LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

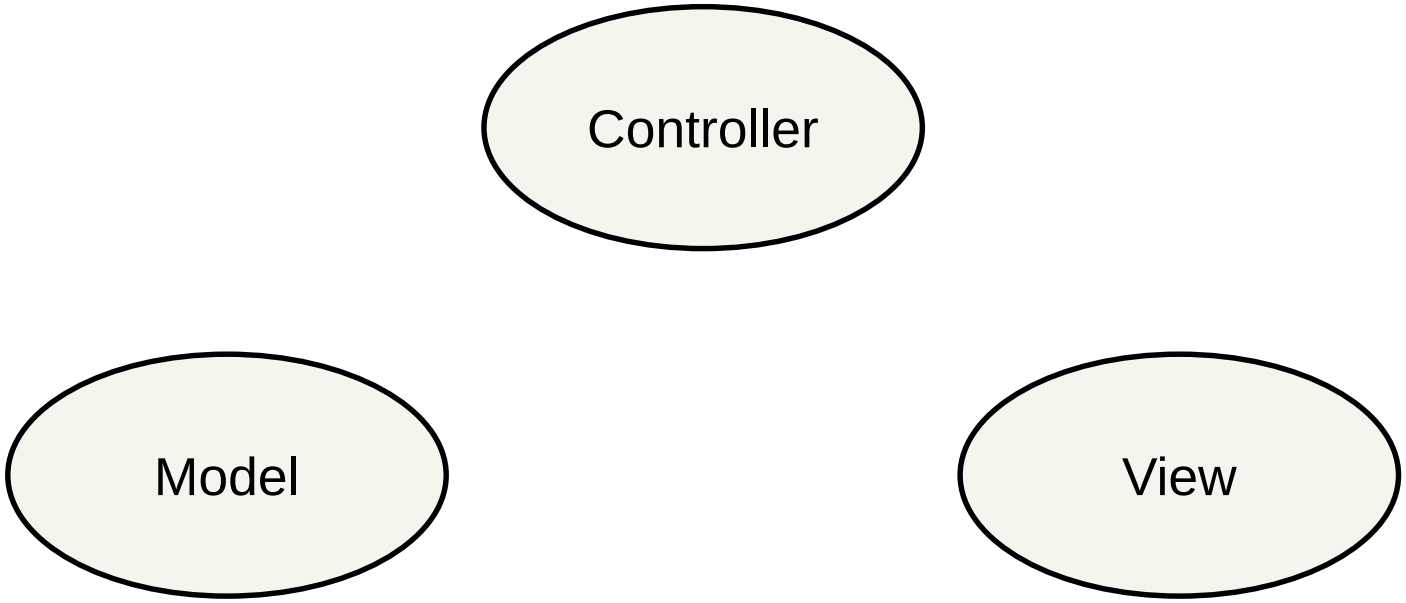
Praktikum iOS-Entwicklung

Sommersemester 2017

Prof. Dr. Linnhoff-Popien
Lenz Belzner, Kyrill Schmid

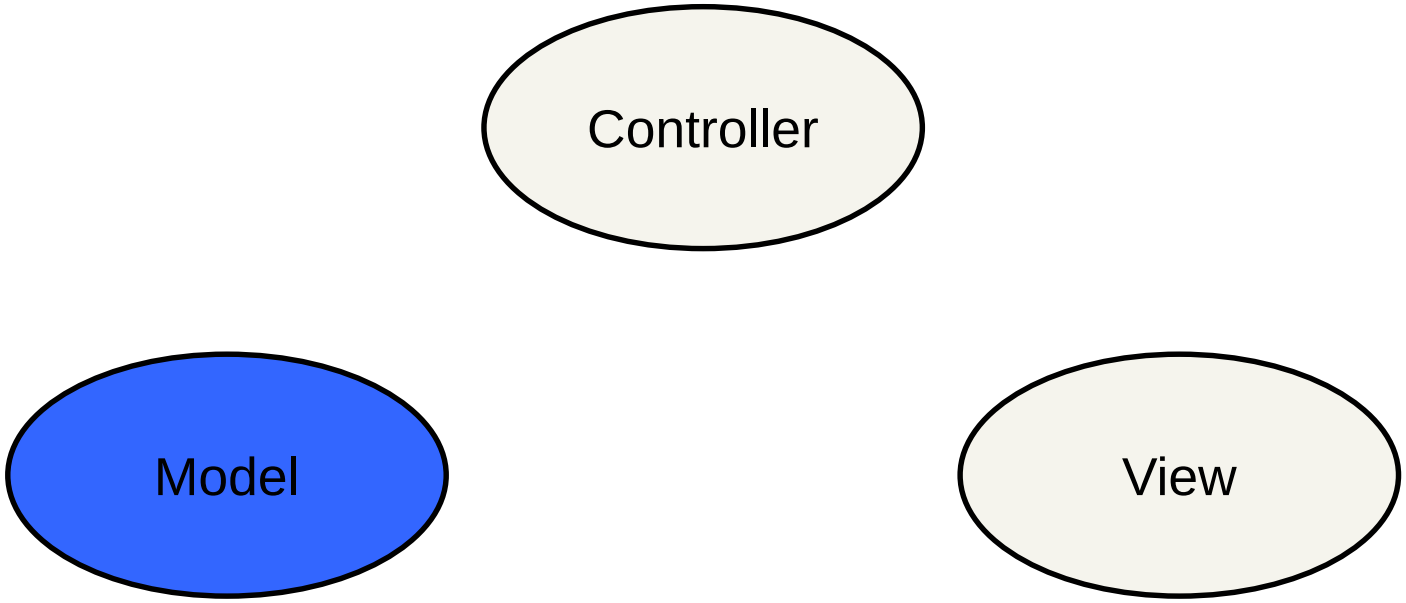
Model-View-Controller

Model-View-Controller (MVC)



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

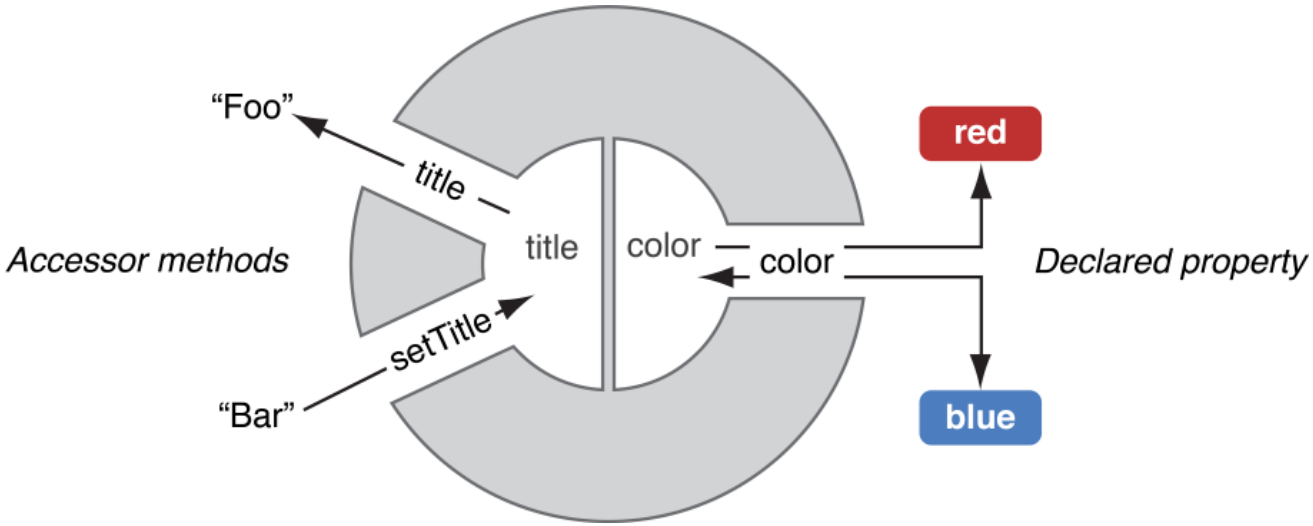
Model-View-Controller (MVC)



Model: Worum handelt es sich bei der Anwendung?

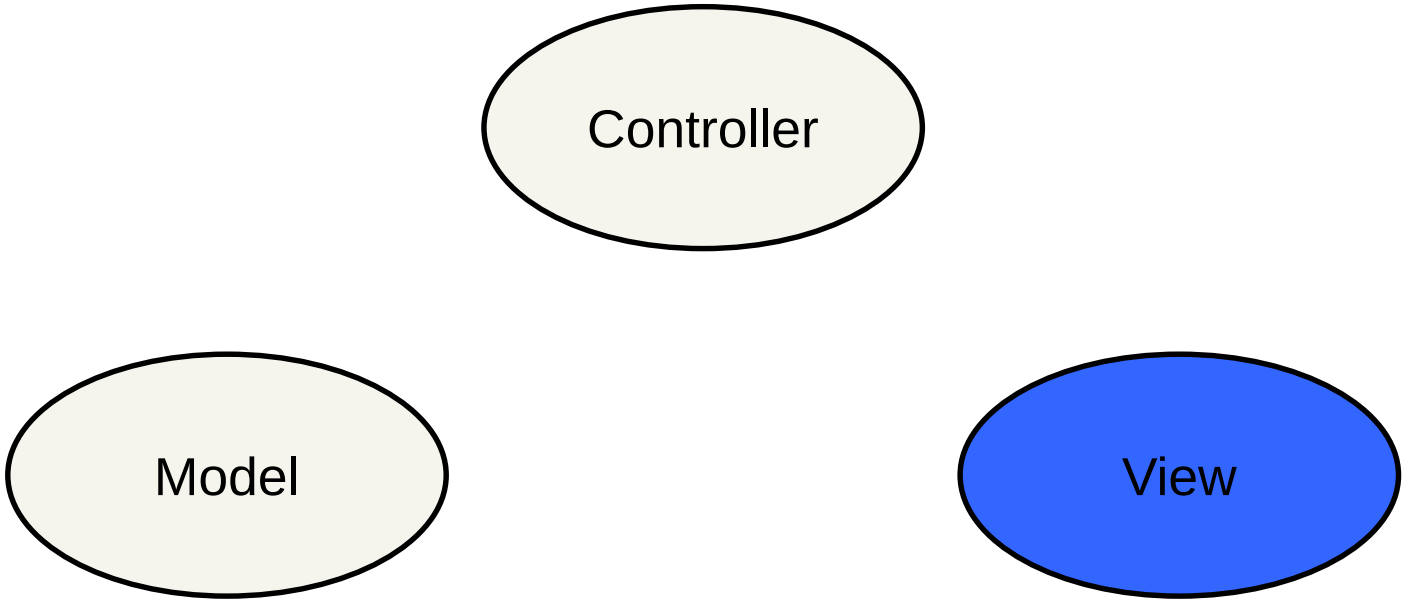
- Enthält Daten bzw. Datenmodell
- Ist unabhängig von der eigentlichen Darstellung!

