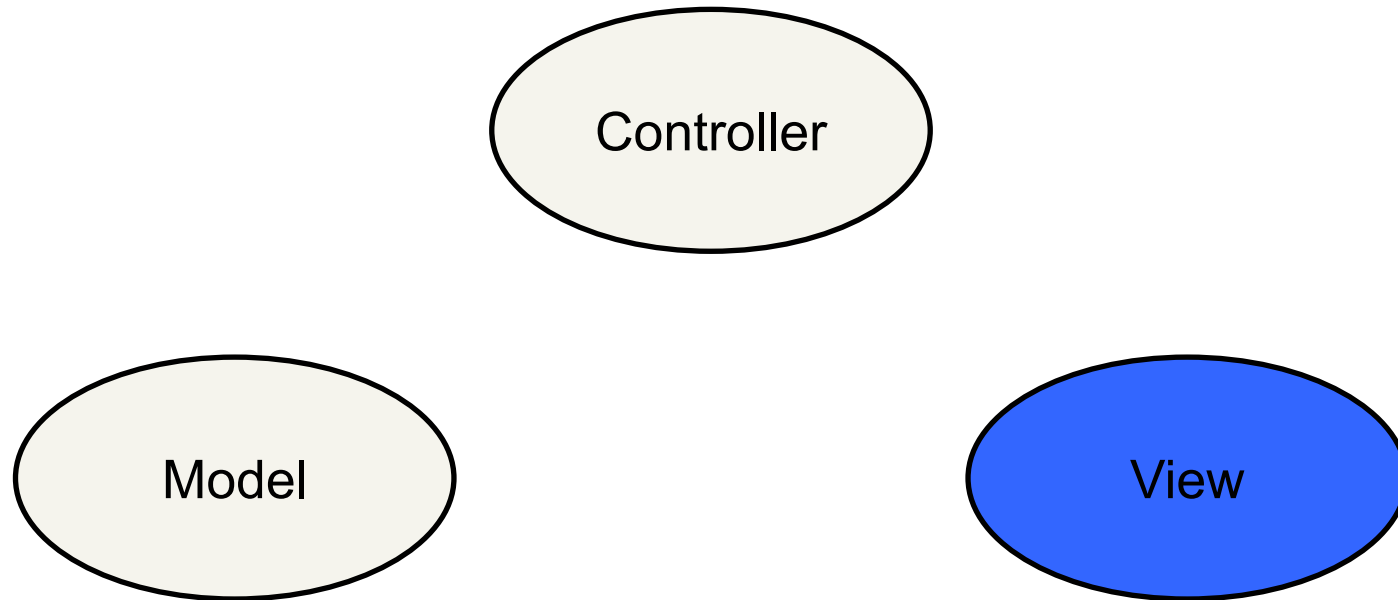


- Programmierparadigma zur Strukturierung von Source Code
- Trennung dient der Wiederverwendbarkeit/Austauschbarkeit von Code
- Aufteilung von Objekten in drei unterschiedliche Gruppen



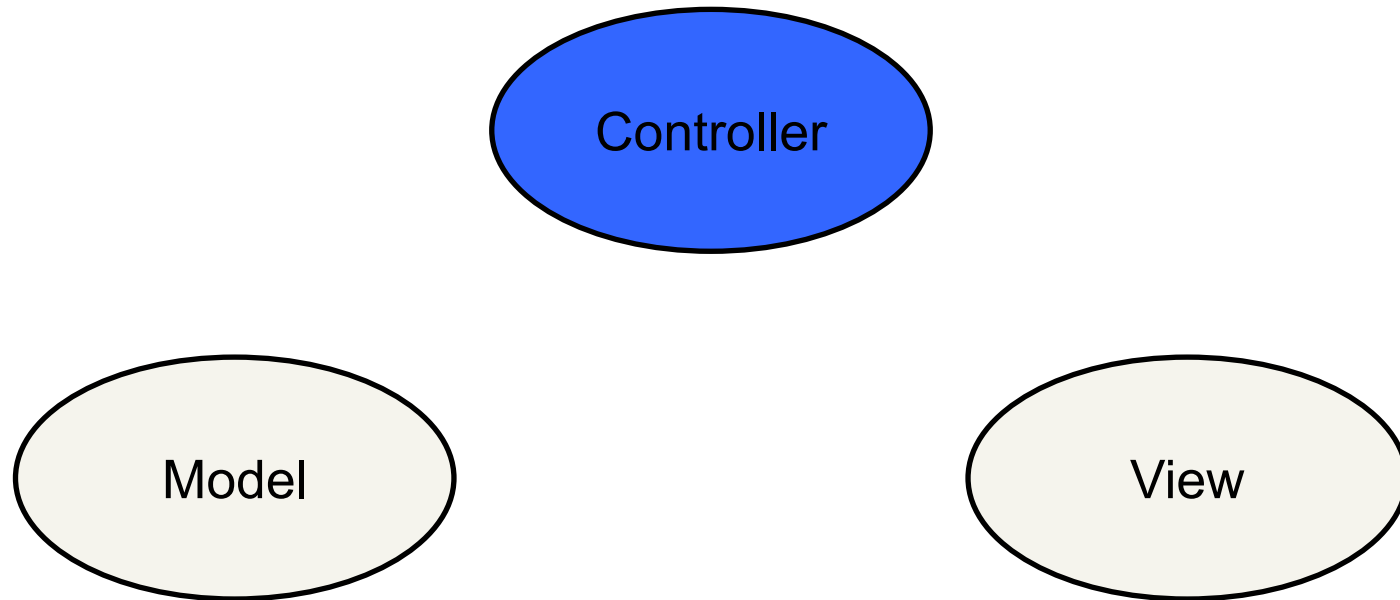
Model: Worum handelt es sich bei der Anwendung?

- Enthält Daten bzw. Datenmodell
- Ist unabhängig von der eigentlichen Darstellung!



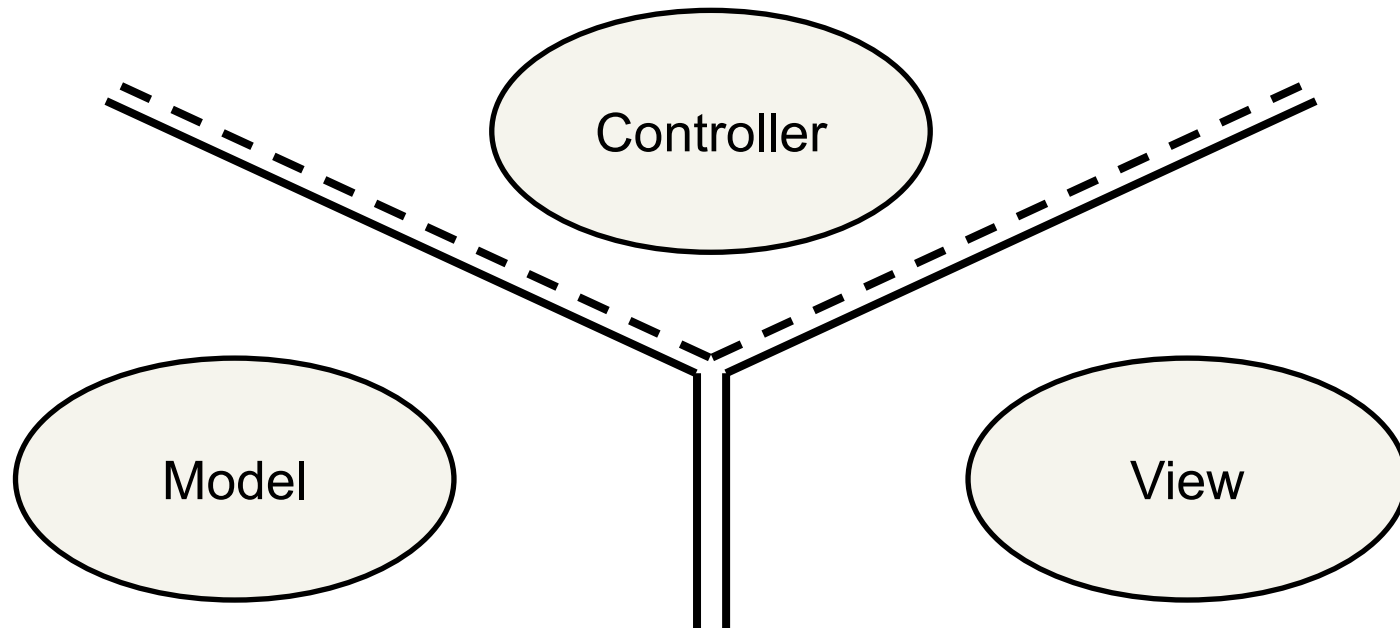
View: Darstellung des Model

- Schnittstelle zum Benutzer (Interaktion)
- Keine Verarbeitung von Daten!

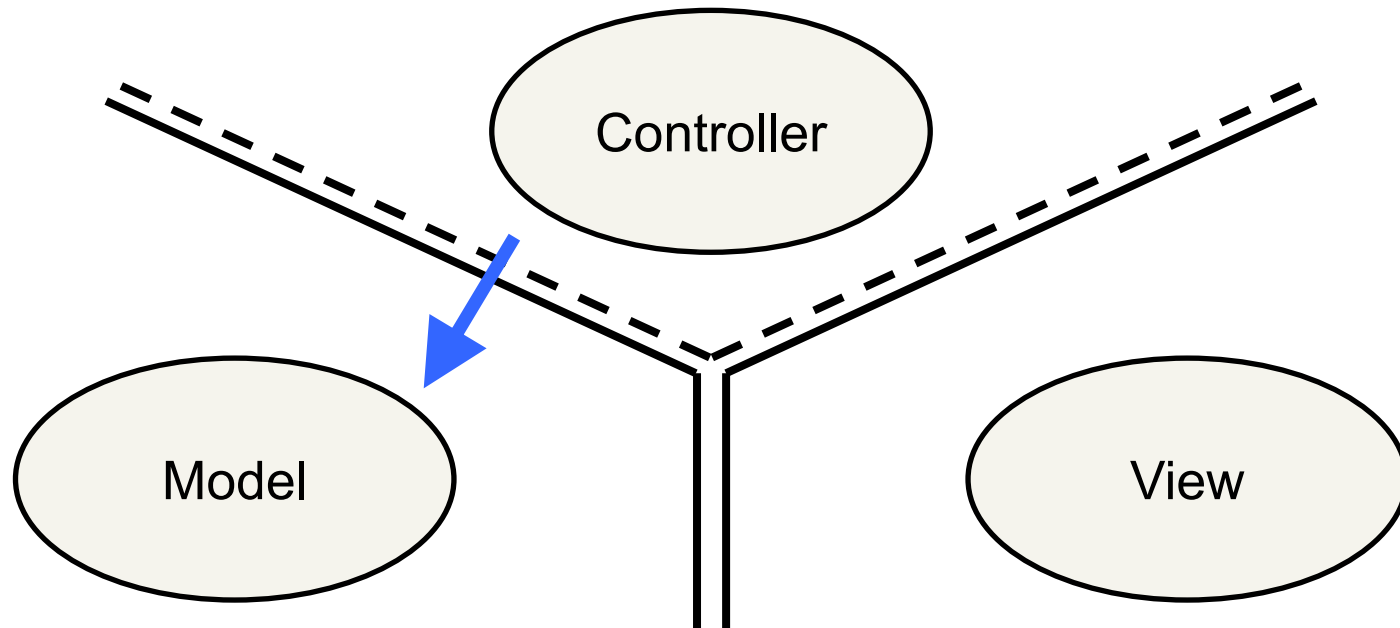


Controller: Kontrolliert die Präsentation des Model gegenüber dem Nutzer

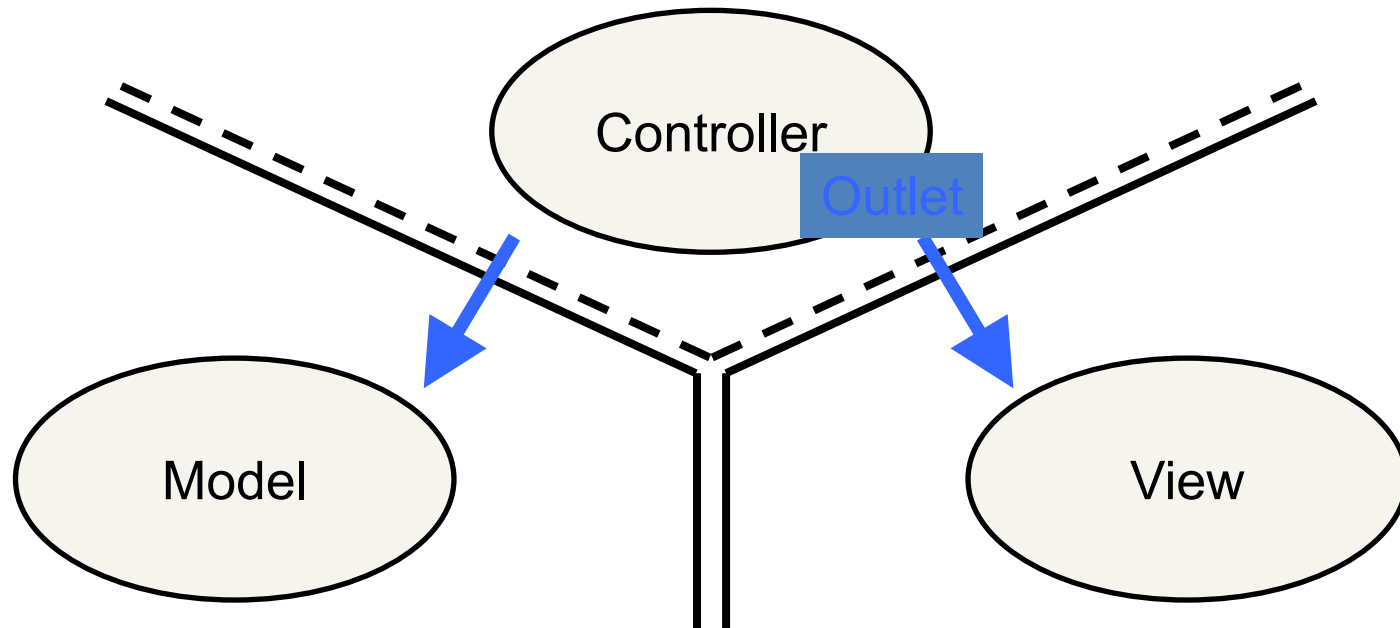
- Vermittlung zwischen Datenmodell und Darstellung (Logik der Darstellung!)
 - Auswertung von Benutzerinteraktionen (View)
 - Manipulation von Daten (Model)
 - Zu jeder View existiert genau ein Controller