Model Objekte



- Kapselt Daten, bietet Zugriff auf Daten und implementiert die Logik

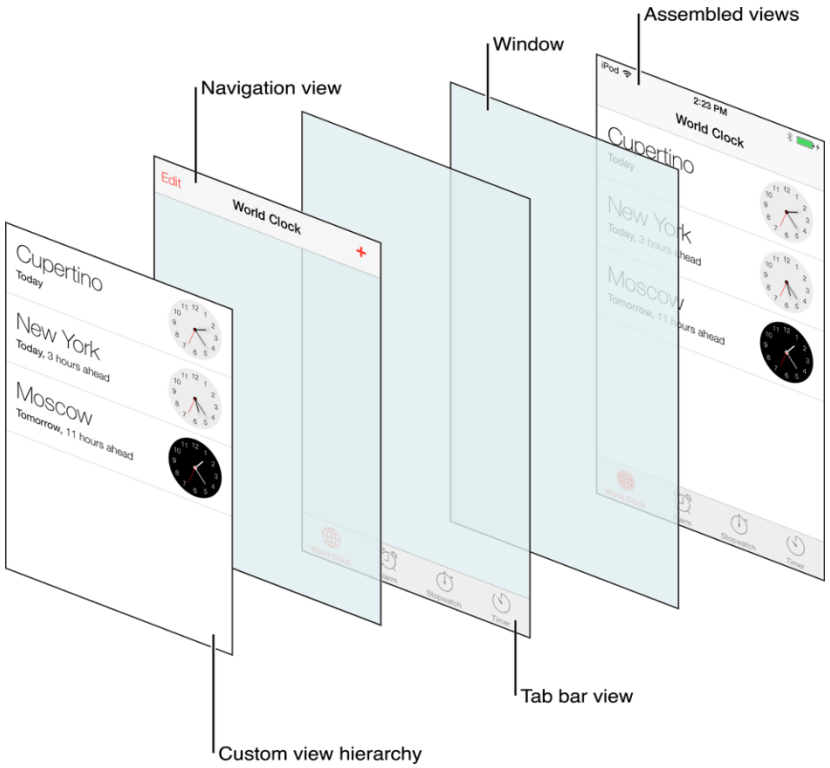
Model-View-Controller (MVC)



View: Darstellung des Model

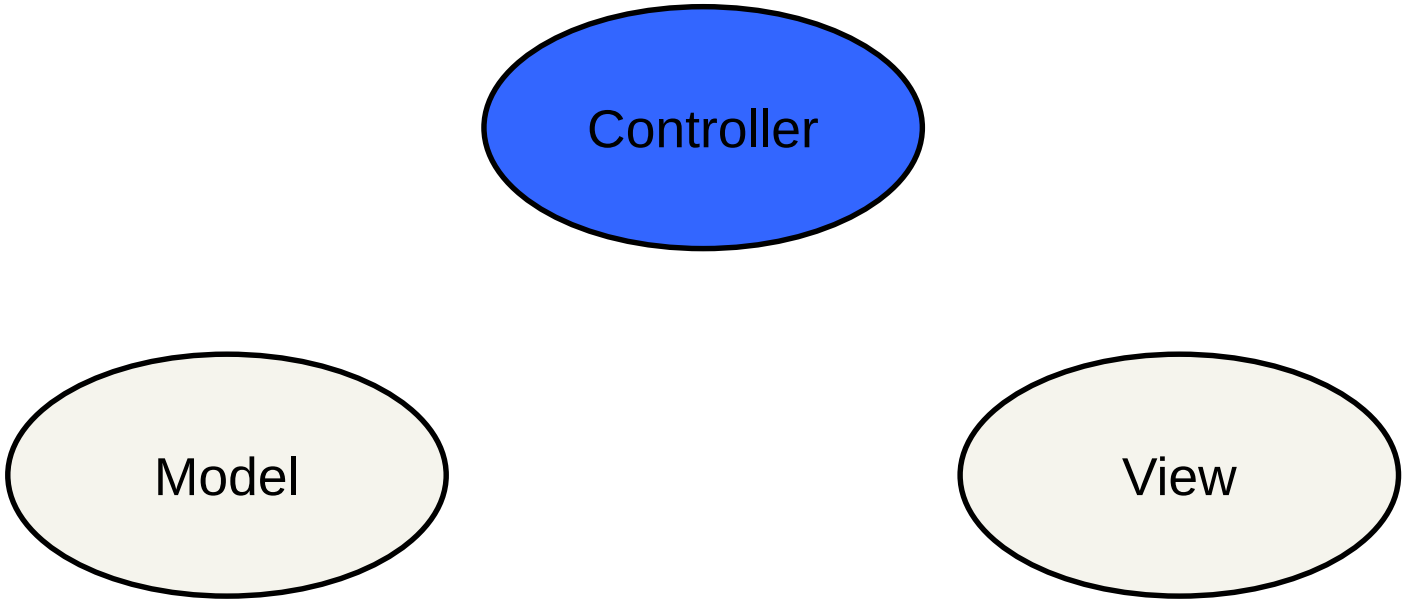
- Schnittstelle zum Benutzer (Interaktion)
- Keine Verarbeitung von Daten!

View Objekte



- View Objekte sind i.d.R. Objekte die User sieht

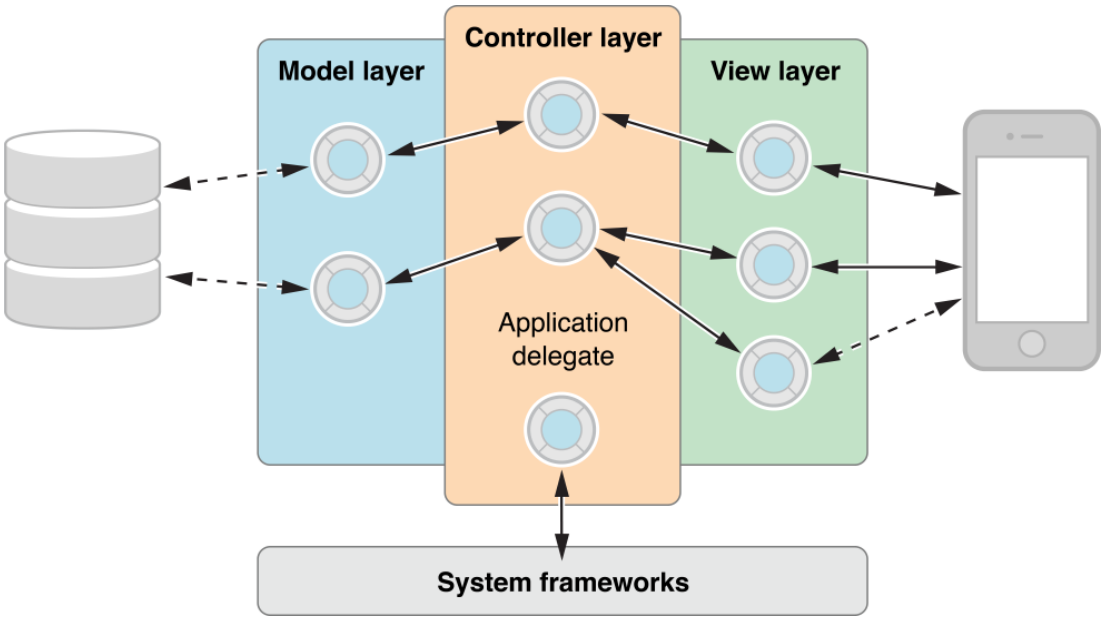
Model-View-Controller (MVC)



Controller: Kontrolliert die Präsentation des Model gegenüber dem Nutzer

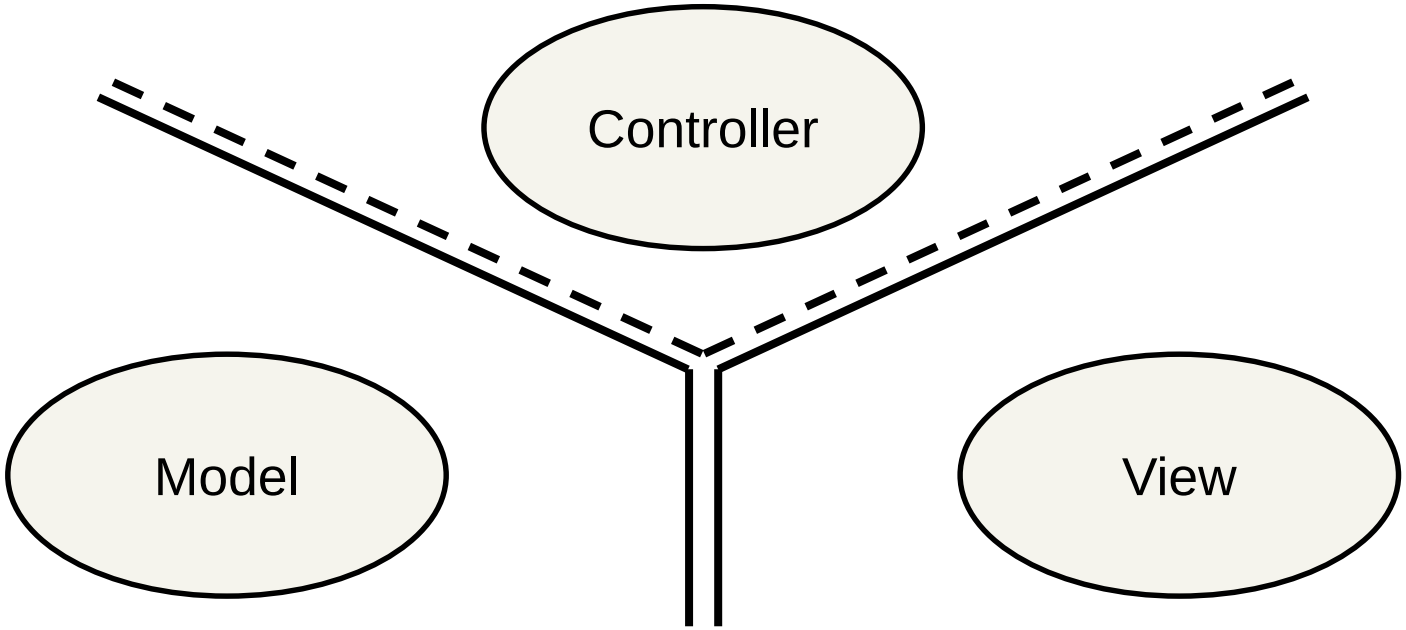
- Vermittlung zwischen Datenmodell und Darstellung (Logik der Darstellung!)
  - Auswertung von Benutzerinteraktionen (View)
  - Manipulation von Daten (Model)
  - Zu jeder View existiert genau ein Controller