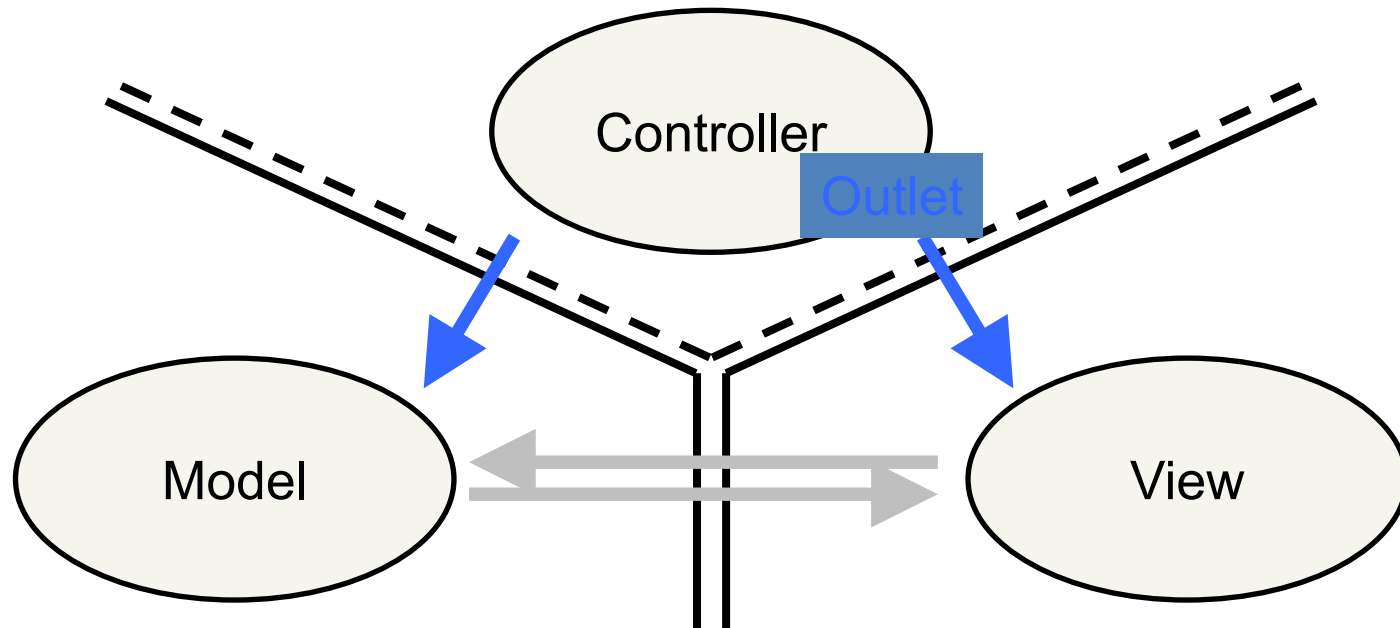
Welche Art der Kommunikation zwischen den drei Gruppen ist erlaubt?



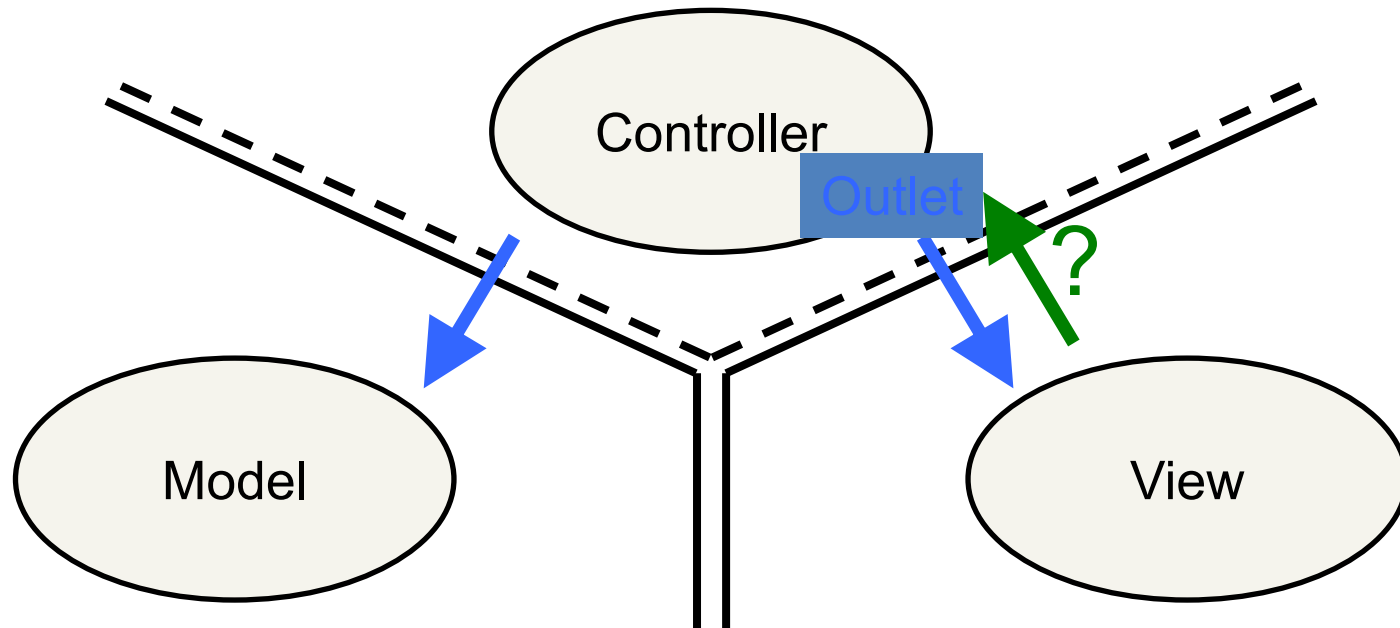
Controller kann immer direkt auf sein Model zugreifen



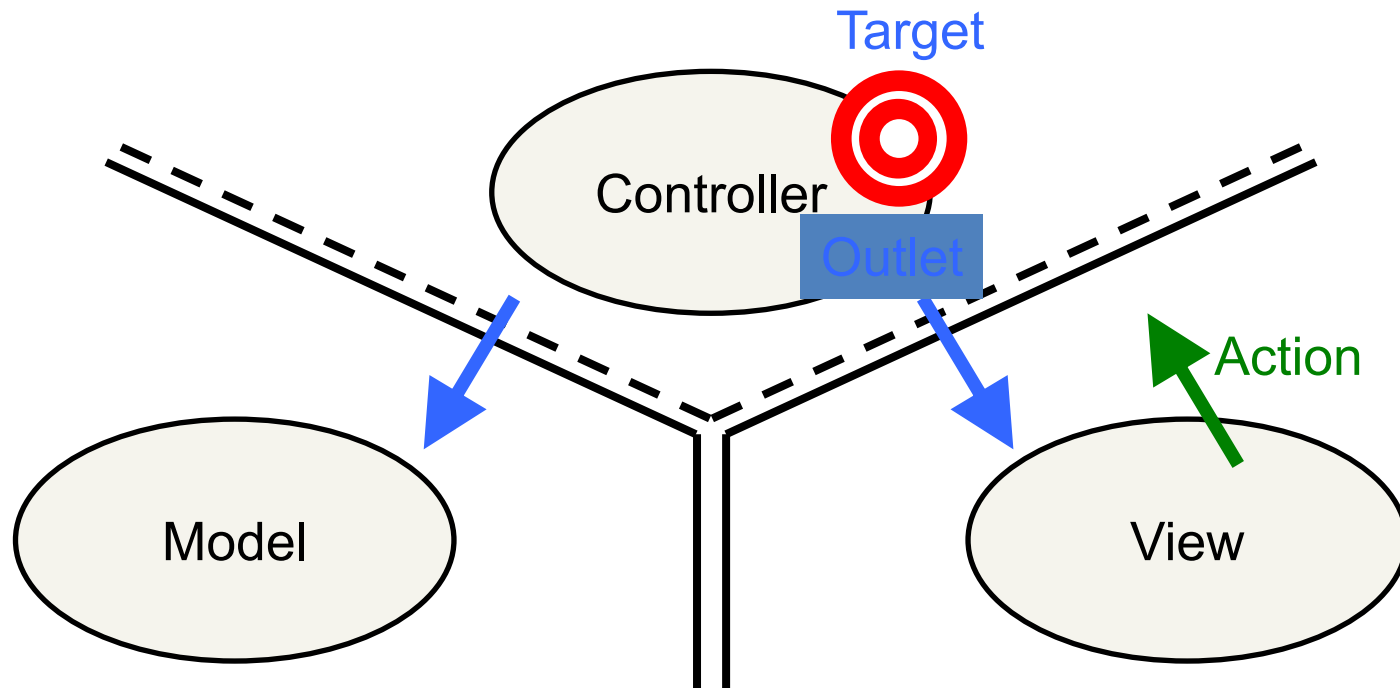
Controller kann direkt Nachrichten an seine View senden



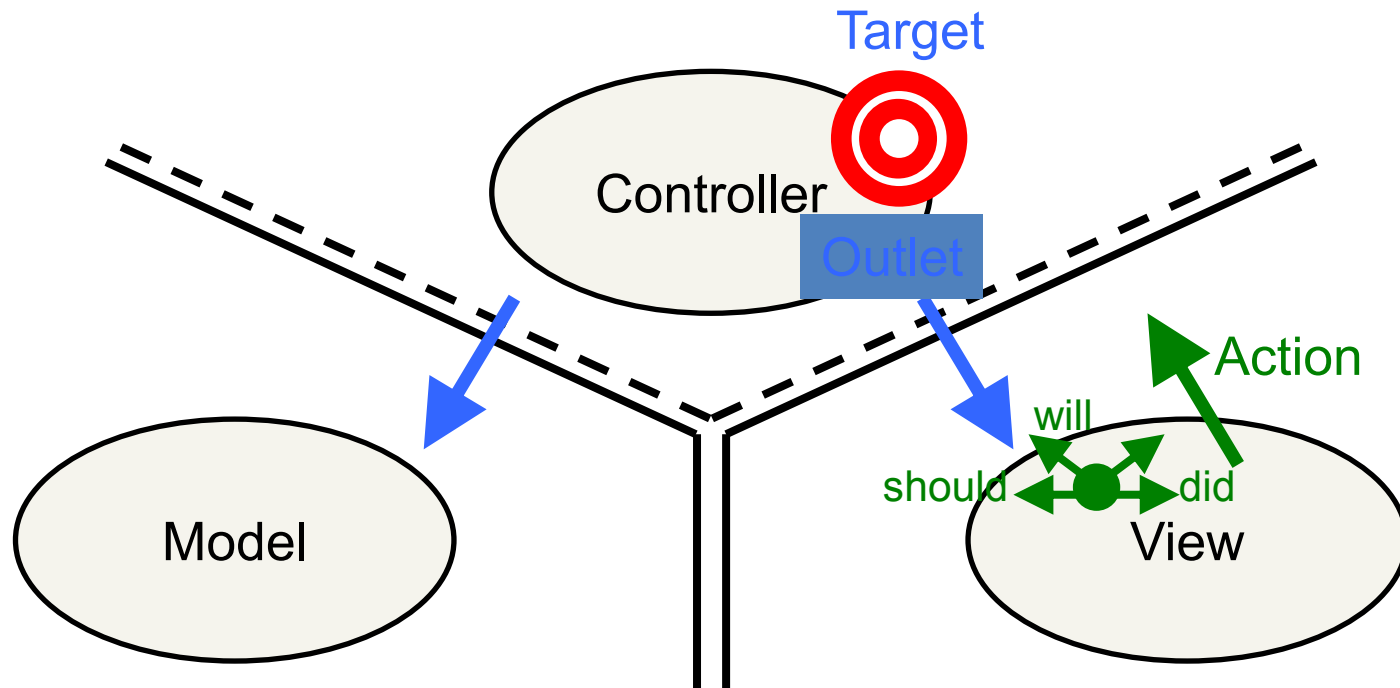
Model und View sollten niemals miteinander kommunizieren!



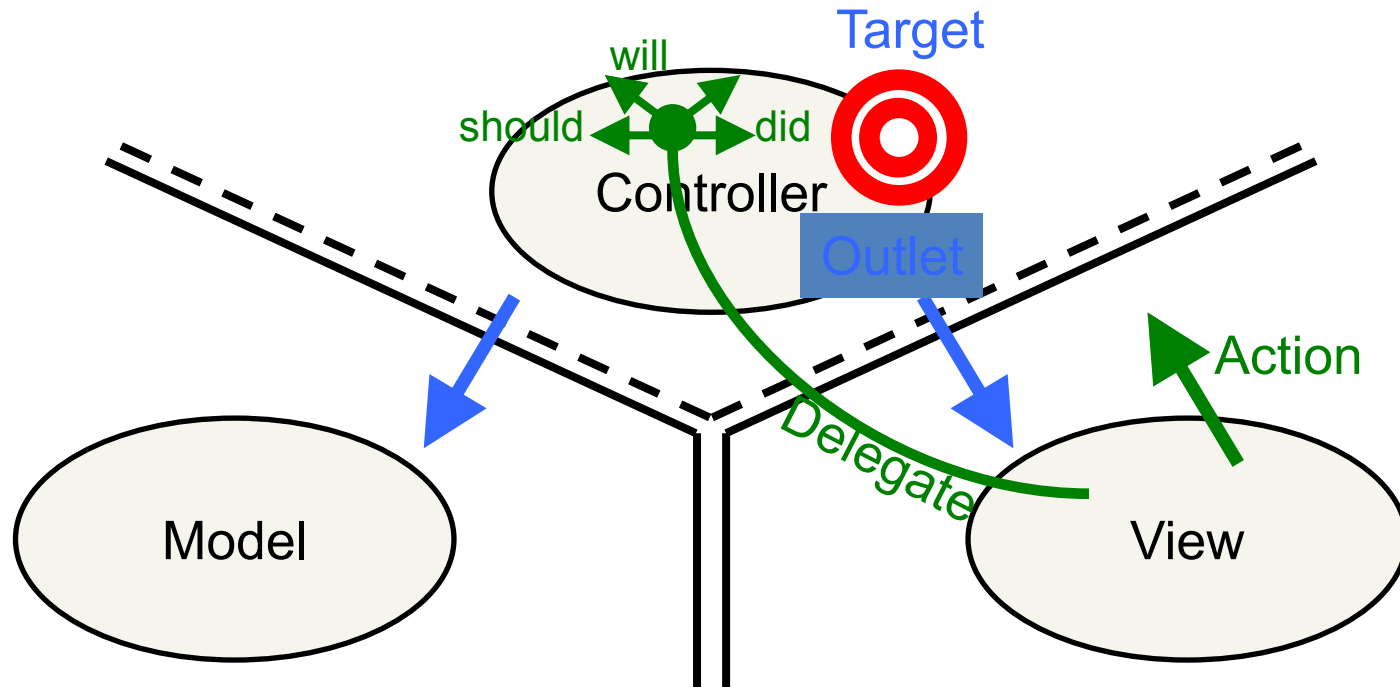
- Wie werden Interaktionen des Benutzers dem Controller kommuniziert?
- Kann die View Nachrichten an den Controller senden?



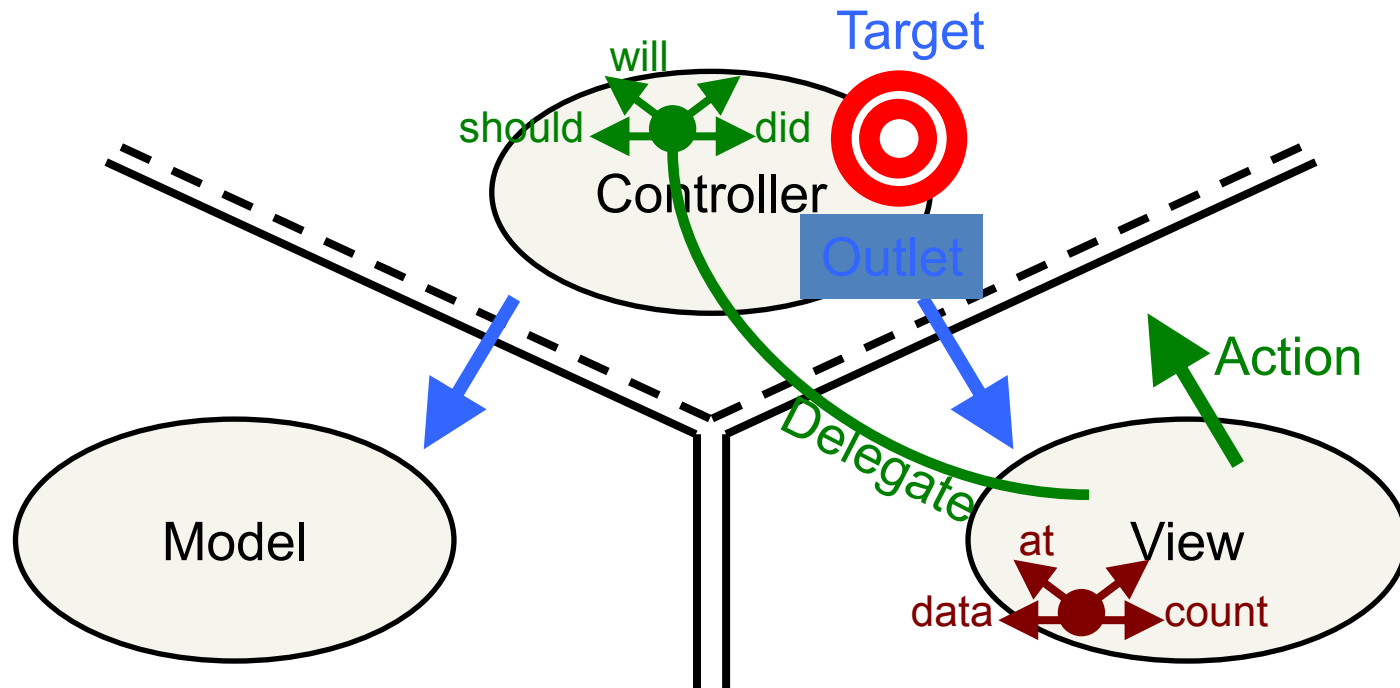
- Controller kann der View sich selbst als Zielobjekt (Target) bestimmter Interaktionen mitteilen
- Kommunikation von Ereignissen erfolgt als Aktion (Action)



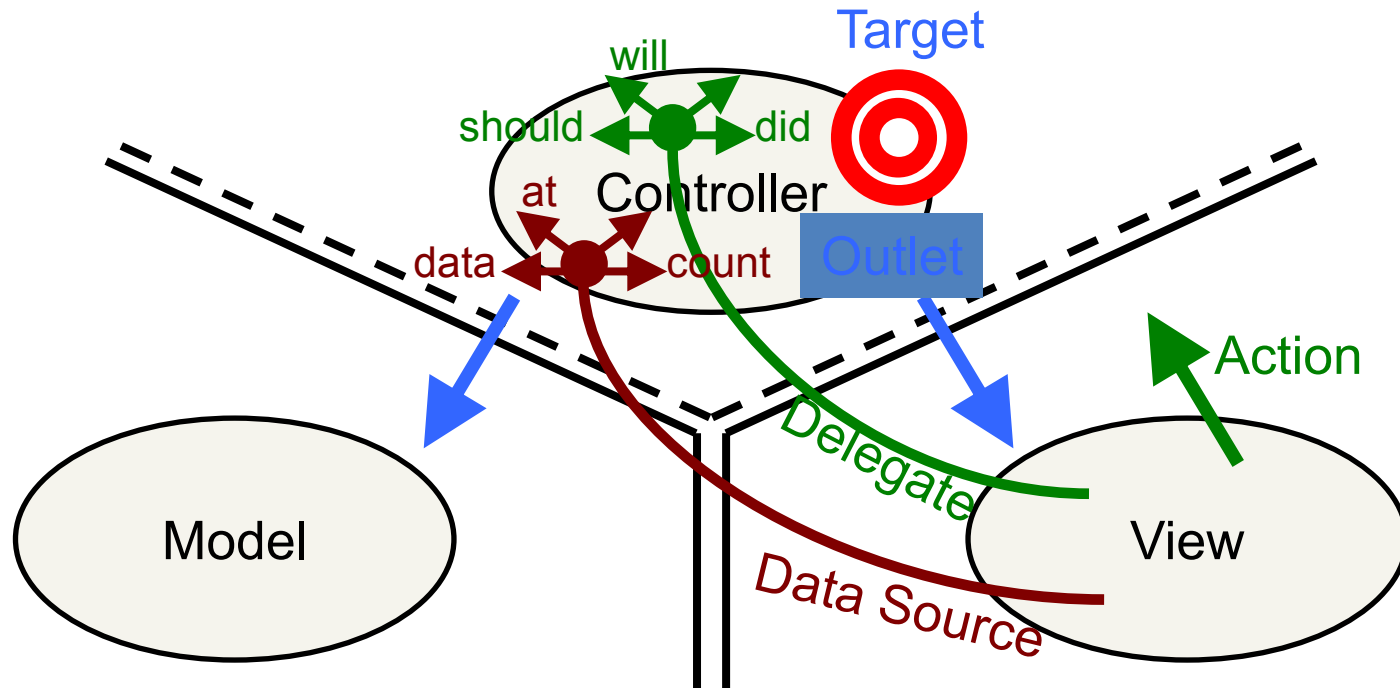
- Manchmal müssen sich eine View und ihr Controller unabhängig von Nutzerinteraktionen synchronisieren



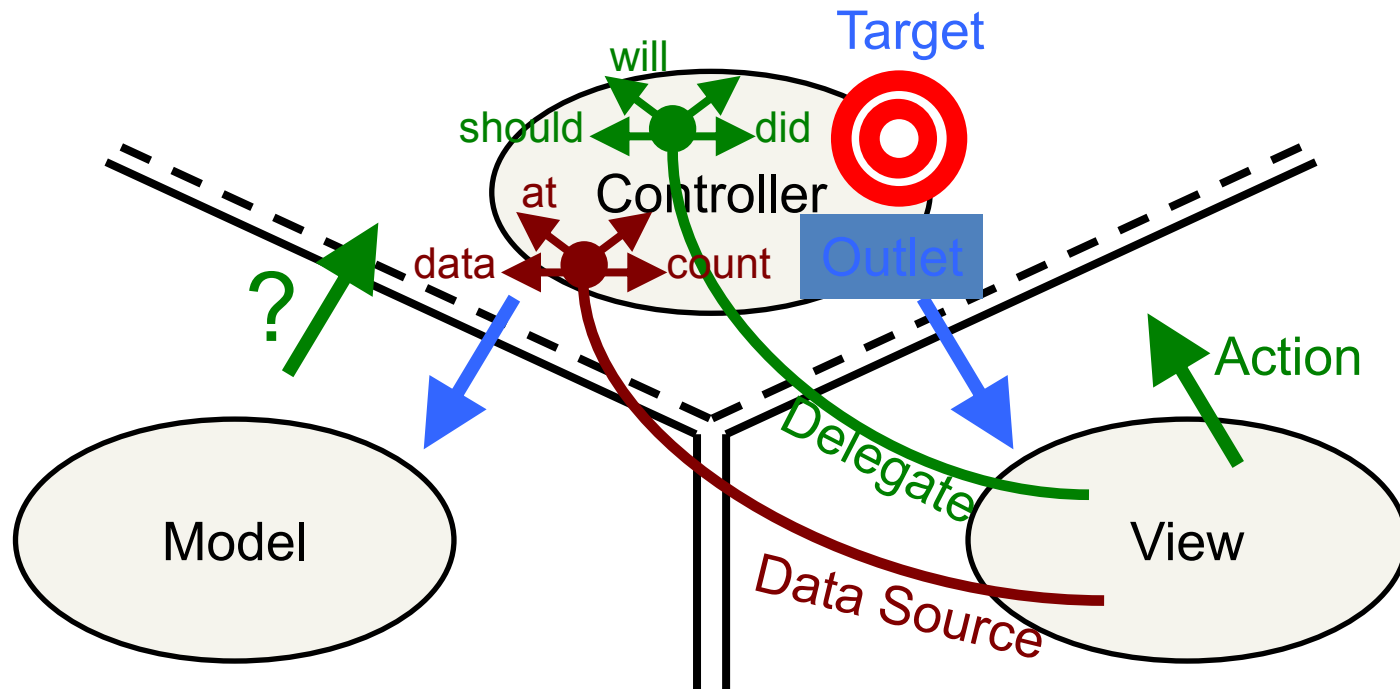
- Synchronisation erfolgt über Delegation (Delegate)
- Delegates werden über Protocols realisiert (ähnlich der Realisierung eines Interface in Java)
- Beispiel:
 - `UITableViewDelegate` Protocol, `tableView:didSelectRowAtIndexPath:`)



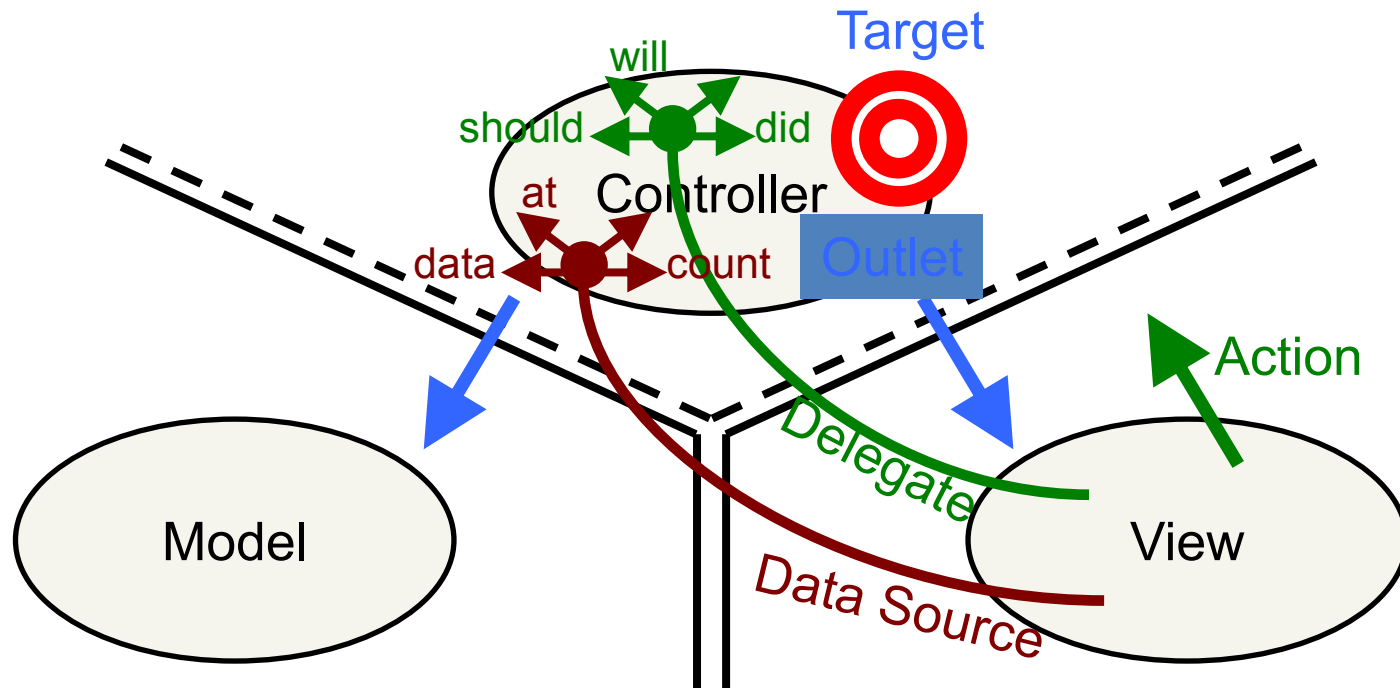
- Views besitzen nicht die Daten (Model), die sie darstellen
- Views verwenden ebenfalls ein Protocol, wenn sie Daten benötigen



- Fast immer stellt der Controller die Datenquelle dar (nicht das Model!)
- Nur der Controller interpretiert und formatiert Daten des Modells für die View!
- Beispiel:
 - `UITableViewDataSource`, `tableView:cellForRowAtIndexPath:`