Controller Objekte



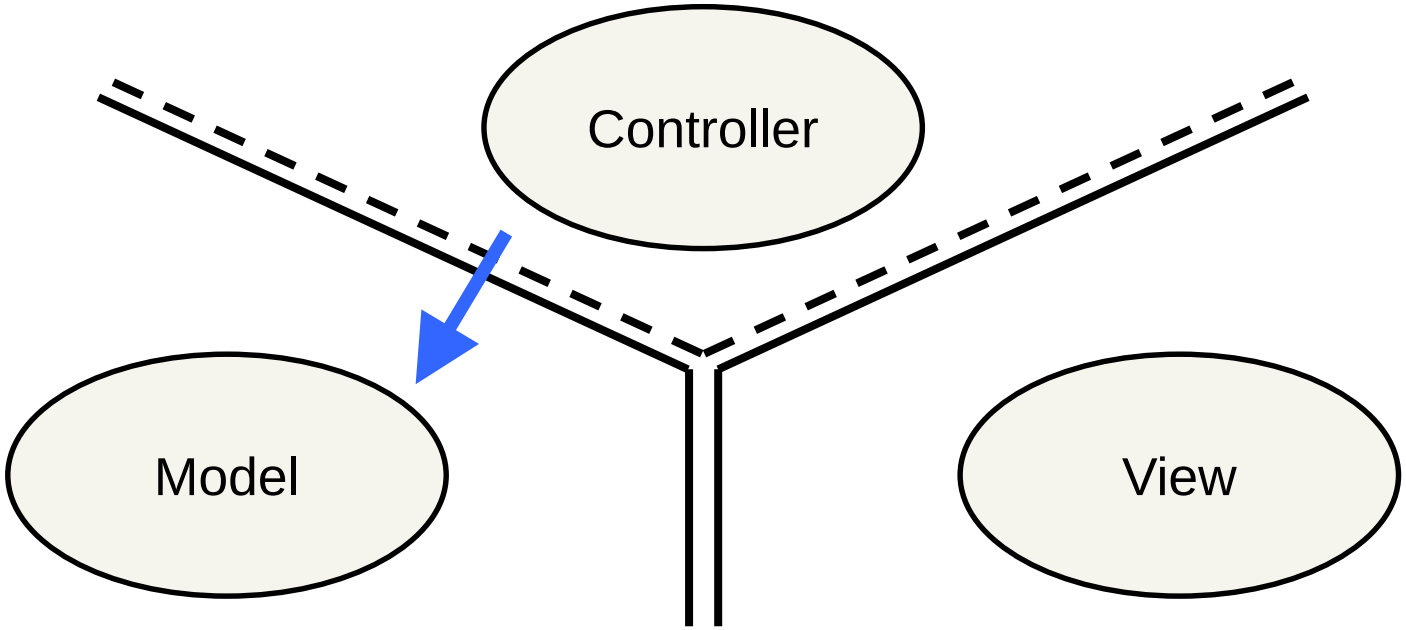
- Verbindungsglied für View- und Model Objekte
- Übernehmen Setup und koordinative Aufgaben
- Interpretieren User Actions und kommuniziert Veränderungen

Model-View-Controller (MVC)

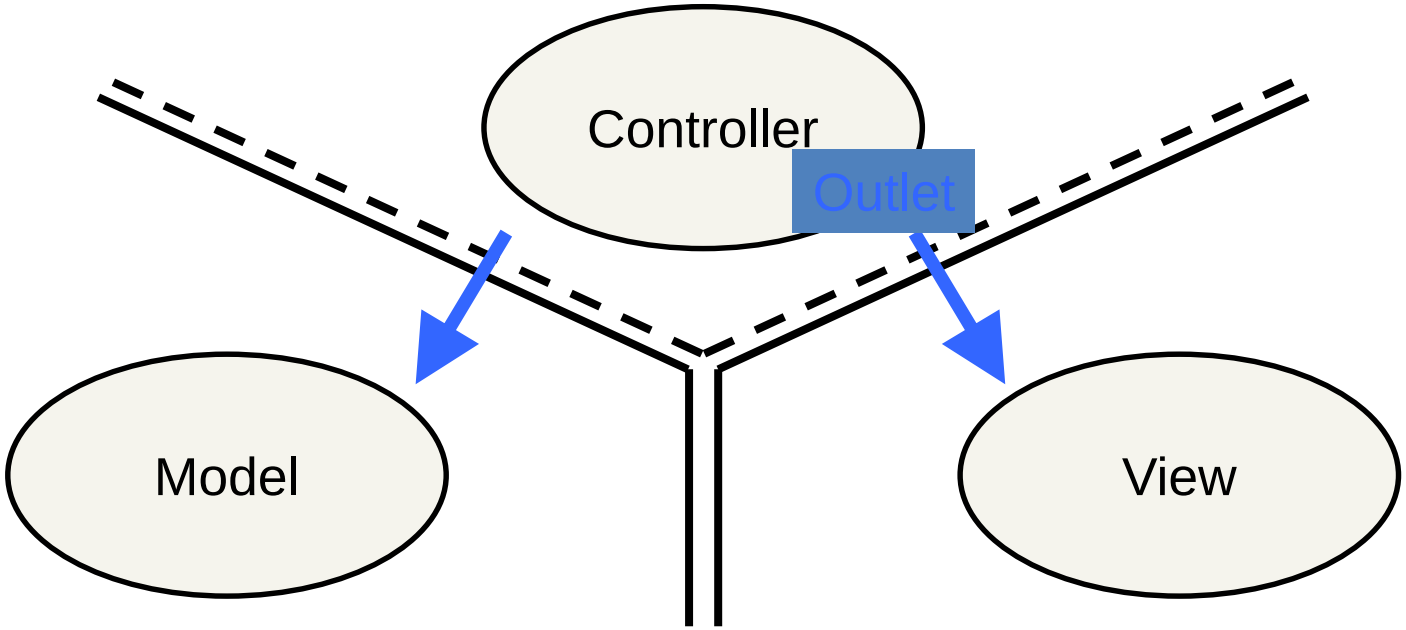


Welche Art der Kommunikation zwischen den drei Gruppen ist erlaubt?

Model-View-Controller (MVC)

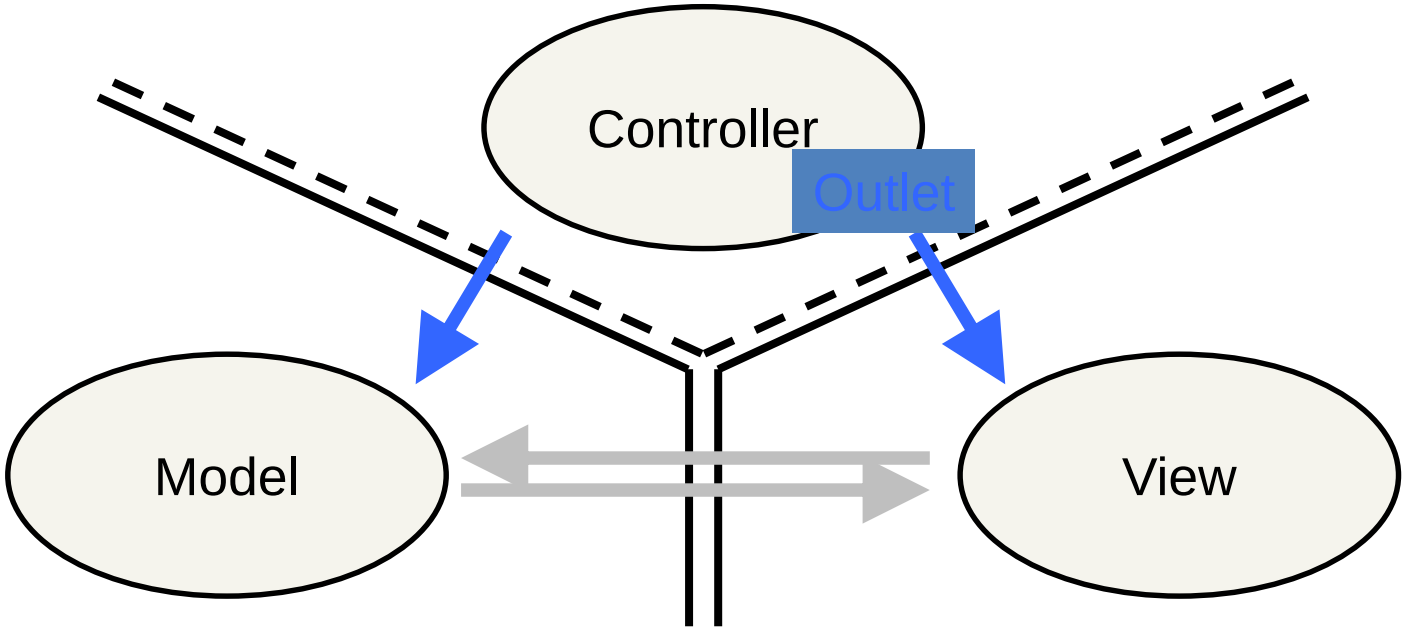


Controller kann immer direkt auf sein Model zugreifen



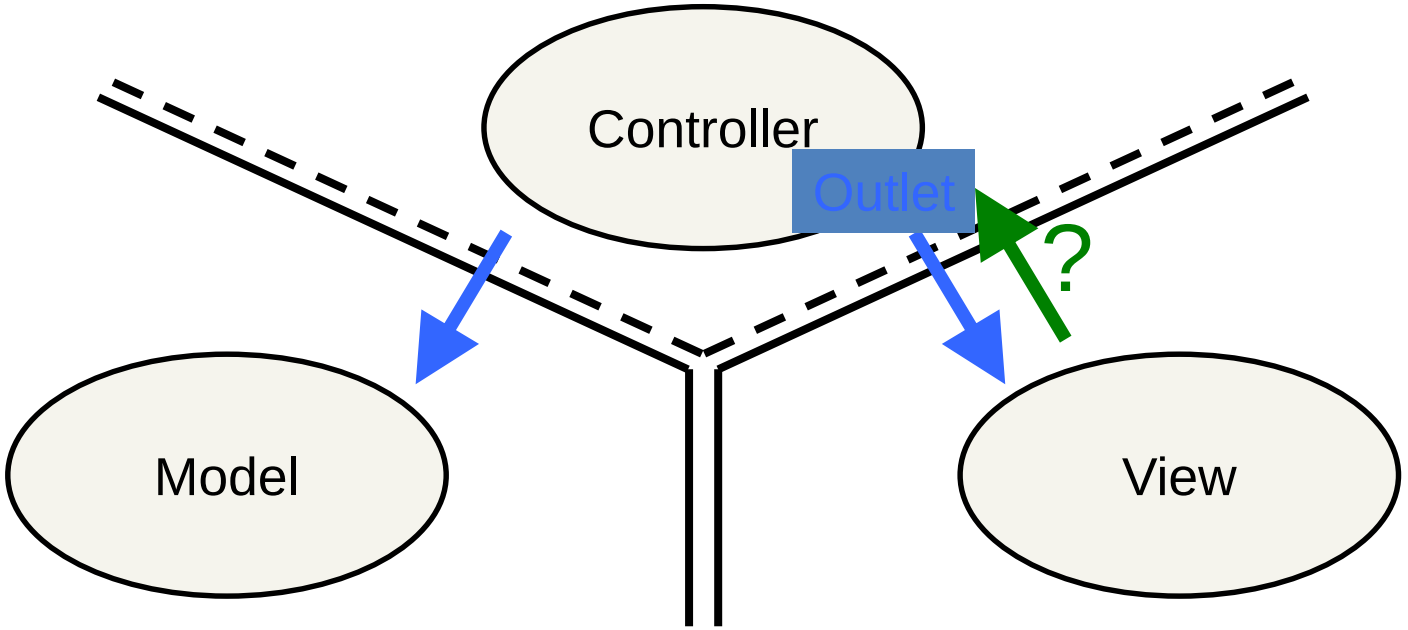
Controller kann direkt Nachrichten an seine View senden

Model-View-Controller (MVC)