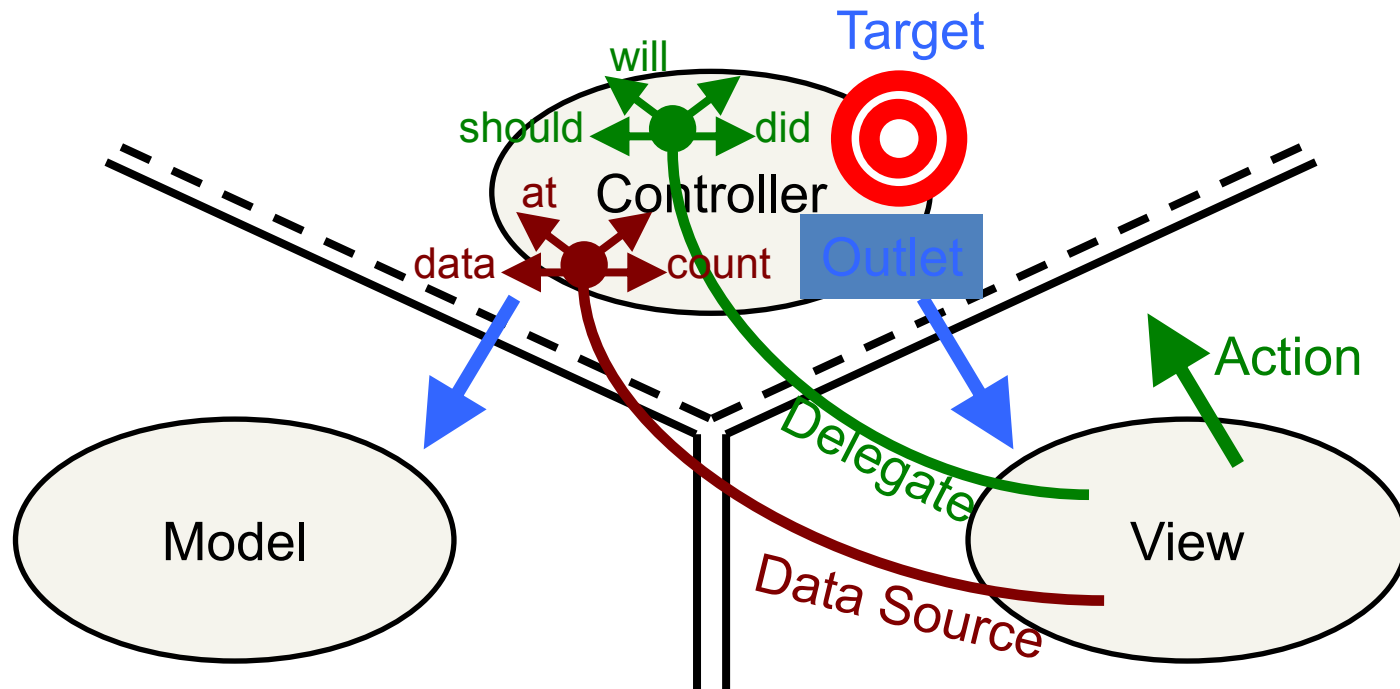


Frage 1: Kann das Model dem Controller Nachrichten senden?

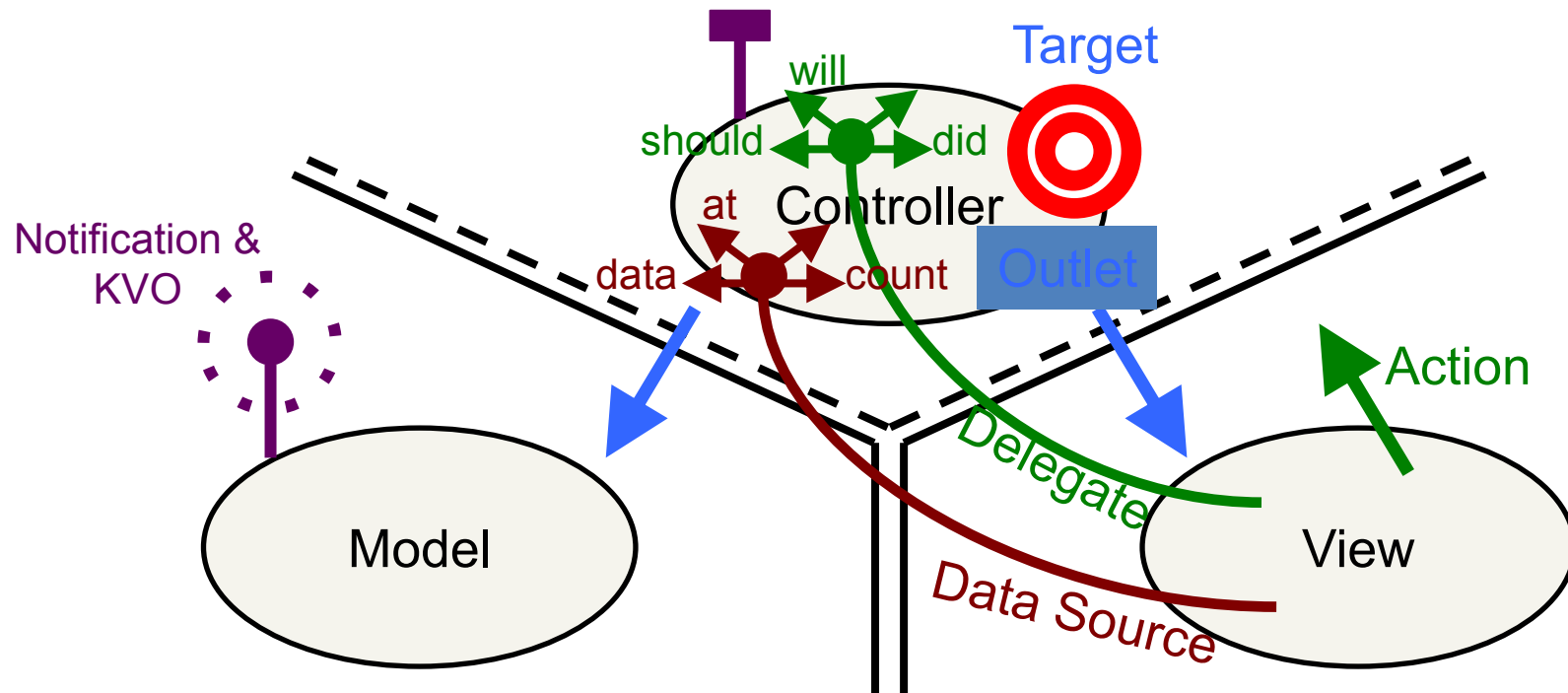


Frage 1: Kann das Model dem Controller Nachrichten senden?

- Nein! Model muss unabhängig von (der Logik) der Darstellung sein

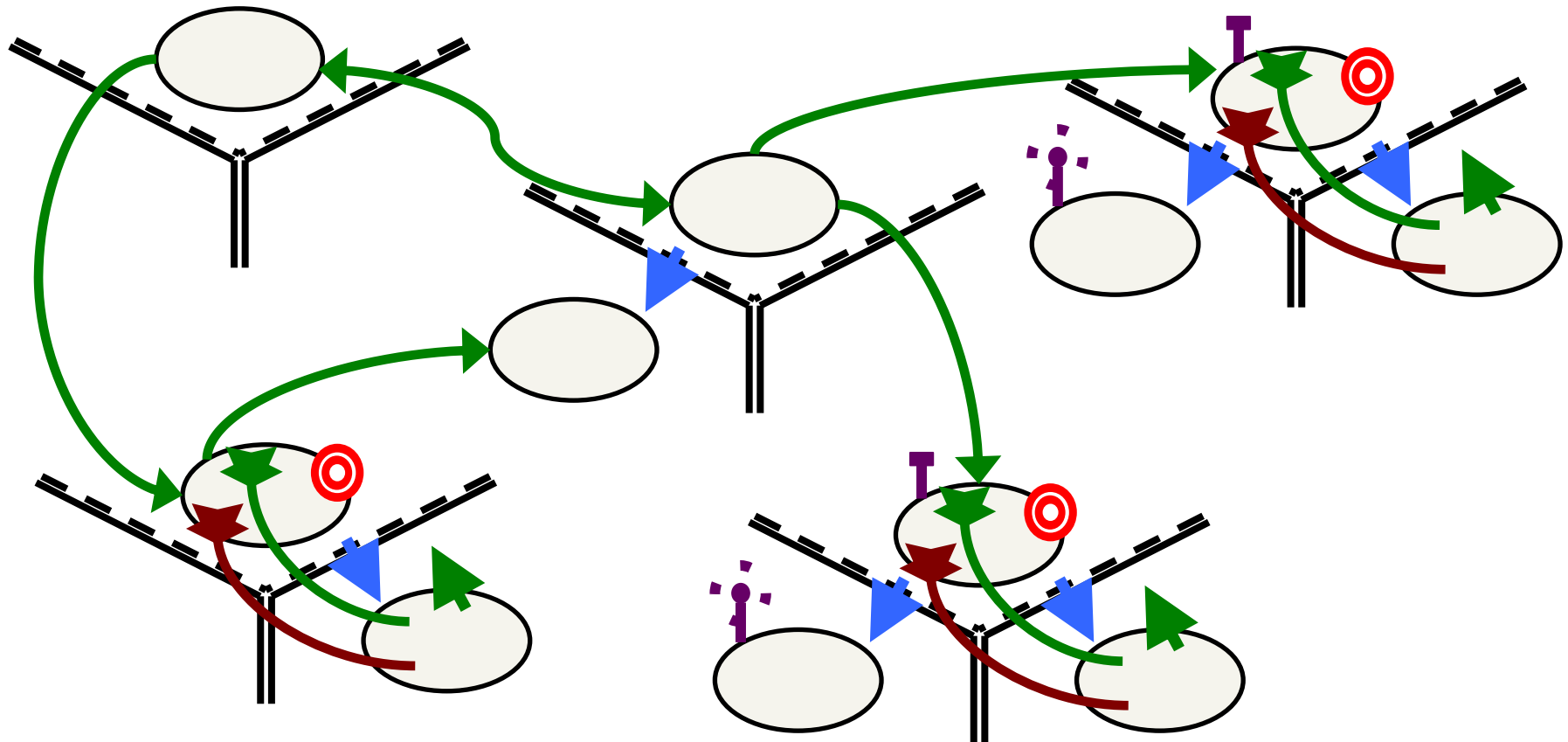


Frage 2: Wie werden Modifikationen an den Daten des Model dem Controller mitgeteilt bzw. in der View aktualisiert?



Frage 2: Wie werden Modifikationen an den Daten des Model dem Controller mitgeteilt bzw. in der View aktualisiert?

- Verwendung eines Broadcast Mechanismus (Notifications und Key-Value-Observing (KVO))
- Controller "lauschen" nach interessanten Nachrichten bzw. Veränderungen

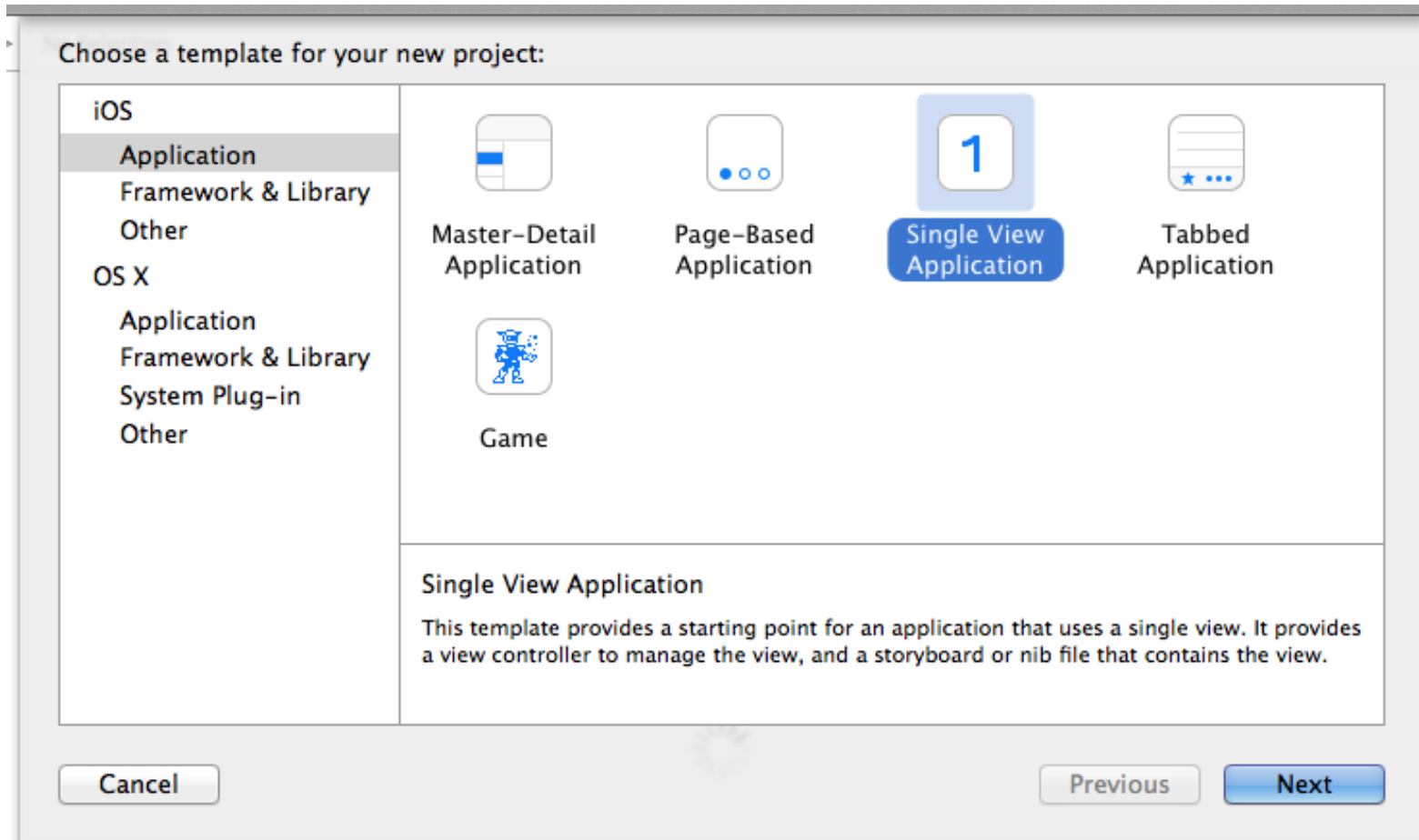


Komplexe Programme entstehen durch die Kombination mehrerer MVC-Gruppen

ERSTE APP IN OBJECTIVE-C (DEMO)



Create New Xcode Project



- File → New → Project → Single View Application
- Erzeugt neues Projekt mit genau einer View

Choose options for your new project:

Product Name:

Organization Name:

Organization Identifier:

Bundle Identifier: Your organization's bundle identifier

Language:

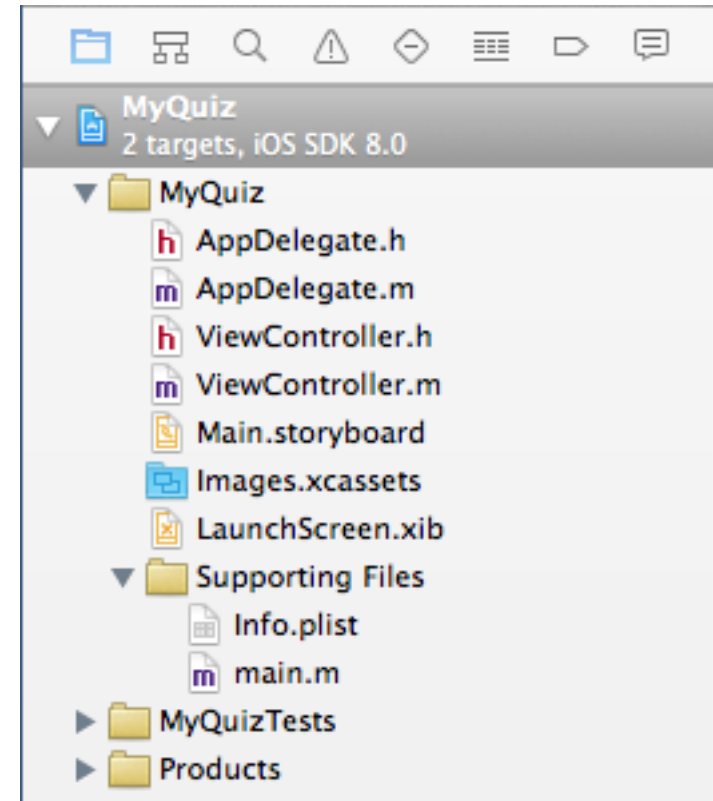
Devices:

☐ Use Core Data

Setzen von Product Name und Organization Identifier

Project Navigator

- Zeigt alle Dateien, aus denen sich ein Projekt zusammensetzt
- Dateien können in Ordnern organisiert werden
- Die Ordnerstruktur ist unabhängig von der Struktur auf dem Dateisystem!
- Für das Template "Single View Application" wird automatisch eine View (in Main.storyboard) und ein Default Controller erstellt!



```
// ViewController.h (Automatisch generierter Code)
```

```
#import <UIKit/UIKit.h>
```

```
@interface ViewController : UIViewController  
@end
```

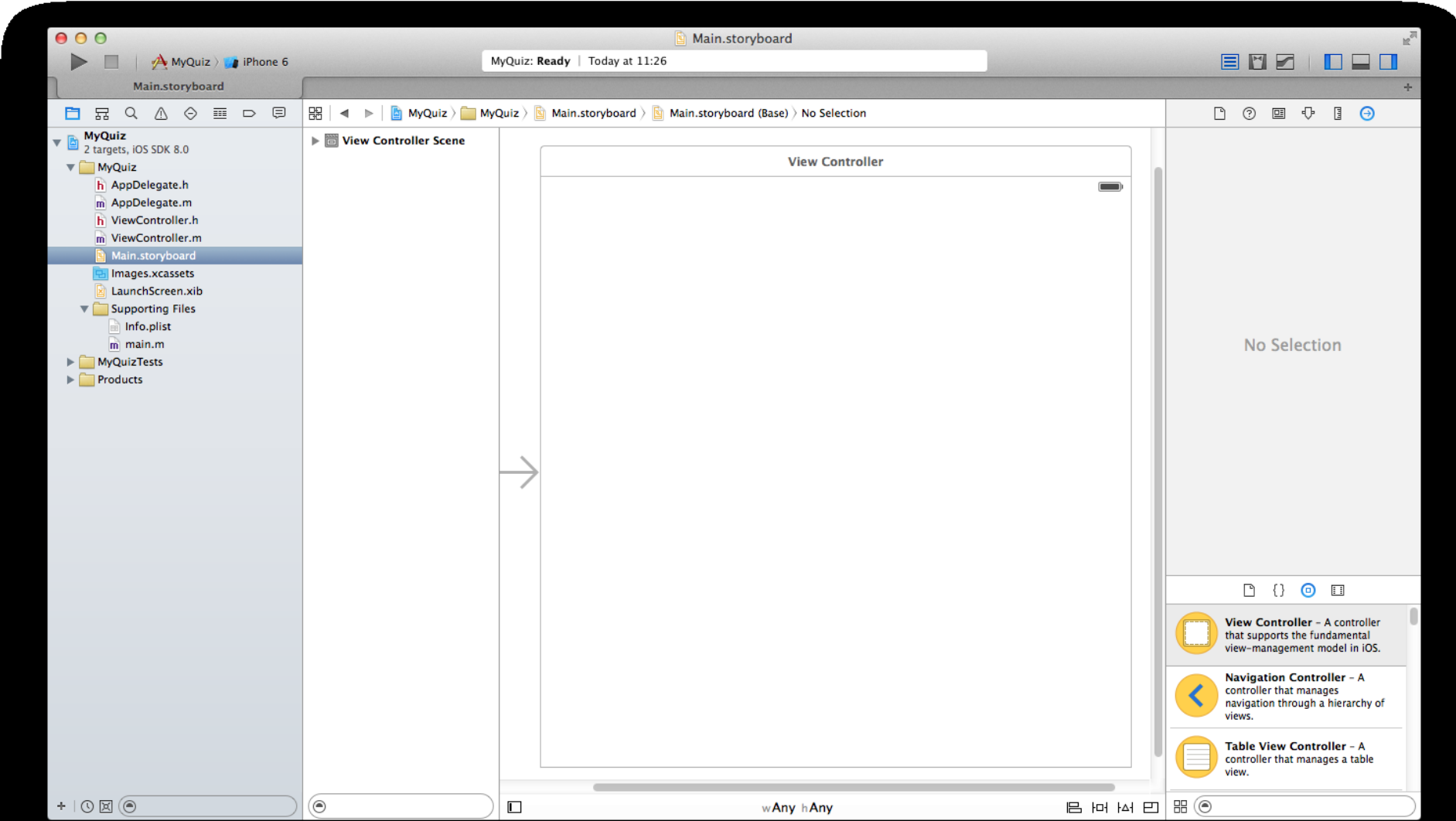
```
// ViewController.m (Automatisch generierter Code)
```

```
#import "ViewController.h"
```

```
@interface ViewController ()  
@end
```

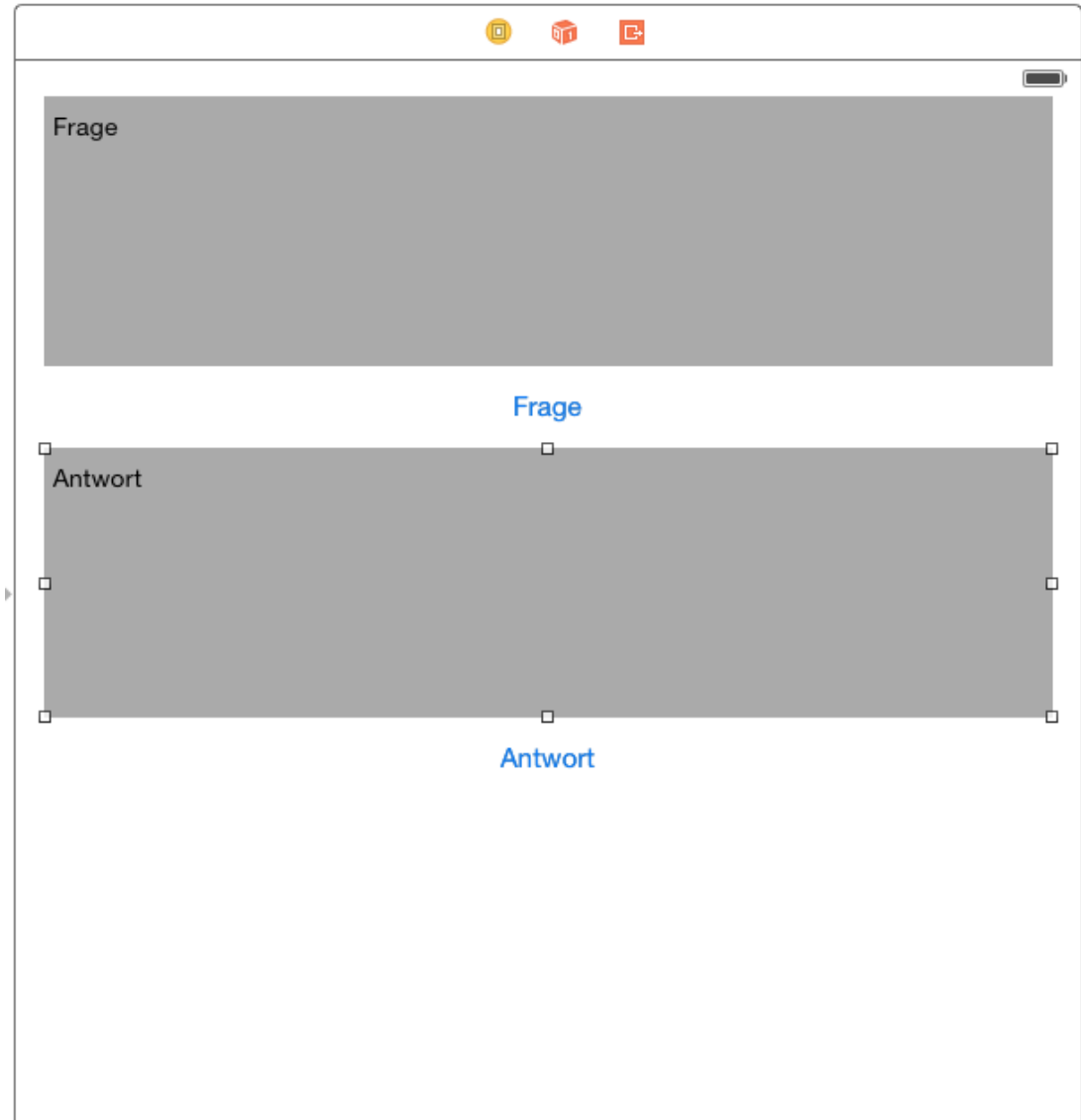
```
@implementation ViewController  
- (void)viewDidLoad {  
    [super viewDidLoad];  
}  
- (void)didReceiveMemoryWarning {  
    [super didReceiveMemoryWarning];  
}  
@end
```

Die View wird innerhalb eines Storyboards ebenfalls automatisch angelegt

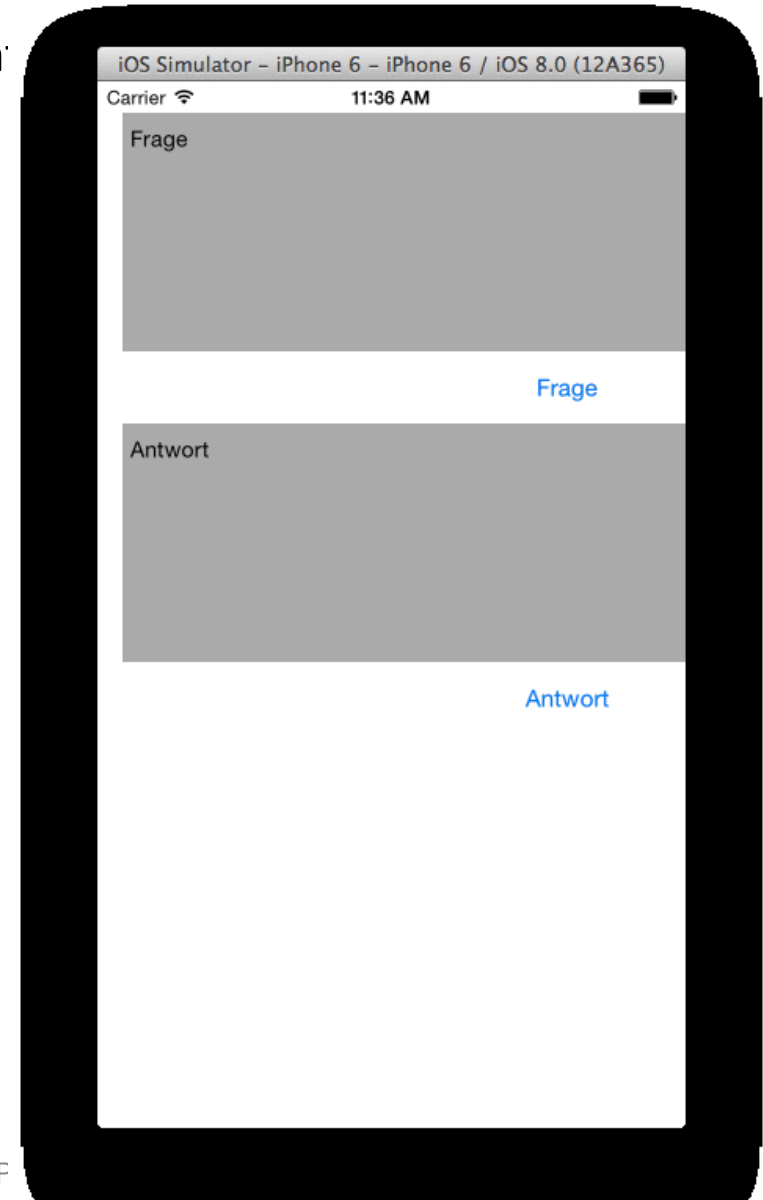


Editieren der View

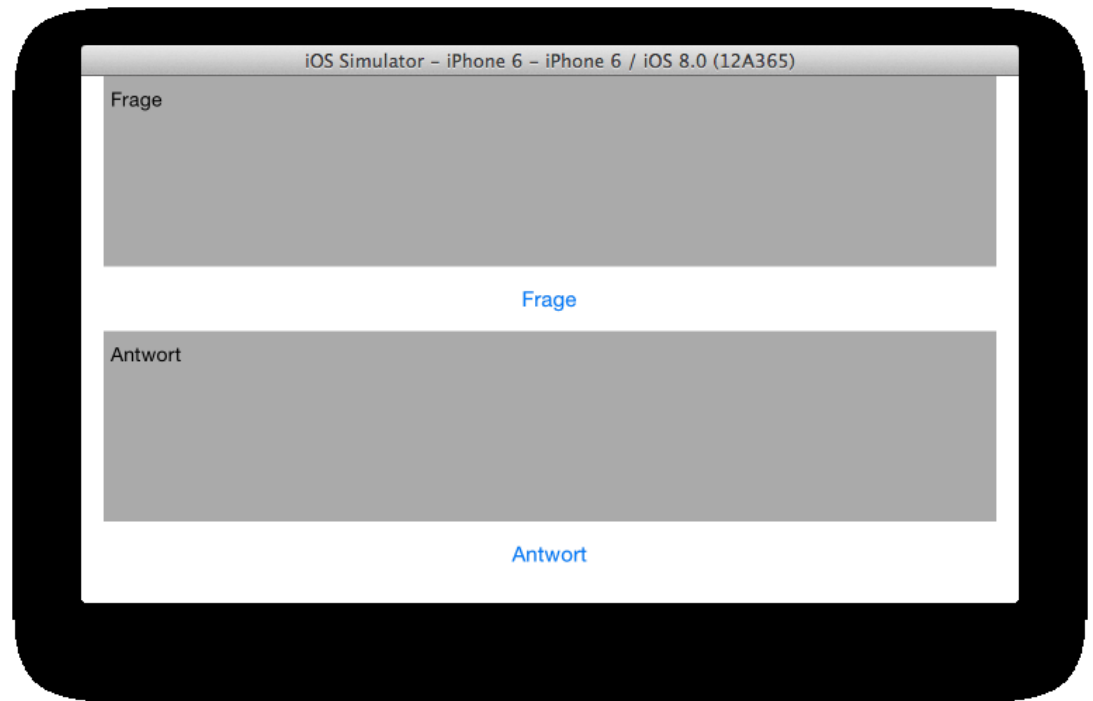
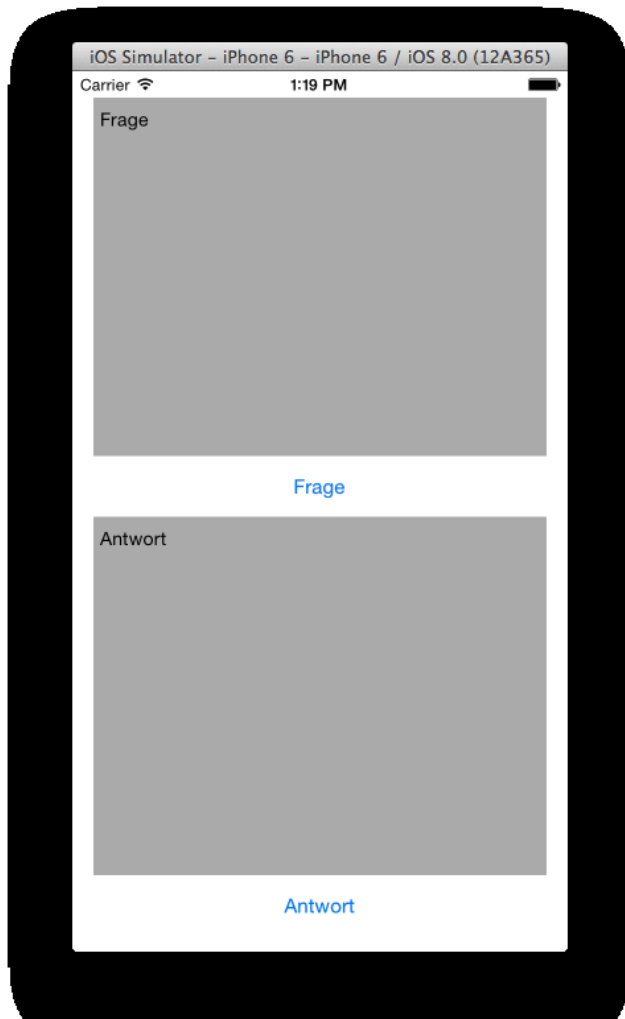
- Mit Hilfe der Object-Library lassen sich sehr einfach die grafischen Elemente platzieren



- Ausführen bringt bisher nicht das gewünschte Ergebnis...
- Problem: Im Storyboard werden durch das Platzieren der UI-Elemente nicht-adaptive Layout-Constraints vergeben



- Lösung: Auflösen der Konflikte über explizite Vergabe adaptiver Layout Constraints (siehe Hausaufgabe)



Hinzufügen eines Model

- File → New → File → OS X (Source) → Cocoa Class
- QuestionPool: dient der Erzeugung und dem Zugriff auf Fragen und Antworten

1. Weg: Implementierung unter direkter Verwendung von Instanzvariablen

```
#import <Foundation/Foundation.h>

@interface QuestionPool : NSObject {
    NSArray* _questions;
    NSArray* _answers;
}

// Getter-Methoden für den Zugriff auf Instanzvariablen
-(NSArray*)questions;
-(NSArray*)answers;

@end
```

1. Weg: Implementierung unter direkter Verwendung von Instanzvariablen

```
#import "QuestionPool.h"

@implementation QuestionPool

- (instancetype)init {
    self = [super init];
    if (self) {
        _questions = @[@"Wie heißt die Landeshauptstadt von Bayern?",
                       @"Wie viele Einwohner hat München?",
                       @"Wie hoch ist die Frauenkirche?"];
        _answers = @[@"München",
                     @"1,4 Millionen",
                     @"98,67 Meter"];
    }
    return self;
}

- (NSArray *)answers {
    return _answers;
}

- (NSArray *)questions {
    return _questions;
}

@end
```

2. Weg: Implementierung mit Hilfe von Properties

```
// QuestionPool.h

#import <Foundation/Foundation.h>

@interface QuestionPool : NSObject

@property (readonly, nonatomic, strong) NSArray *questions;
@property (readonly, nonatomic, strong) NSArray *answers;

@end
```

2. Weg: Implementierung mit Hilfe von Properties

- Lazy Instantiation: Alles so spät wie möglich...

```
@implementation QuestionPool
```

```
// Achtung: @synthesize hier notwendig, da in der
// getter-Methode der readonly-Properties die
// automatisch erzeugten Instanzvariablen verändert werden!
@synthesize questions = _questions;
@synthesize answers = _answers;
-(NSArray*)questions {
    if(!_questions) {
        _questions = @[@"Wie heißt die Landeshauptstadt von Bayern?",
                        @"Wie viele Einwohner hat München?",
                        @"Wie hoch ist die Frauenkirche?"];
    }
    return _questions;
}
-(NSArray*)answers {
    if(!_answers) {
        _answers = @[@"München",
                     @"1,4 Millionen",
                     @"98,67 Meter"];
    }
    return _answers;
}
@end
```

Verbinden von Model und Controller

```
// ViewController.m

#import "ViewController.h"
#import "QuestionPool.h"

@interface ViewController ()

@property (strong, nonatomic) QuestionPool *questionPool;
@property (nonatomic) NSUInteger currentQuestionIndex;

@end

@implementation ViewController

[...]
```

@end

Instanziieren des Model

- Lazy Instantiation: Alles so spät wie möglich... (mehr zu [viewWillAppear:](#) später)

```
// ViewController.m

#import "ViewController.h"
#import "QuestionPool.h"

@interface ViewController ()
@property (strong, nonatomic) QuestionPool *questionPool;
@property (nonatomic) NSUInteger currentQuestionIndex;
@end

@implementation ViewController

-(void)viewWillAppear:(BOOL)animated {
    if(!self.questionPool) {
        self.questionPool = [QuestionPool new];
    }
}

@end
```

Properties für UI Elemente

- Erzeugen der Properties für `UIButton`- und `UITextView`-Instanzen (`IBOutlet` ist nur ein `typedef` auf `void` und hilft dem Compiler beim Erzeugen der Links zwischen Header- und Storyboard- / XIB-Files)

```
// ViewController.m

#import "ViewController.h"
#import "QuestionPool.h"

@interface ViewController ()

@property (weak, nonatomic) IBOutlet UITextView *questionTextView;
@property (weak, nonatomic) IBOutlet UITextView *answerTextView;
@property (weak, nonatomic) IBOutlet UIButton *questionButton;
@property (weak, nonatomic) IBOutlet UIButton *answerButton;

@property QuestionPool *questionPool;
@property (nonatomic) NSUInteger currentQuestionIndex;
@end
[...]
```

Deklaration der Methoden (Actions)

```
// ViewController.m

#import "ViewController.h"
#import "QuestionPool.h"

@interface ViewController ()
[...]
@end

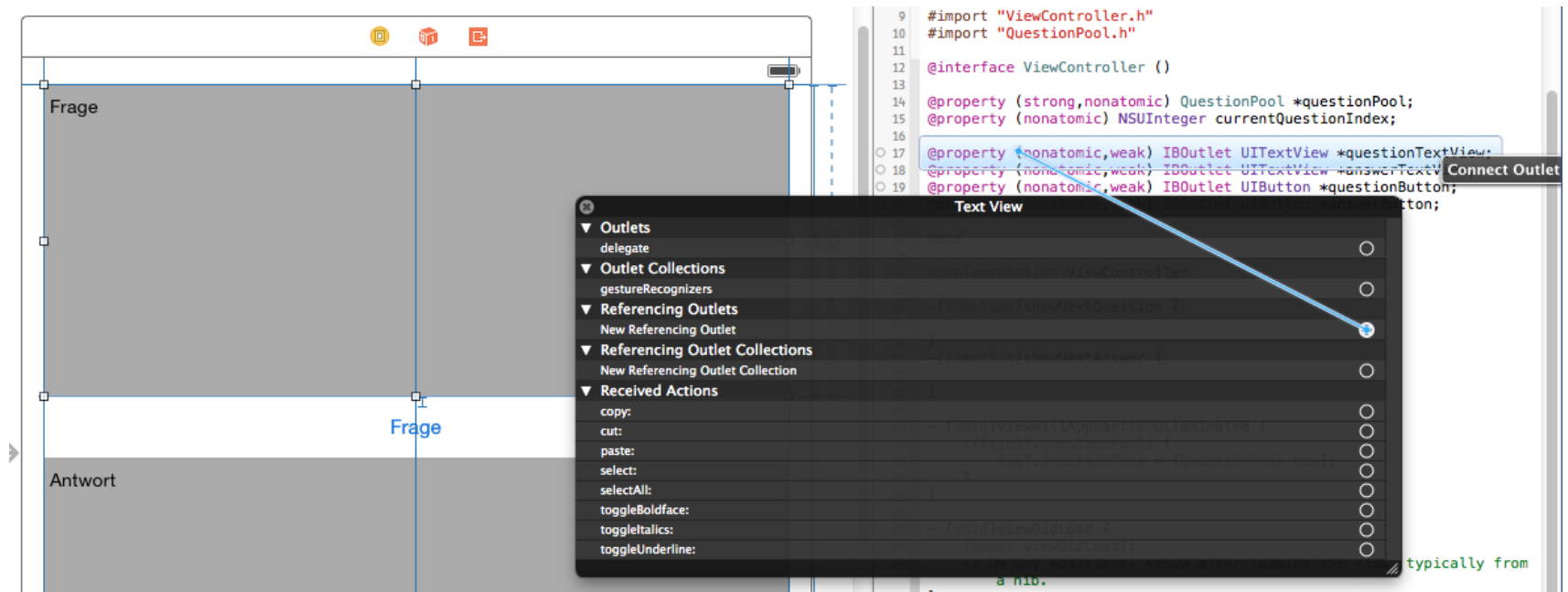
@implementation ViewController

-(IBAction)showNextQuestion {}
-(IBAction)showAnswer {}

@end
```

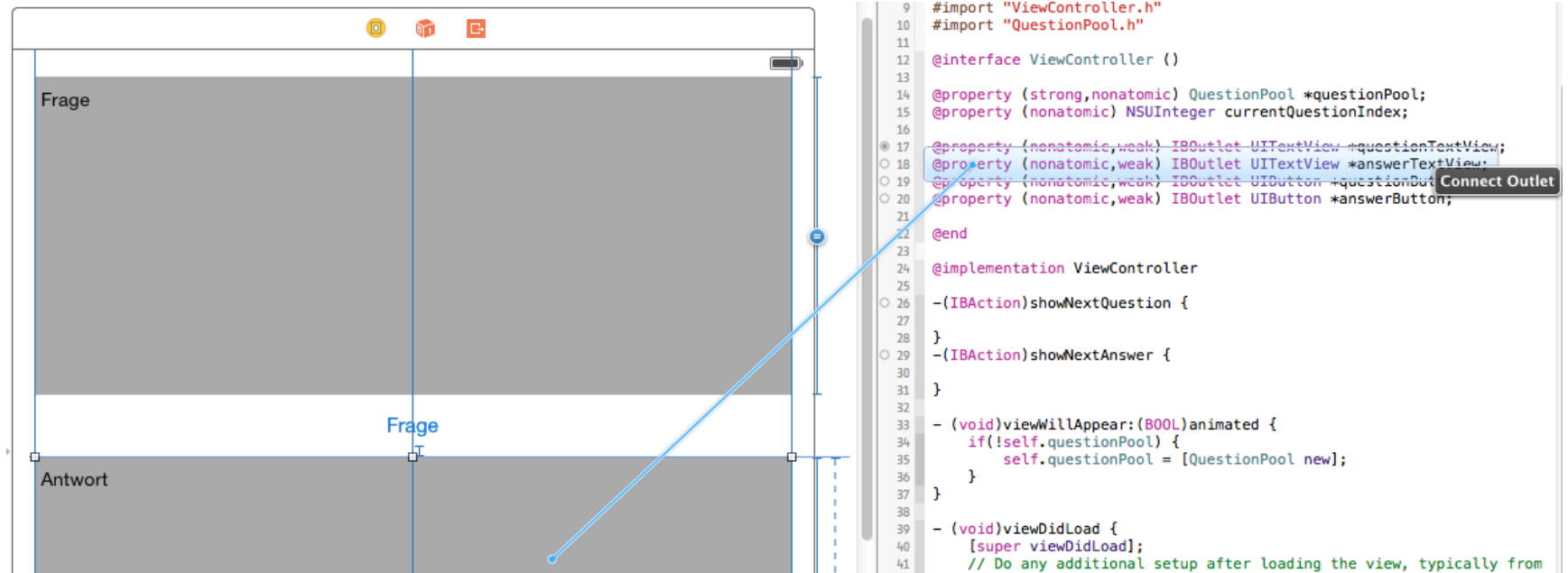
Verbinden von Controller und View (Outlets und Actions)

- Mehrere Methoden möglich



Verbinden von Controller und View (Outlets und Actions)