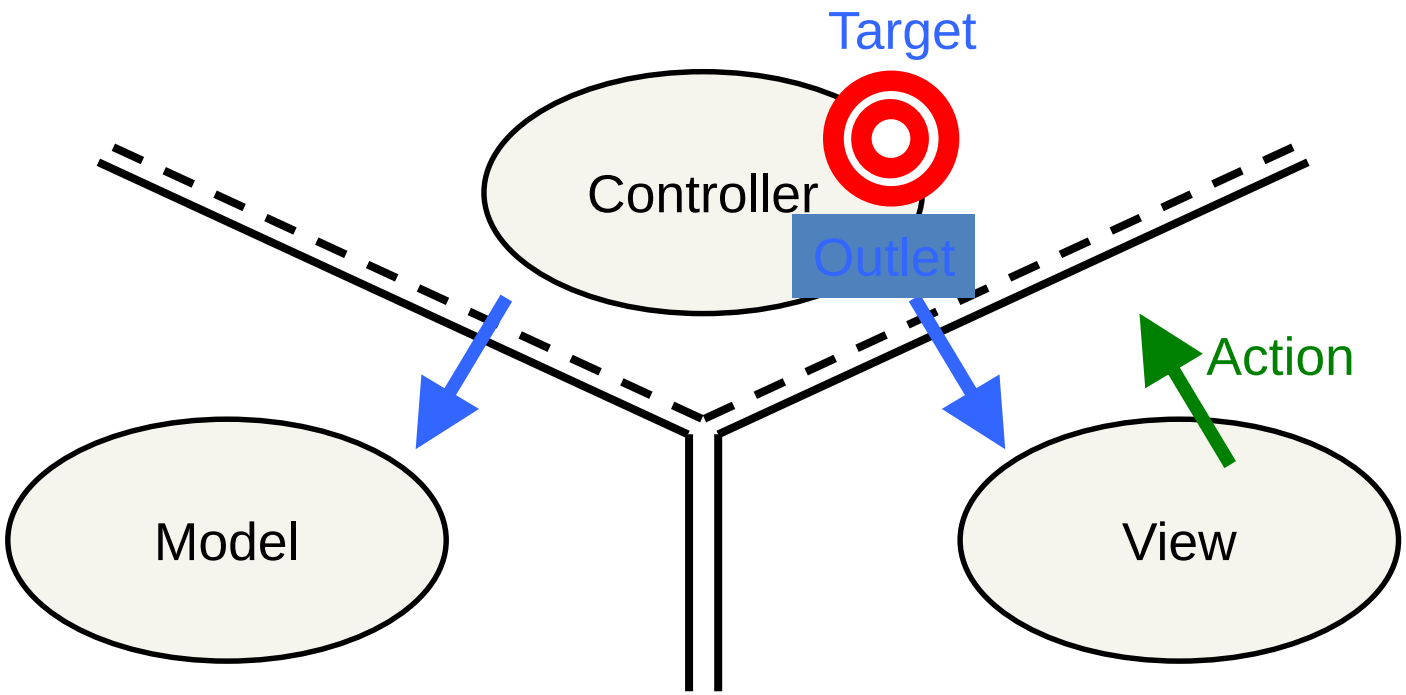


Model und View sollten niemals miteinander kommunizieren!

Model-View-Controller (MVC)

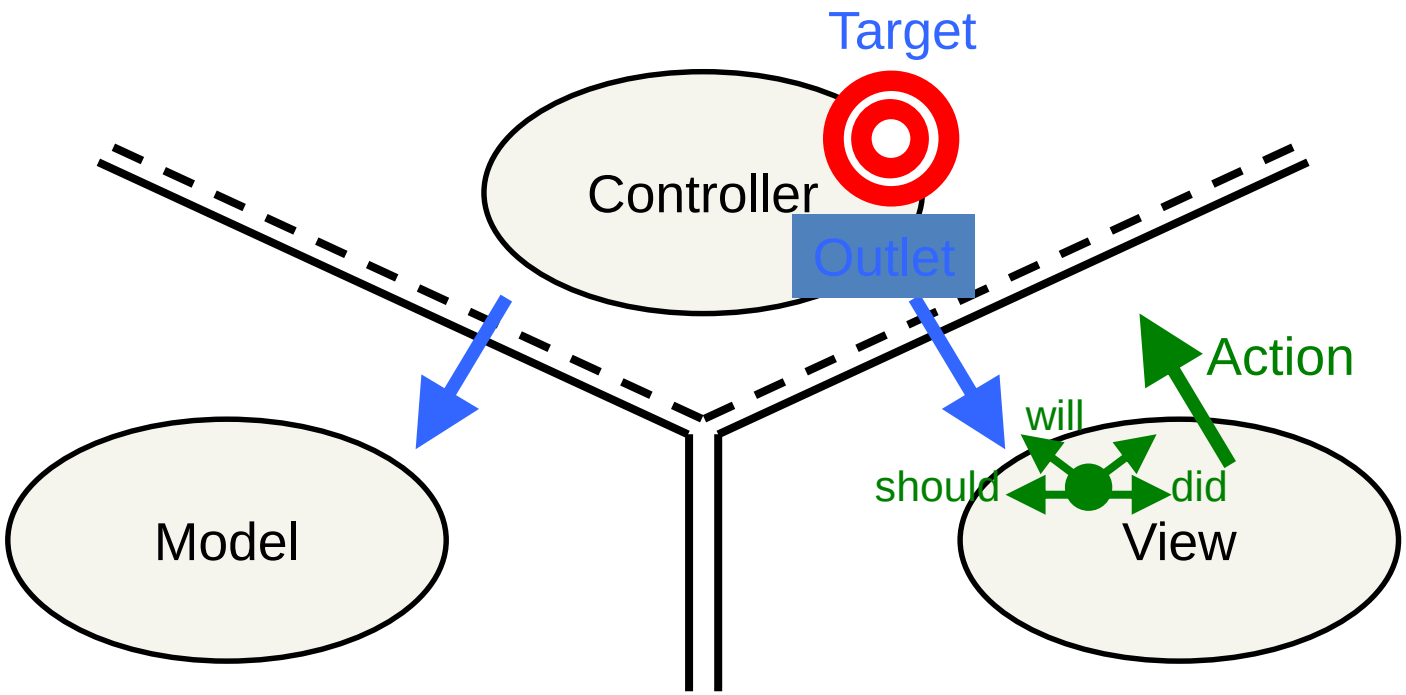


- Wie werden Interaktionen des Benutzers dem Controller kommuniziert?
- Kann die View Nachrichten an den Controller senden?



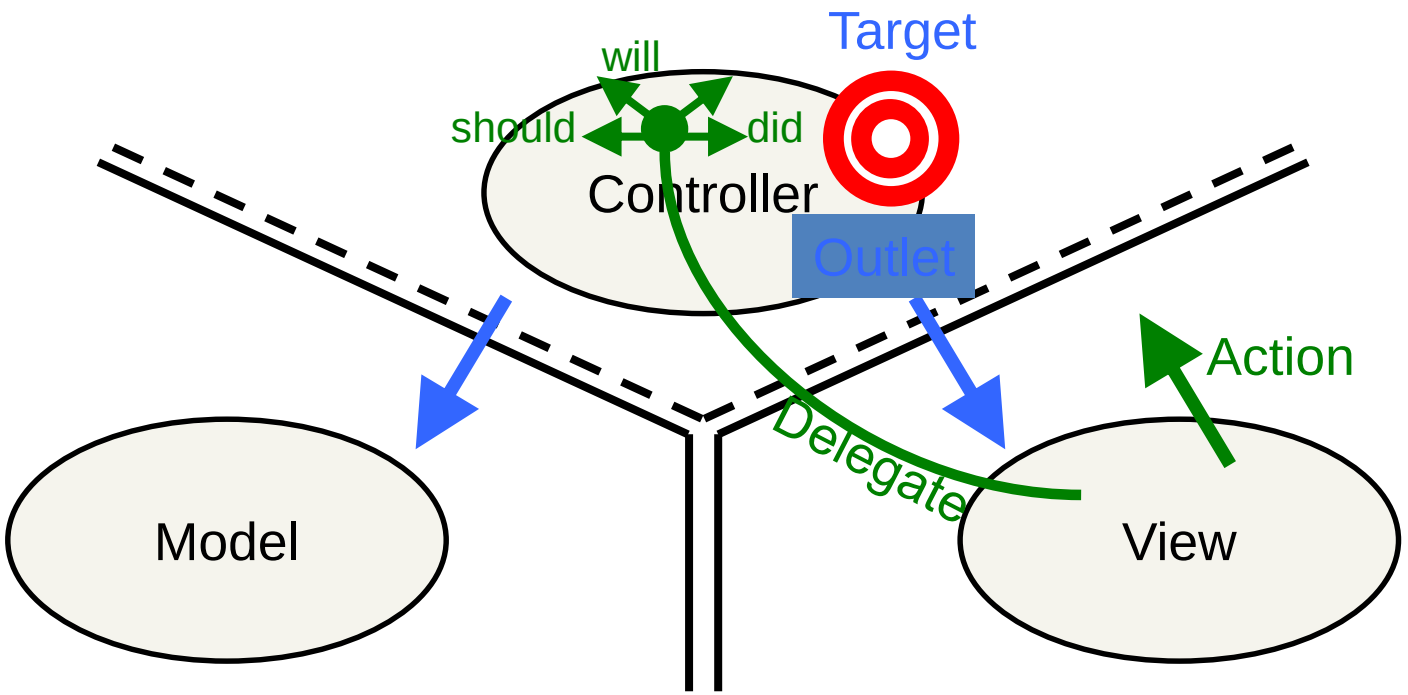
- Controller kann der View sich selbst als Zielobjekt (Target) bestimmter Interaktionen mitteilen
- Kommunikation von Ereignissen erfolgt als Aktion (Action)

Model-View-Controller (MVC)



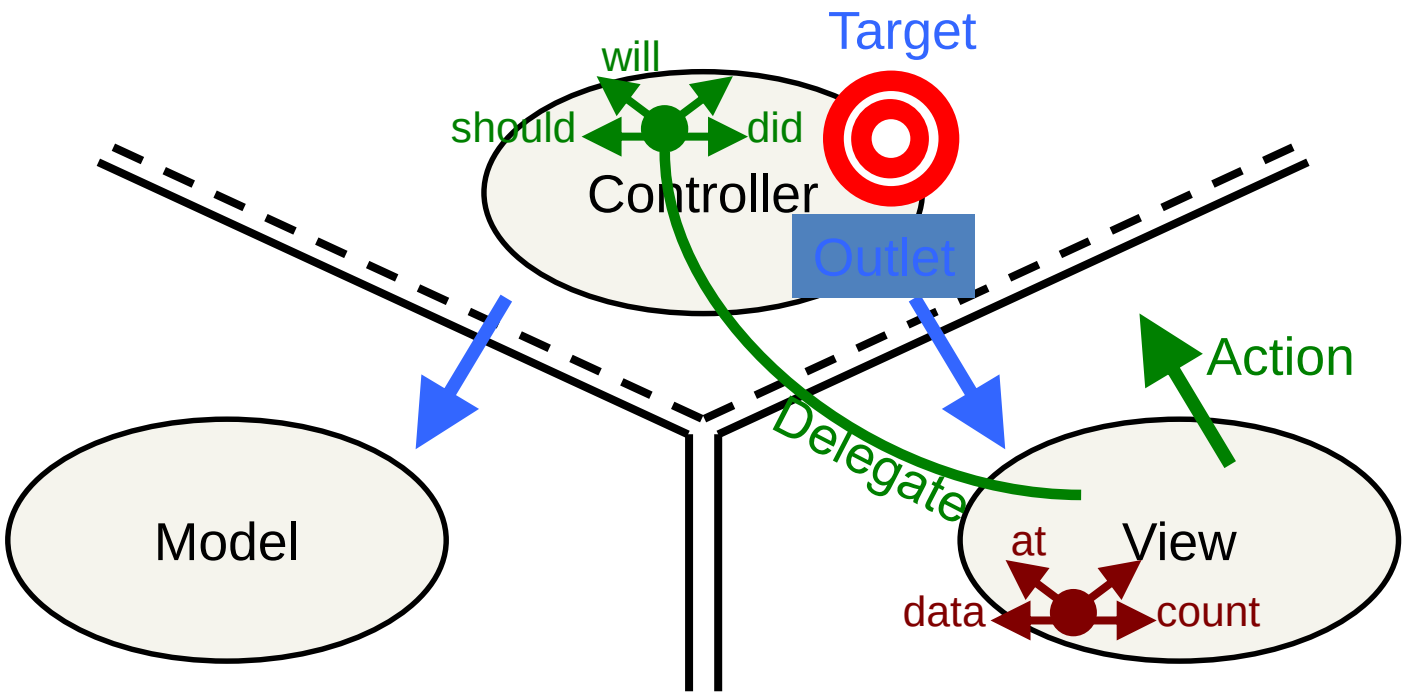
- Manchmal müssen sich eine View und ihr Controller unabhängig von Nutzerinteraktionen synchronisieren

Model-View-Controller (MVC)



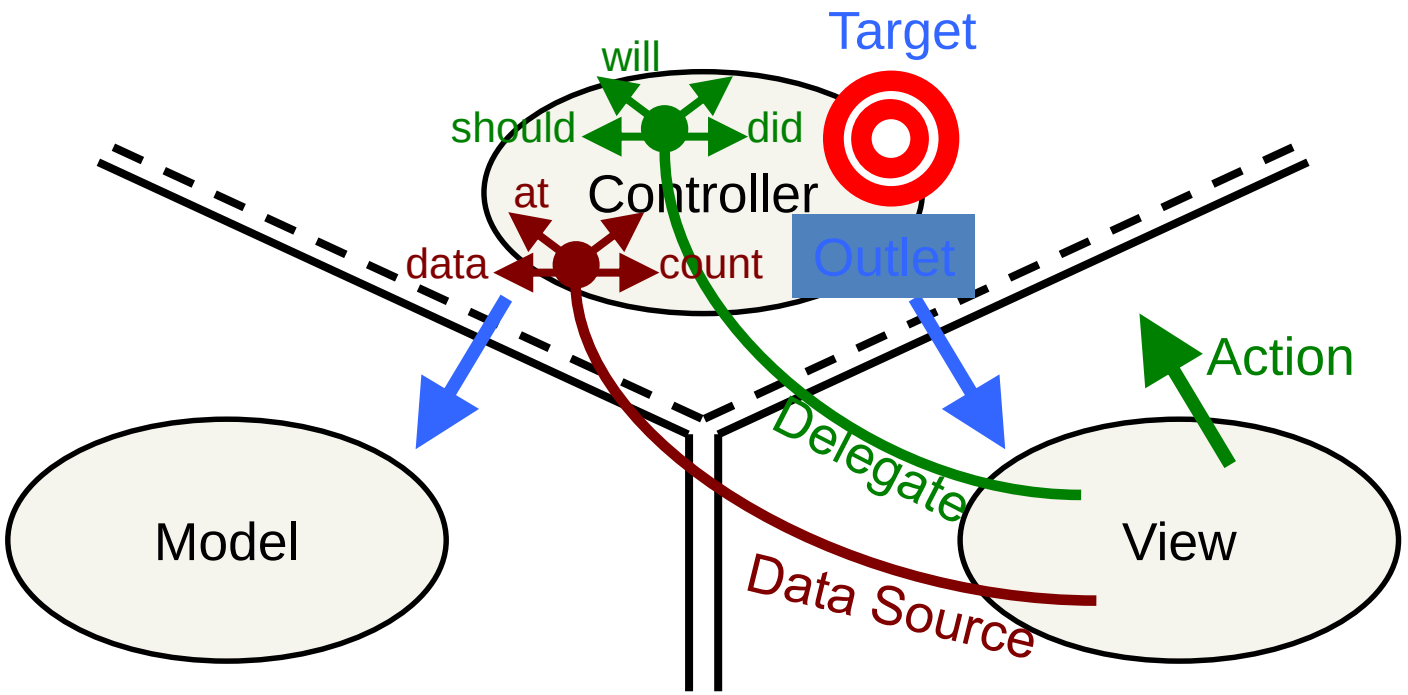
- Synchronisation erfolgt über Delegation (Delegate)
- Delegates werden über Protocols realisiert (ähnlich der Realisierung eines Interface in Java)
- Beispiel:
  - `UITableViewDelegate` Protocol, `tableView:didSelectRowAtIndexPath:`)

Model-View-Controller (MVC)

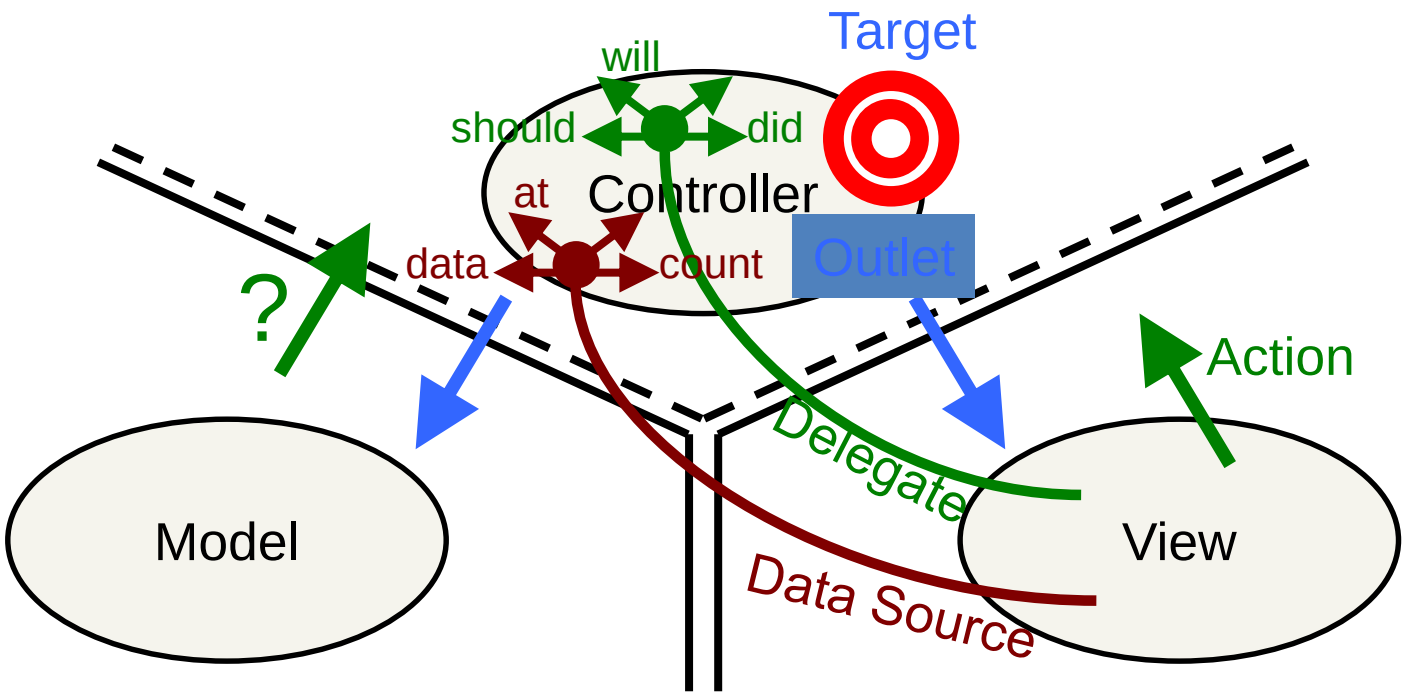


- Views besitzen nicht die Daten (Model), die sie darstellen
- Views verwenden ebenfalls ein Protocol, wenn sie Daten benötigen

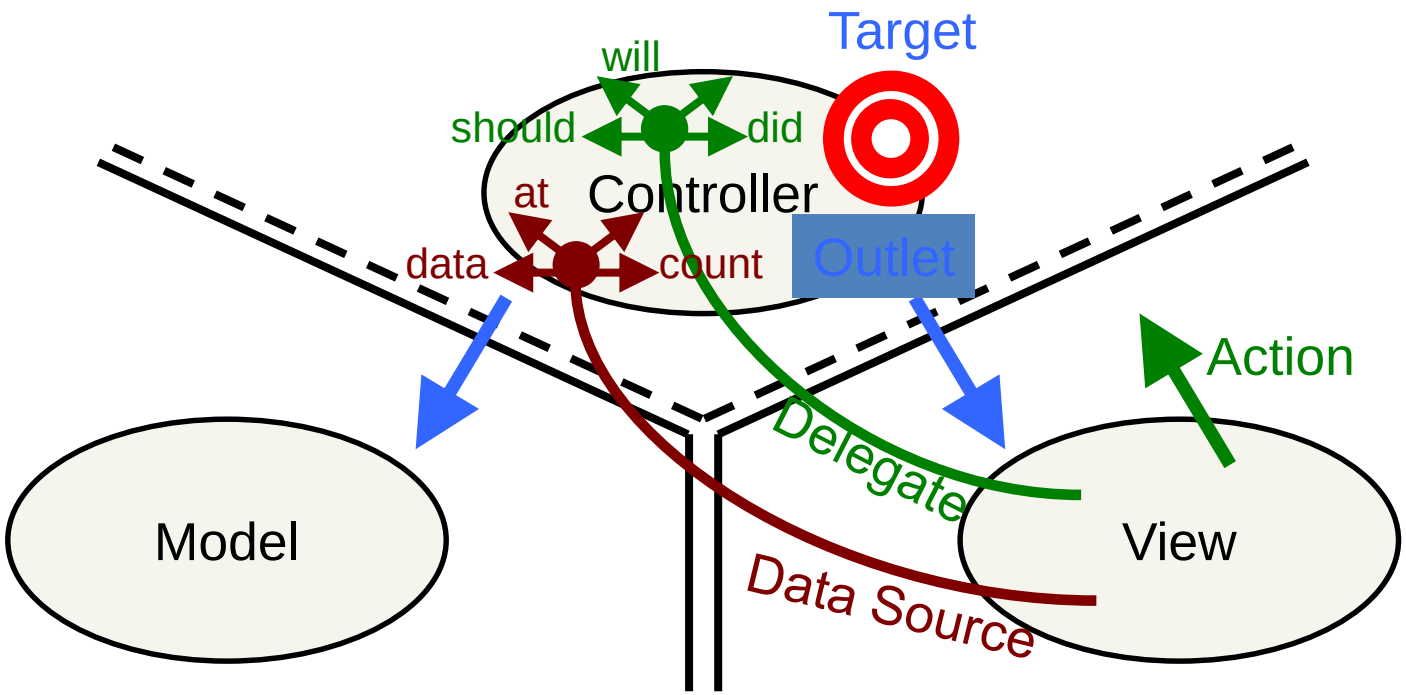
Model-View-Controller (MVC)