- Mehrere Methoden möglich

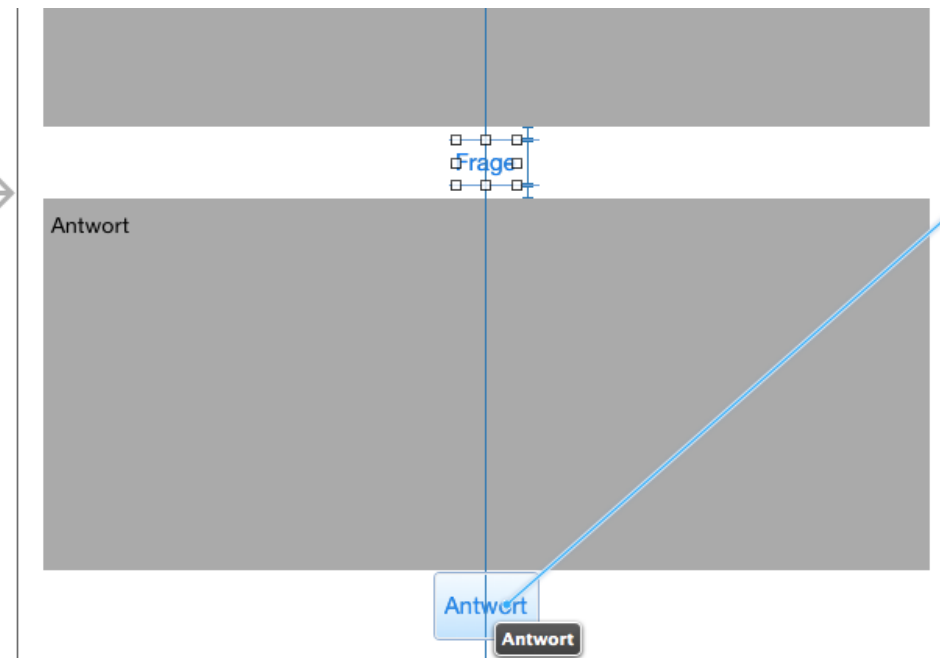


```

9  #import "ViewController.h"
10 #import "QuestionPool.h"
11
12 @interface ViewController ()
13
14 @property (strong, nonatomic) QuestionPool *questionPool;
15 @property (nonatomic) NSUInteger currentQuestionIndex;
16
17 @property (nonatomic, weak) IBOutlet UITextView *questionTextView;
18 @property (nonatomic, weak) IBOutlet UITextView *answerTextView;
19 @property (nonatomic, weak) IBOutlet UIButton *questionButton;
20 @property (nonatomic, weak) IBOutlet UIButton *answerButton;
21
22 @end
23
24 @implementation ViewController
25
26 -(IBAction)showNextQuestion {
27
28 }
29
30 -(IBAction)showNextAnswer {
31
32 }
33
34 -(void)viewWillAppear:(BOOL)animated {
35     if(!self.questionPool) {
36         self.questionPool = [QuestionPool new];
37     }
38 }
39
40 -(void)viewDidLoad {
41     [super viewDidLoad];
42     // Do any additional setup after loading the view, typically from
  
```

Verbinden von Controller und View (Outlets und Actions)

- Mehrere Methoden möglich

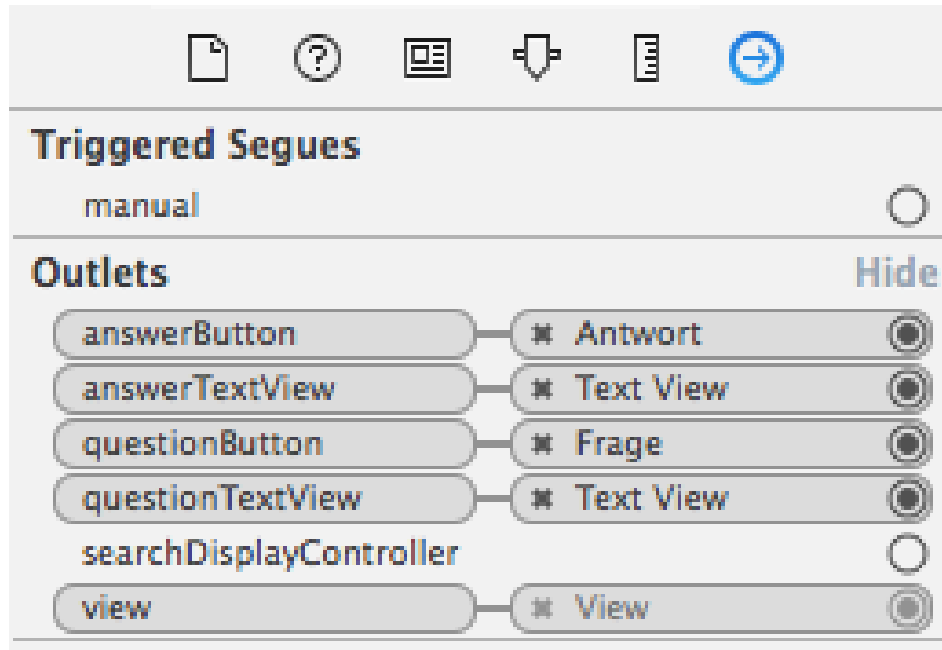


```

14 @property (strong, nonatomic) QuestionPool *questionPool;
15 @property (nonatomic) NSUInteger currentQuestionIndex;
16
17 @property (nonatomic, weak) IBOutlet UITextView *questionTextView;
18 @property (nonatomic, weak) IBOutlet UITextView *answerTextView;
19 @property (nonatomic, weak) IBOutlet UIButton *questionButton;
20 @property (nonatomic, weak) IBOutlet UIButton *answerButton;
21
22 @end
23
24 @implementation ViewController
25
26 -(IBAction)showNextQuestion {
27
28 }
29 -(IBAction)showNextAnswer {
30
31 }
32
33 -(void)viewWillAppear:(BOOL)animated {
34     if(!self.questionPool) {
35         self.questionPool = [QuestionPool new];
36     }
37 }
38
39 -(void)viewDidLoad {
40     [super viewDidLoad];
41     // Do any additional setup after loading the view, typically in a nib.
42 }
43
44 -(void)didReceiveMemoryWarning {
45     [super didReceiveMemoryWarning];

```

Inhalt des Connections Inspector nachdem alle Verbindungen existieren:



Implementierung der Methoden (Actions)

```
// ViewController.m

@implementation ViewController

-(IBAction)showNextQuestion {
    self.questionButton.enabled = NO;
    self.answerButton.enabled = YES;
    self.questionTextView.text =
        self.questionPool.questions[self.currentQuestionIndex];
}

-(IBAction)showAnswer {
    self.questionButton.enabled = YES;
    self.answerButton.enabled = NO;
    self.answerTextView.text =
        self.questionPool.answers[self.currentQuestionIndex];
    [self incrementQuestionIndex];
}

-(void)incrementQuestionIndex {
    self.currentQuestionIndex =
        (self.currentQuestionIndex+1) % [self.questionPool.questions count];
}

@end
```

Problem:

- Man kann Antwort klicken, bevor die Frage gestellt wurde

Lösung:

```
// ViewController.m
[...]
```

```
@implementation ViewController

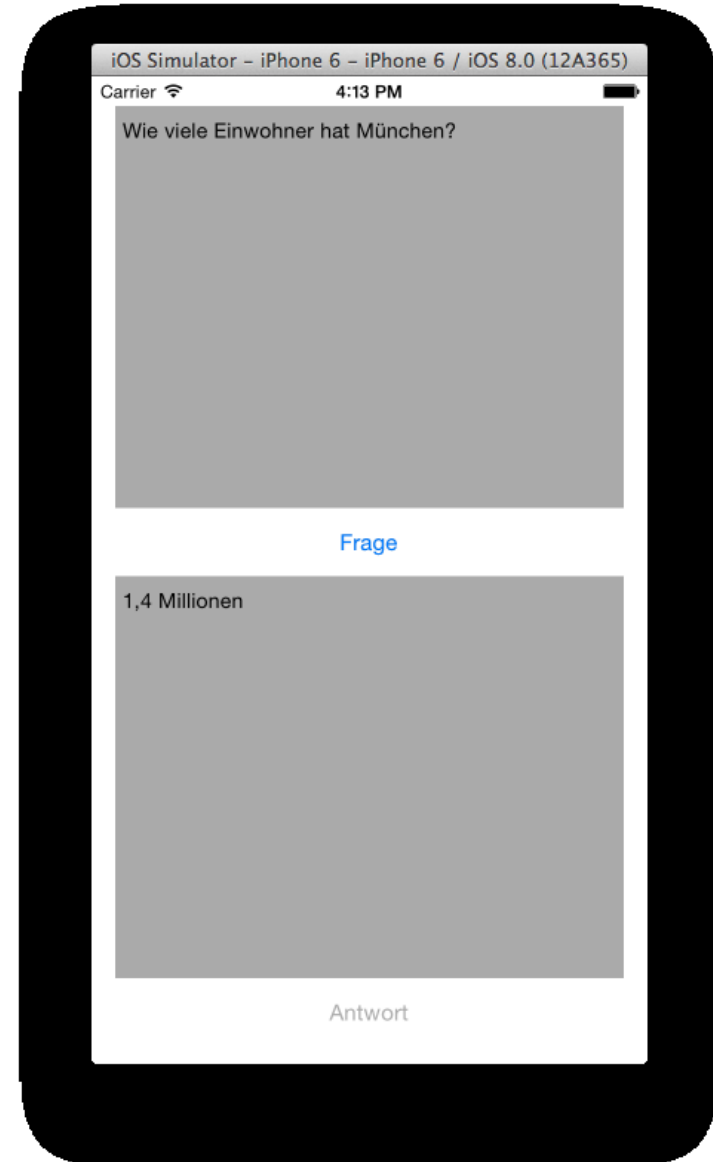
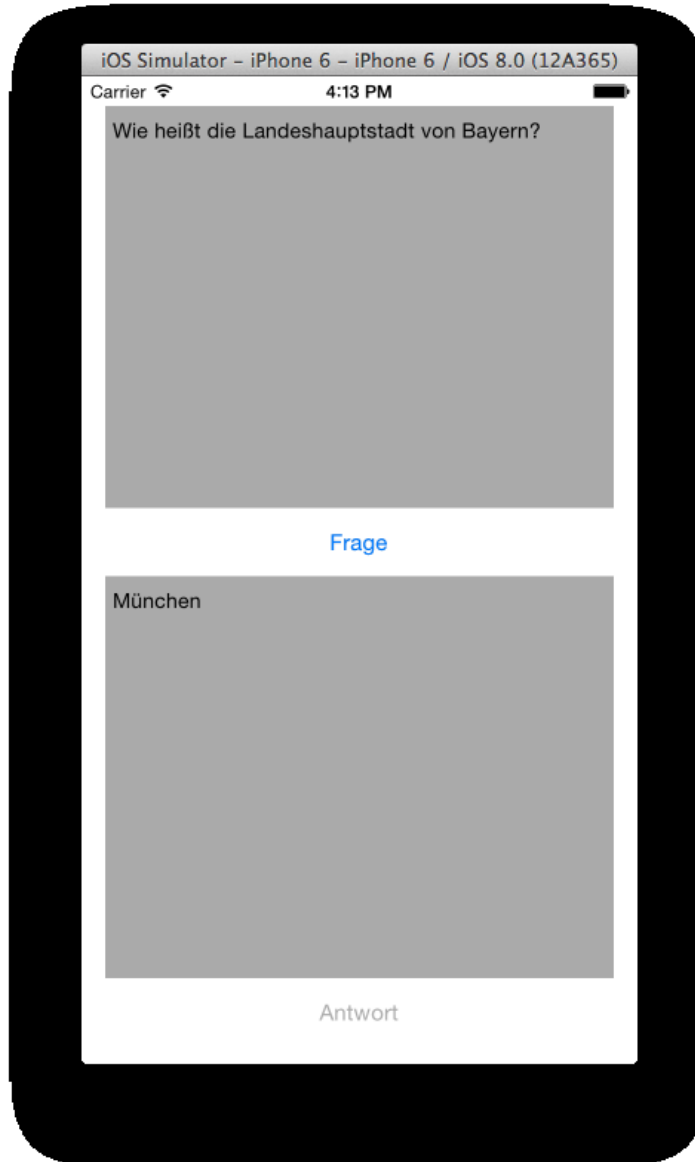
[...]
```

```
-(void)viewDidLoad {
    self.answerButton.enabled = NO;
}
```

```
[...]
```

```
@end
```

Ergebnis:



Storyboards sind schön, aber...