- 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:`

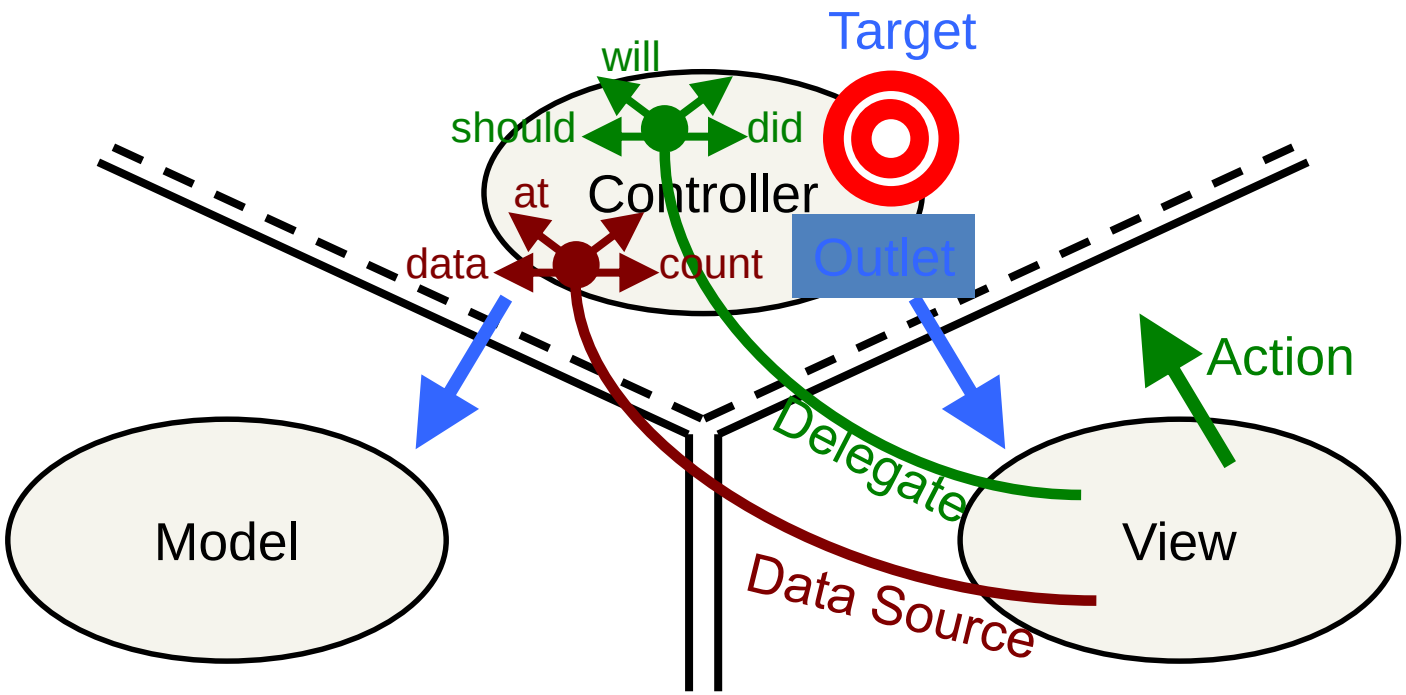


Frage 1: Kann das Model dem Controller Nachrichten senden?

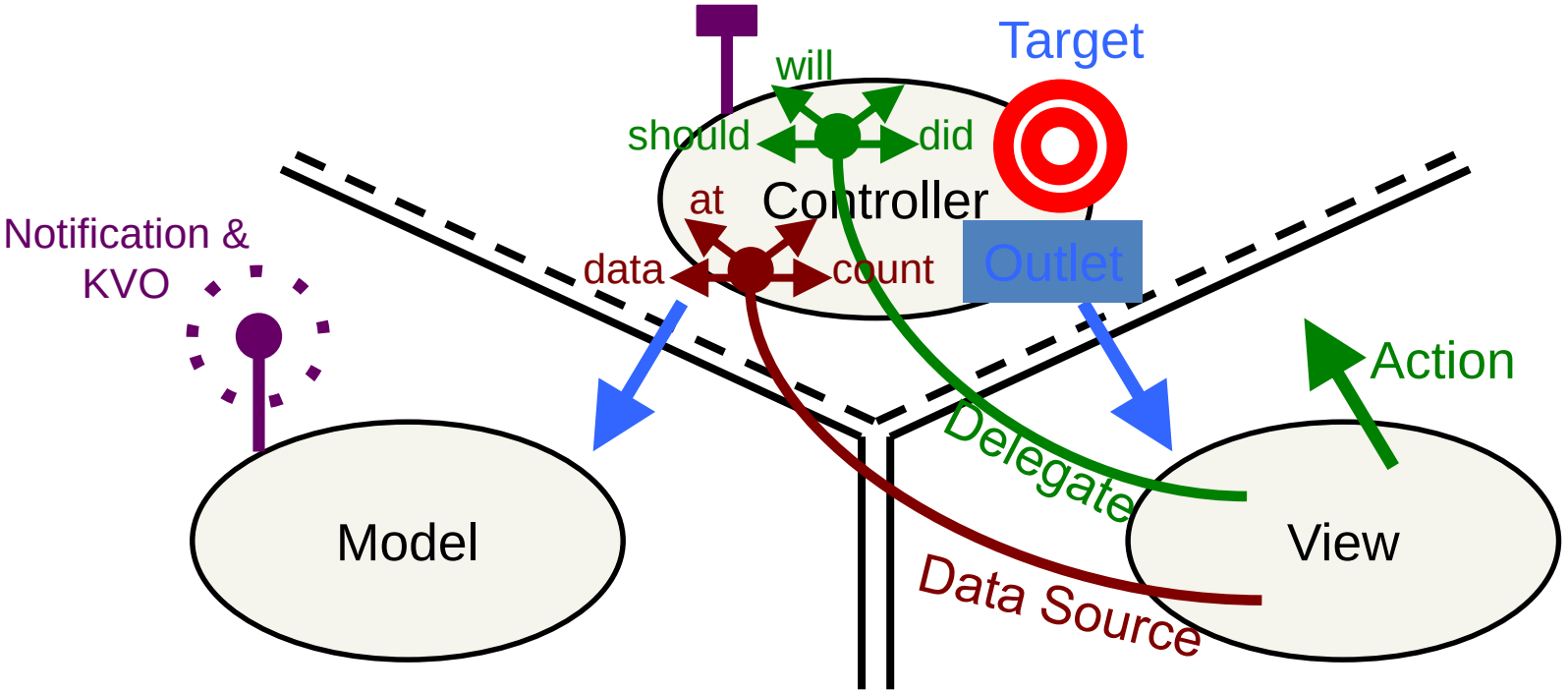


Frage 1: Kann das Model dem Controller Nachrichten senden?

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



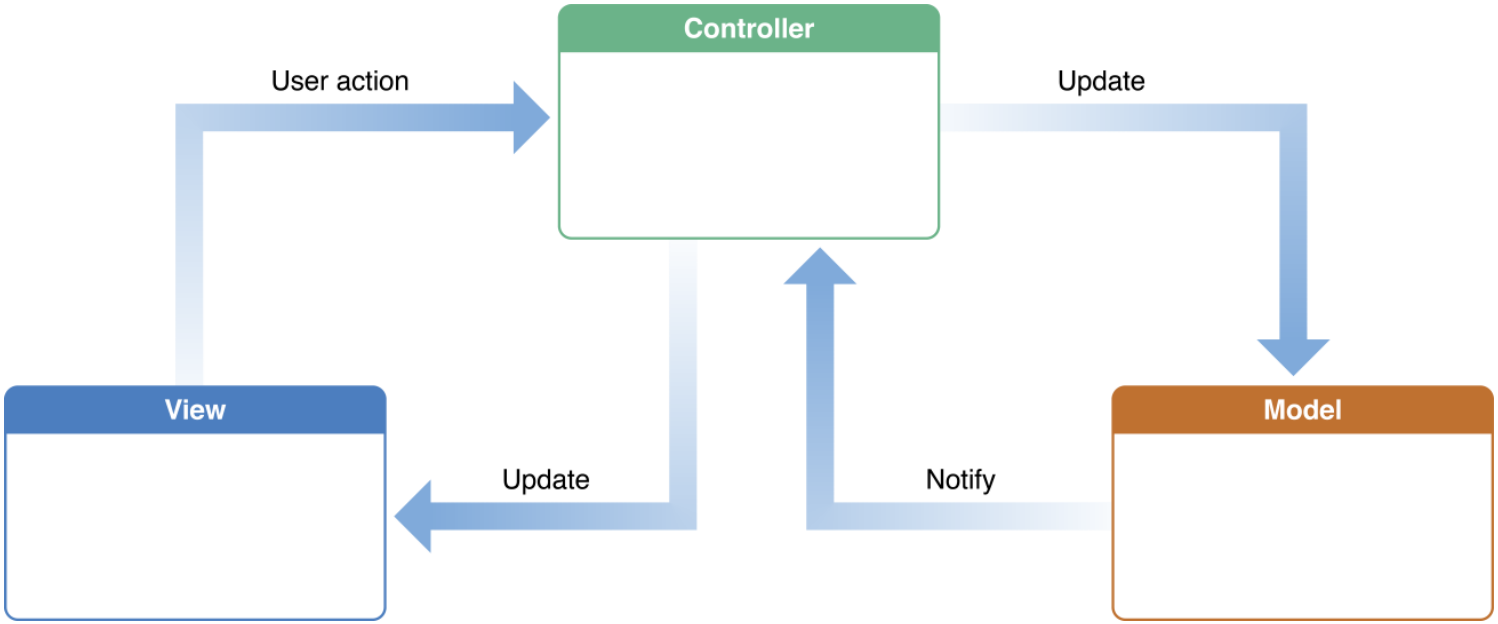
Frage 2: Wie werden Modifikationen an den Daten des Model dem Controller mitgeteilt bzw. in der View aktualisiert?



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

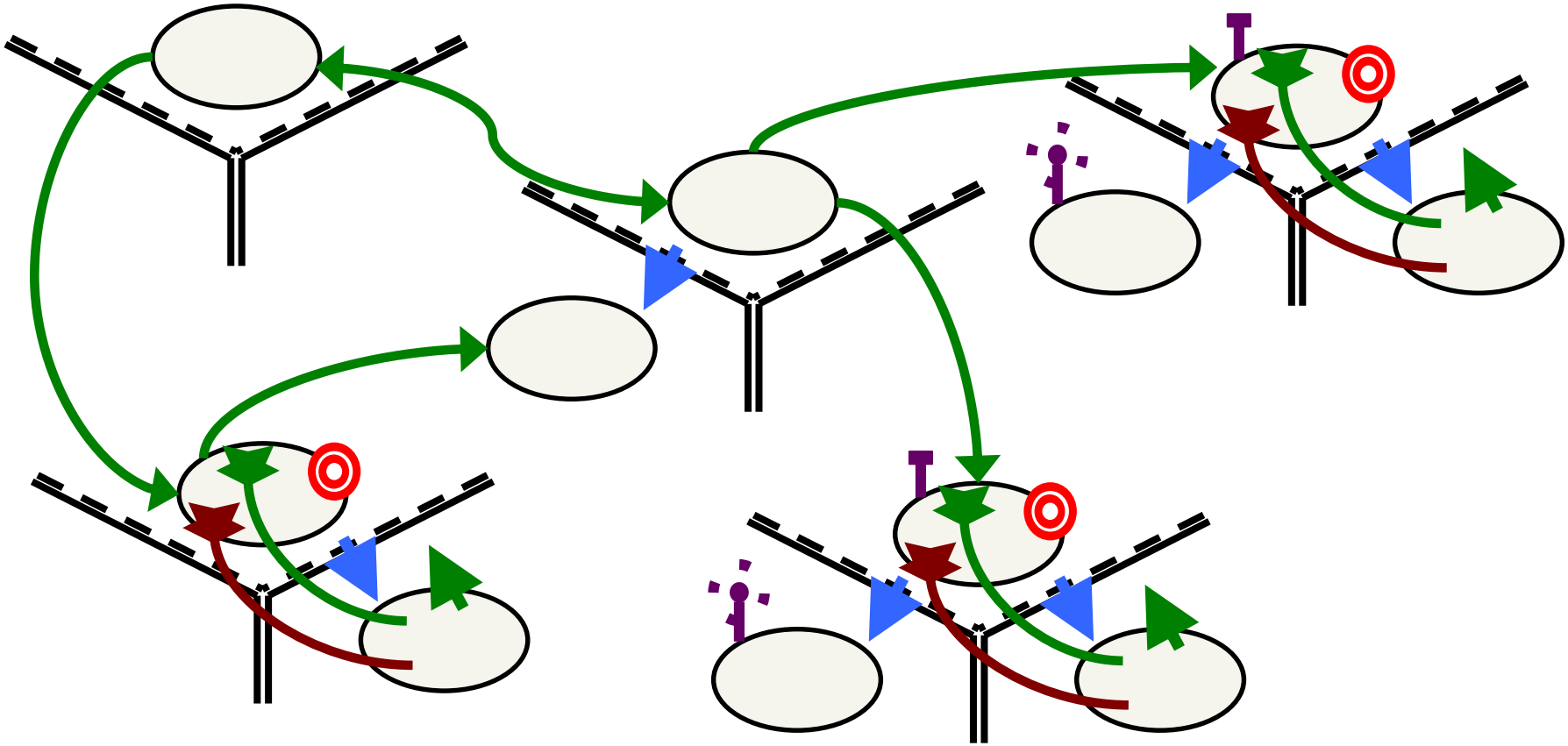
- Verwendung eines Broadcast Mechanismus (Notifications und Key-Value-Observing (KVO))
- Controller "lauschen" nach interessanten Nachrichten bzw. Veränderungen

Model-View-Controller (MVC)



- Kommunikation erfolgt über verschiedene Mechanismen (Delegation, KVO)

Model-View-Controller (MVC)



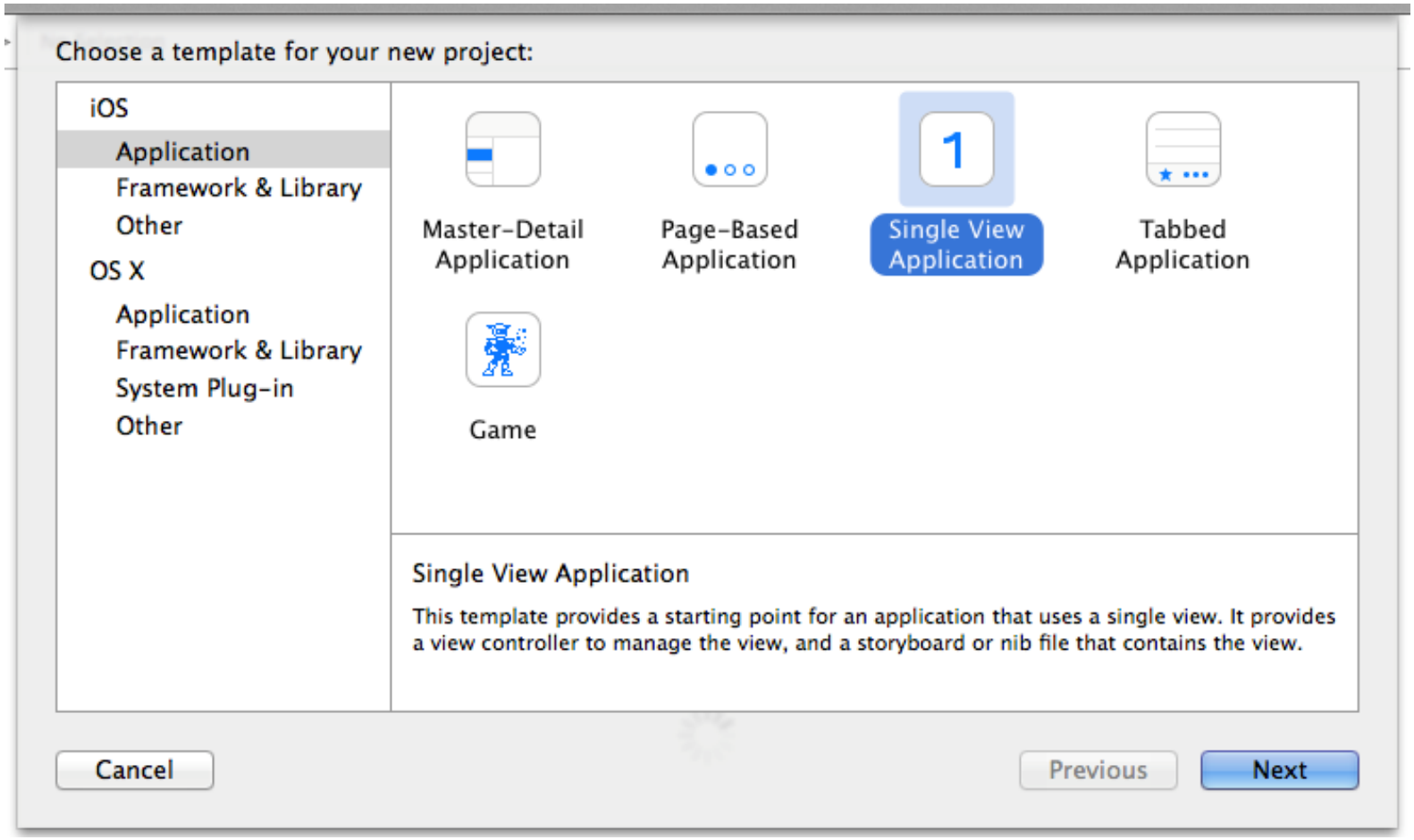
Komplexe Programme entstehen durch die Kombination mehrerer MVC-Gruppen

Erste App in Objective-C (Demo)

DEMO: MyQuiz

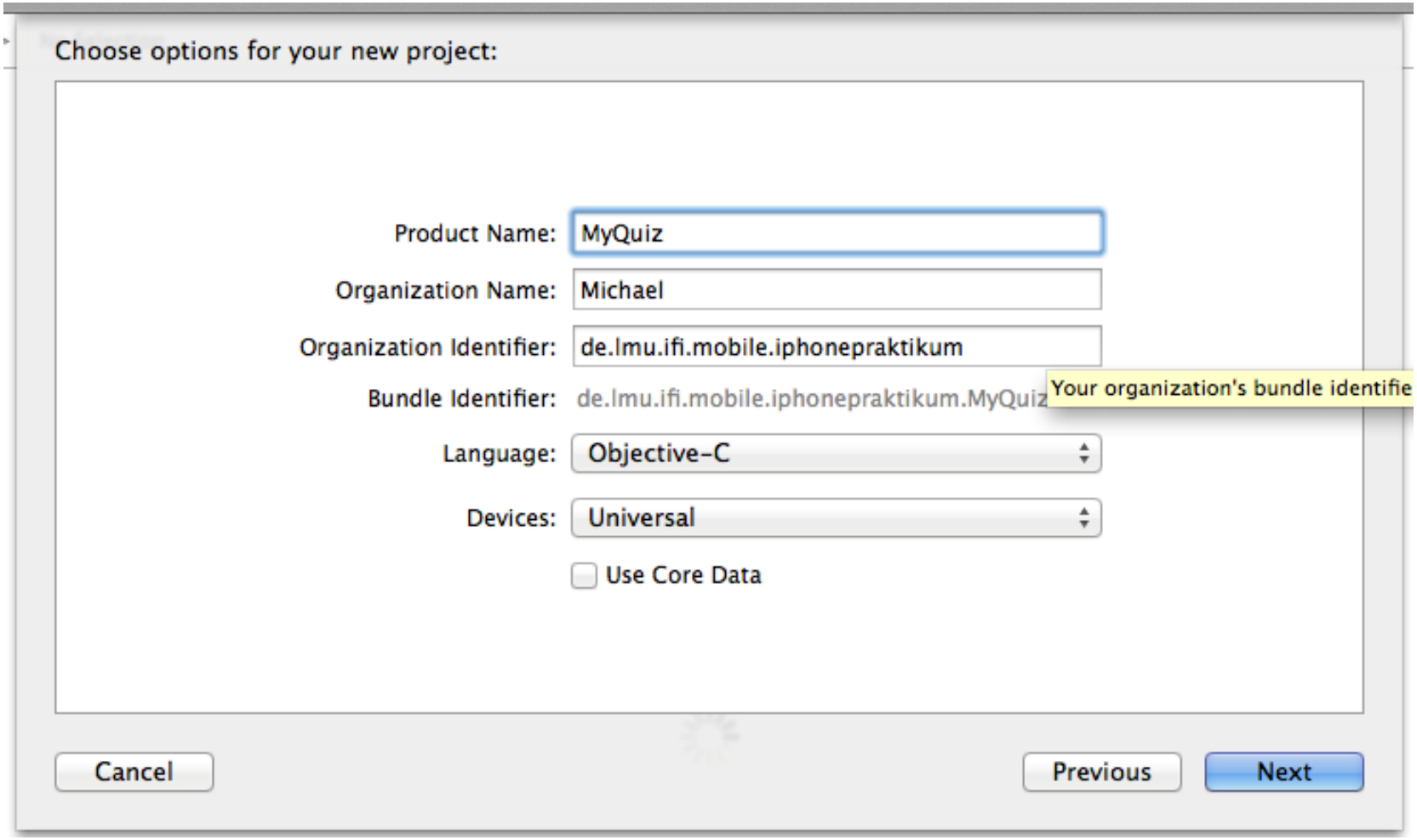
Create New Xcode Project

DEMO: MyQuiz



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

DEMO: MyQuiz

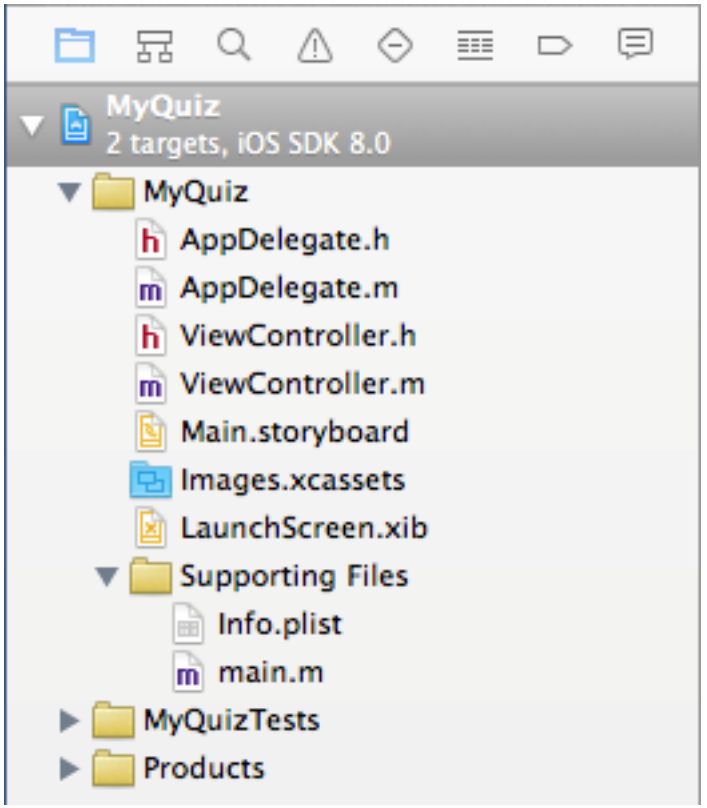


Setzen von Product Name und Organization Identifier

DEMO: MyQuiz

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!



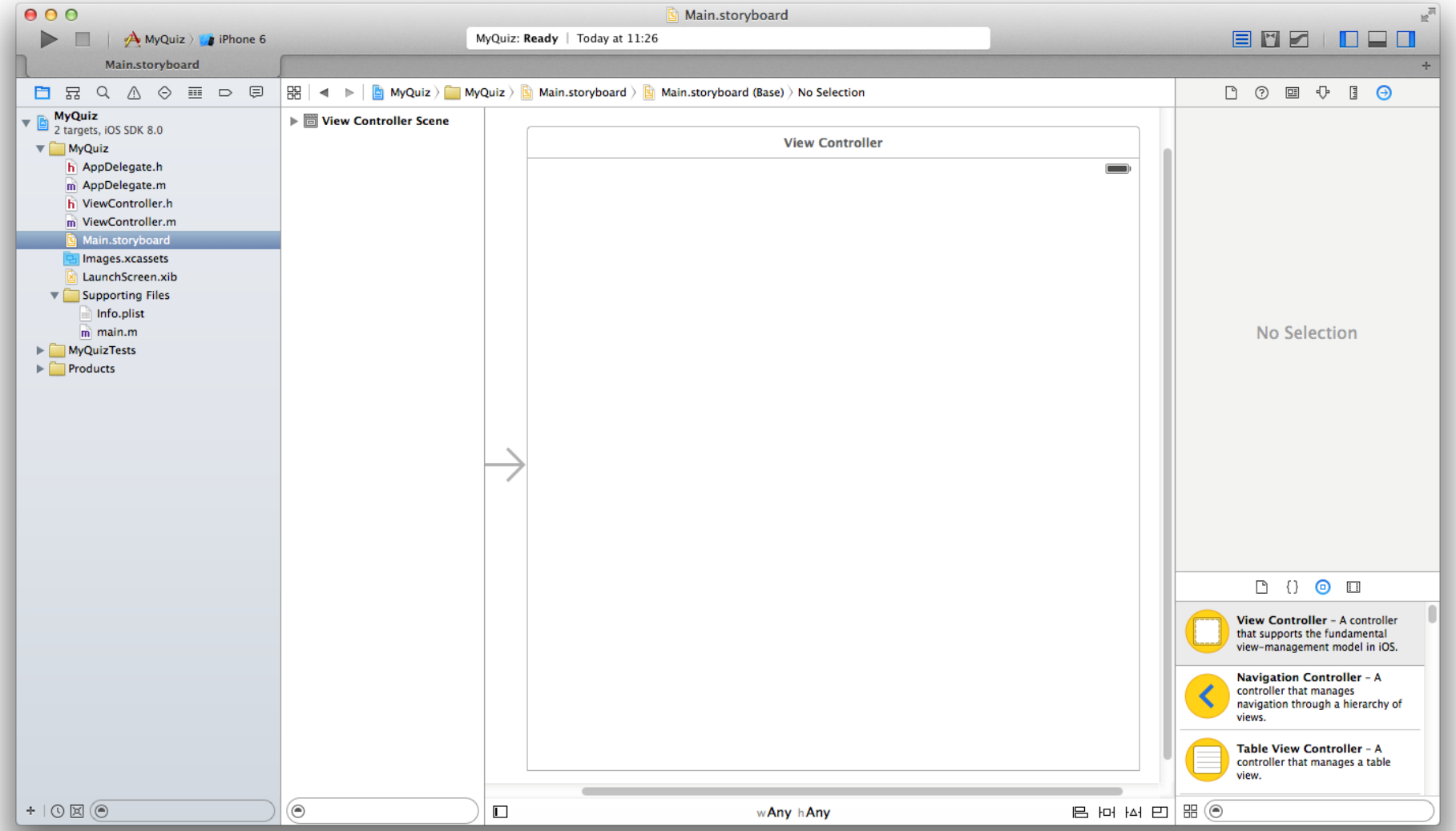
DEMO: MyQuiz


```
// 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
```

DEMO: MyQuiz

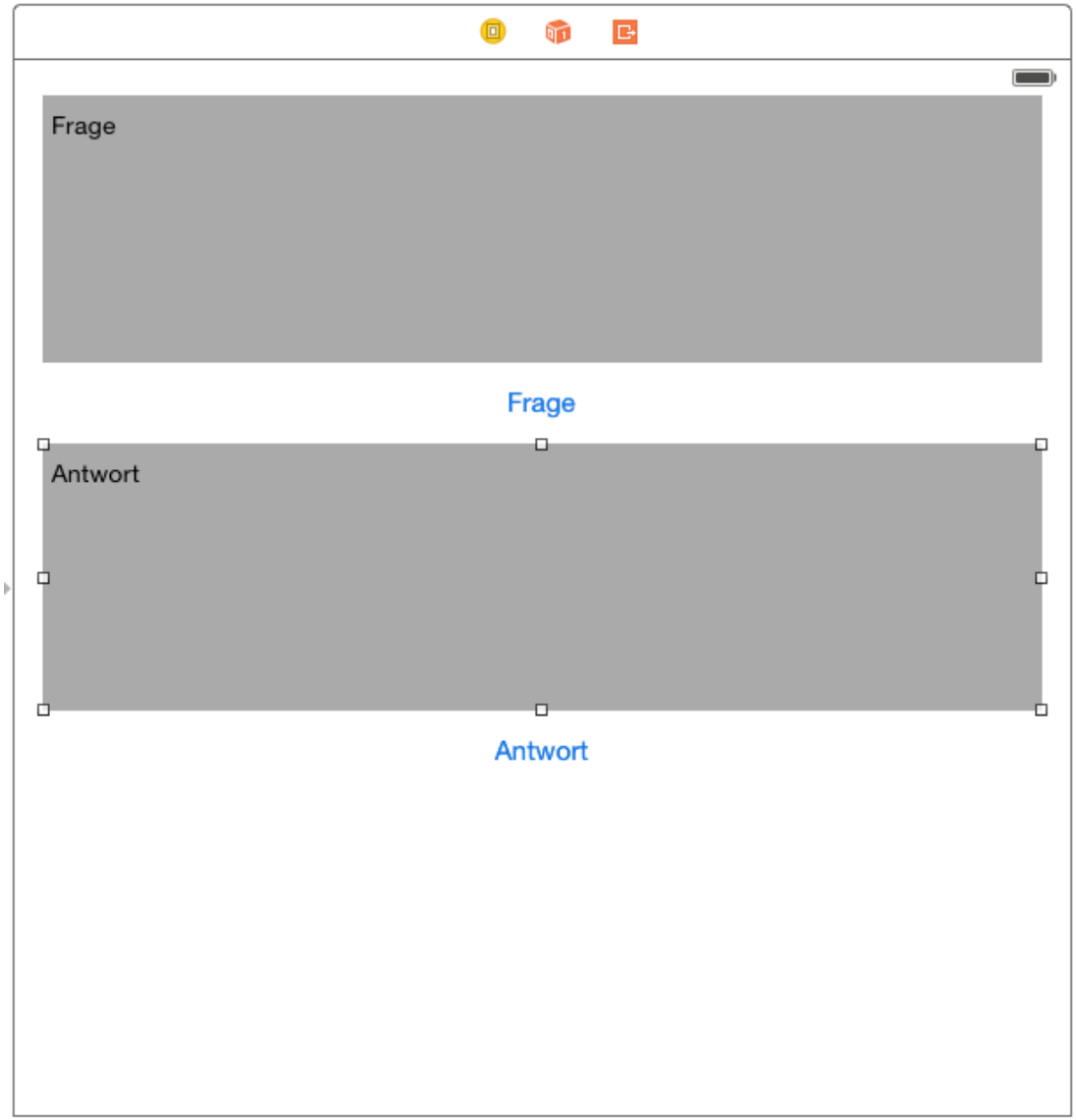
Die View wird innerhalb eines Storyboards ebenfalls automatisch angelegt



DEMO: MyQuiz

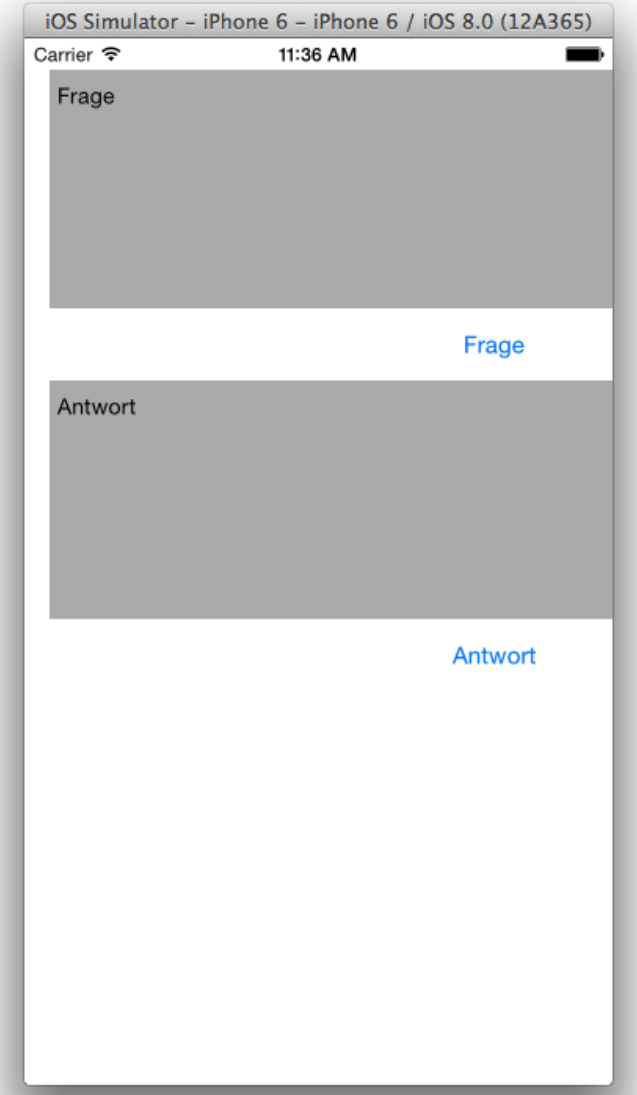
Editieren der View

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



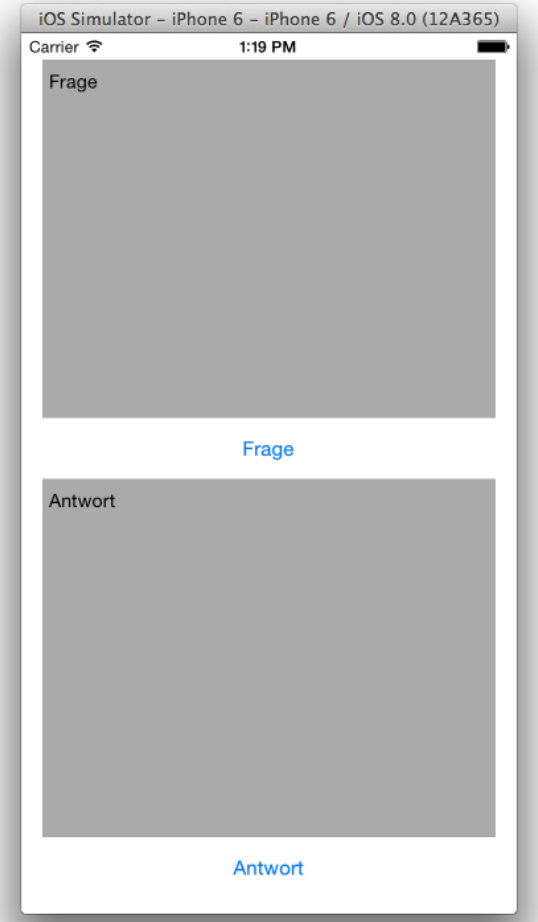
DEMO: MyQuiz