- führen zu Problemen, wenn mehrere Personen parallel an der UI arbeiten
- überflüssig, falls man auch die UI rein programmatisch umsetzen möchte

Alternativ kann man die UI mit Hilfe von XIB-Dateien implementieren

- Pro View eine XIB-Datei

Erzeugen eines neuen Projekts:

- File → New → Project → Single View Application

Choose options for your new project:

Product Name:

Organization Name:

Organization Identifier:

Bundle Identifier:

Language:

Devices:

☐ Use Core Data

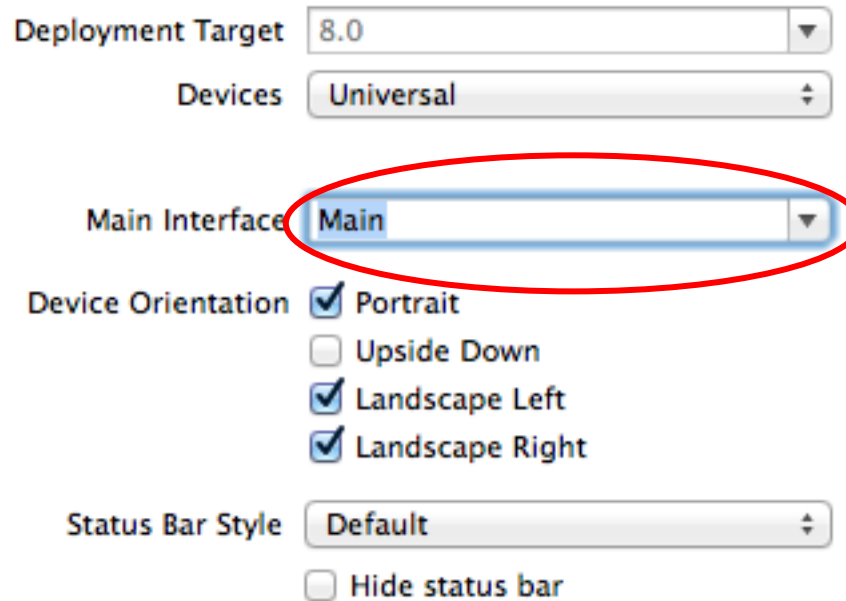
Project Navigator

- Löschen der Datei "Main.storyboard"

Project Navigator → General → Deployment Info

- Löschen des Inhalts von "Main Interface"

▼ Deployment Info



Deployment Target: 8.0

Devices: Universal

Main Interface: Main

Device Orientation:
☒ Portrait
☐ Upside Down
☒ Landscape Left
☒ Landscape Right

Status Bar Style: Default

☐ Hide status bar

Project Navigator

- Anpassen der Datei "AppDelegate"

```
import UIKit

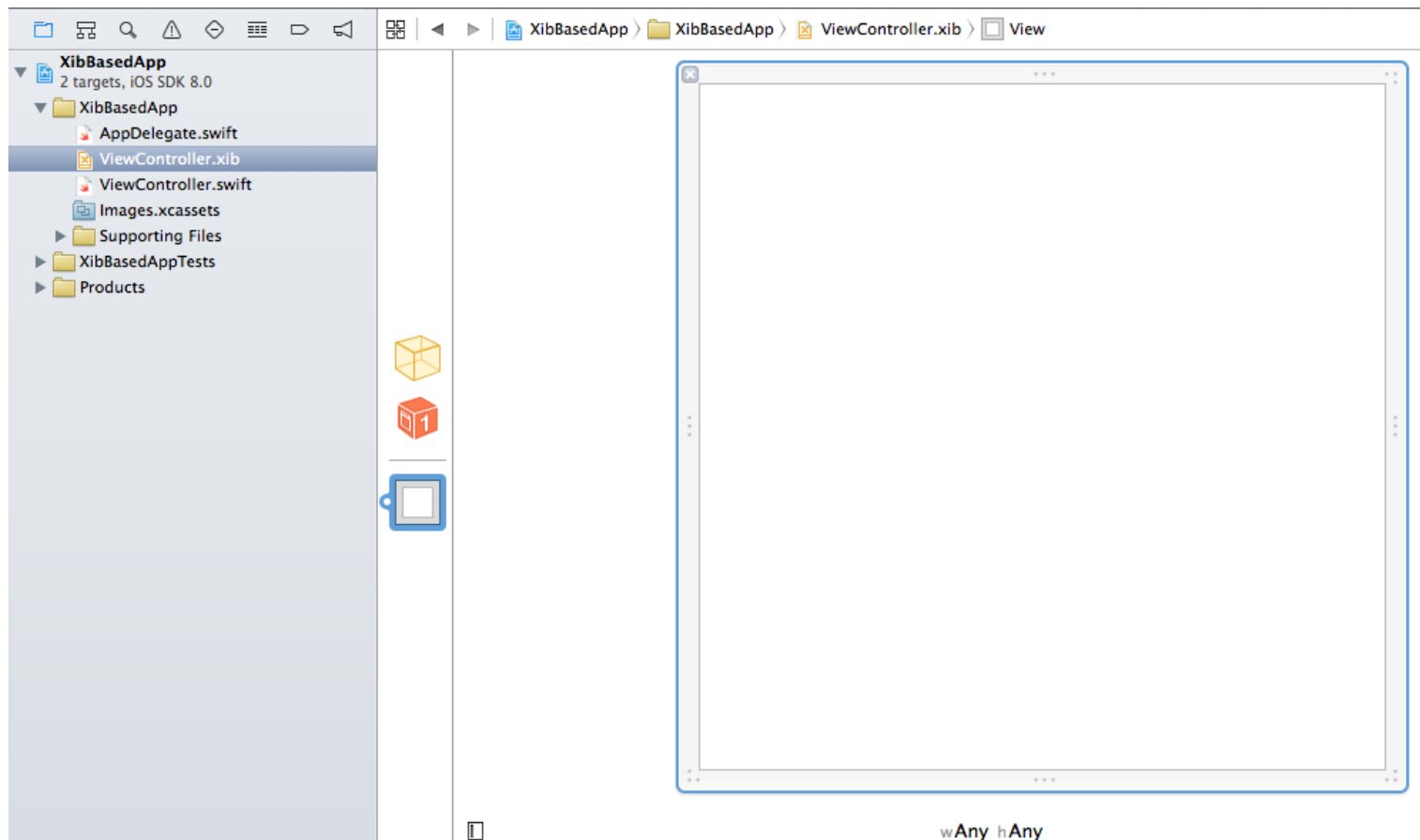
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
    var vc:ViewController?

    func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject: AnyObject]?) -> Bool {
        window = UIWindow(frame: UIScreen.mainScreen().bounds)
        window!.backgroundColor = UIColor.redColor()
        vc = ViewController(nibName: "ViewController", bundle: nil)
        window!.rootViewController = vc
        window!.makeKeyAndVisible()
        return true
    }
    func applicationWillResignActive(application: UIApplication) {}
    func applicationDidEnterBackground(application: UIApplication) {}
    func applicationWillEnterForeground(application: UIApplication) {}
    func applicationDidBecomeActive(application: UIApplication) {}
    func applicationWillTerminate(application: UIApplication) {}
}
```

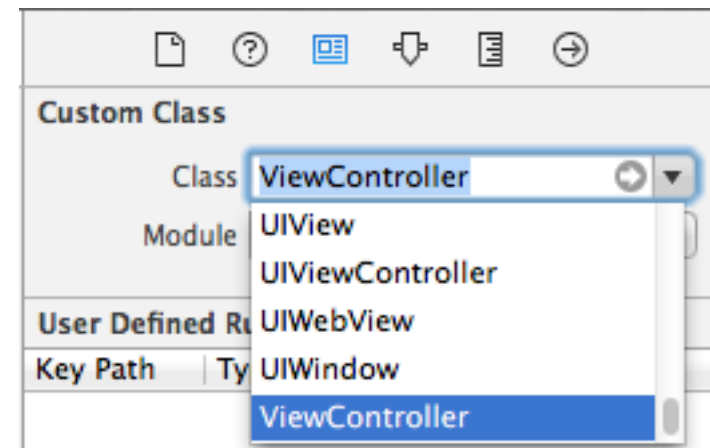
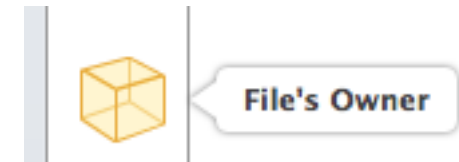
Erzeugen einer XIB-Datei

- File → New → File → User Interface → View
- Name "ViewController"



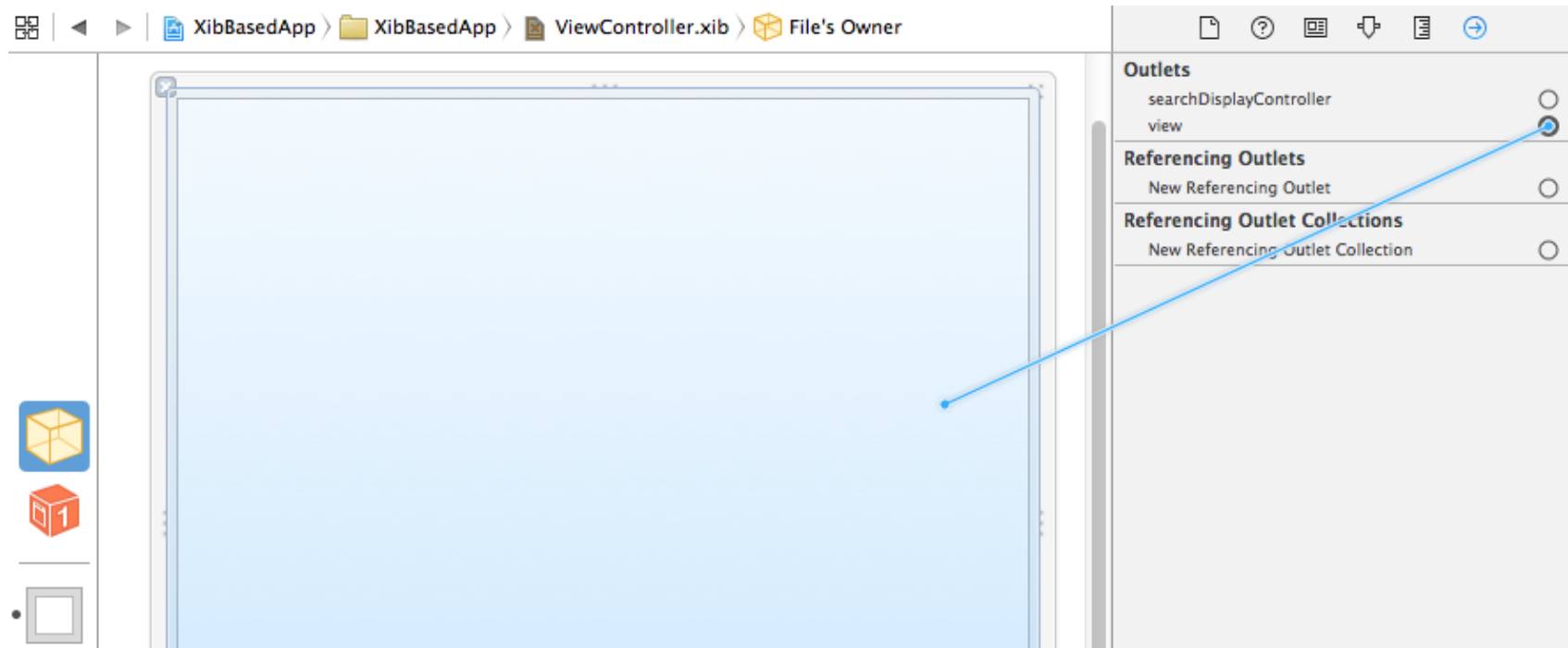
Anpassen der XIB-Datei

- Über das Setzen des "File's Owner" wird eine Verbindung zwischen der Implementierung (ViewController.swift) und der UI (ViewController.xib) hergestellt
- Im Editor
 - Click auf "File's Owner"
- Im "Identity Inspector"
 - Setzen der "Custom Class" auf "ViewController"

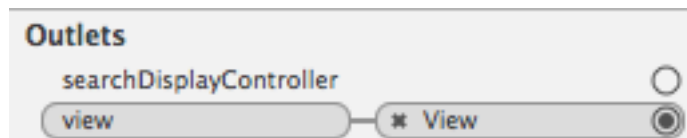


Anpassen der XIB-Datei

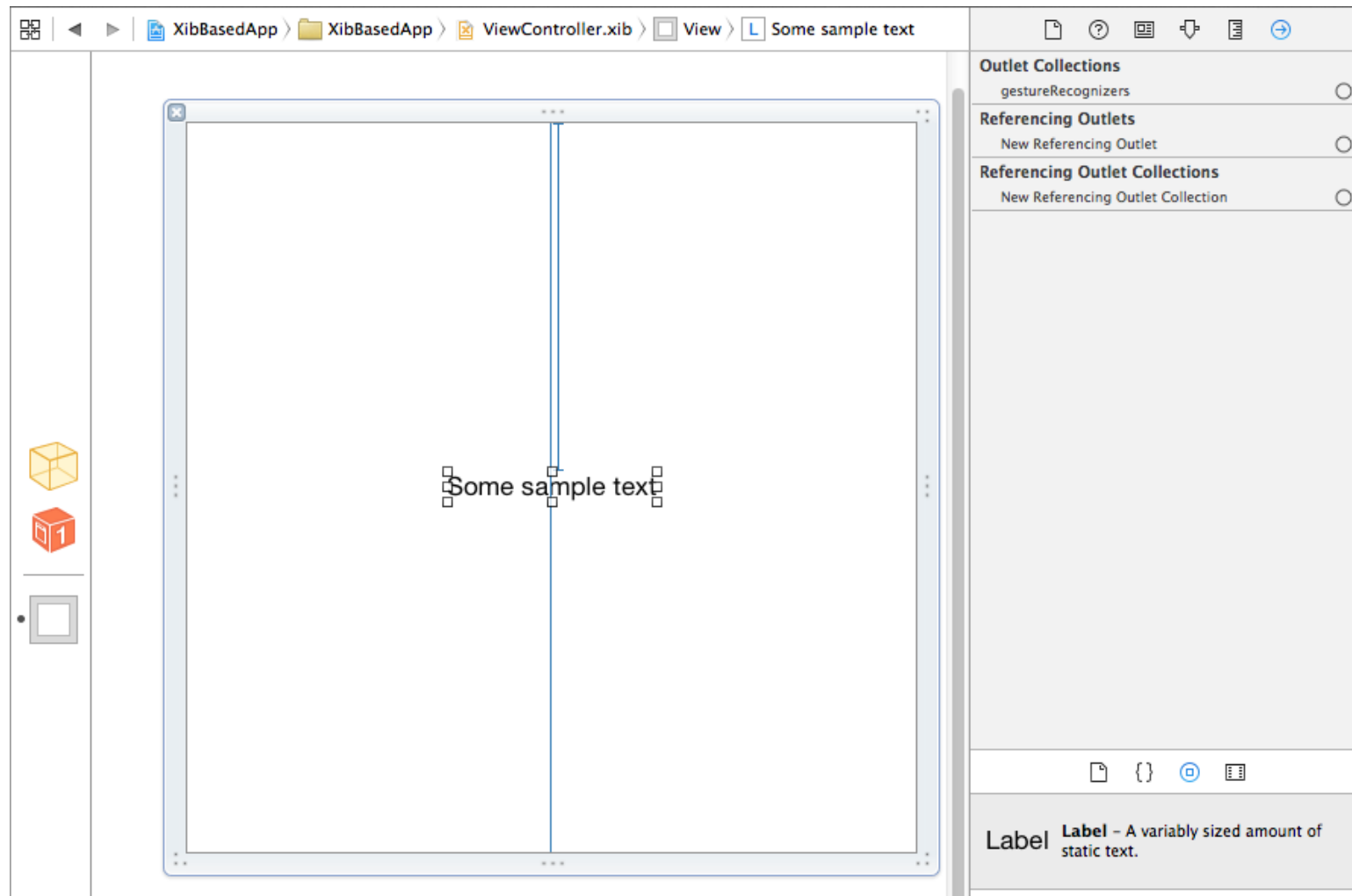
- Im "Connections Inspector"
- Setzen des View Outlets auf die View



- Ergebnis:



Hinzufügen von UI Komponenten wie gehabt...



Ergebnis:

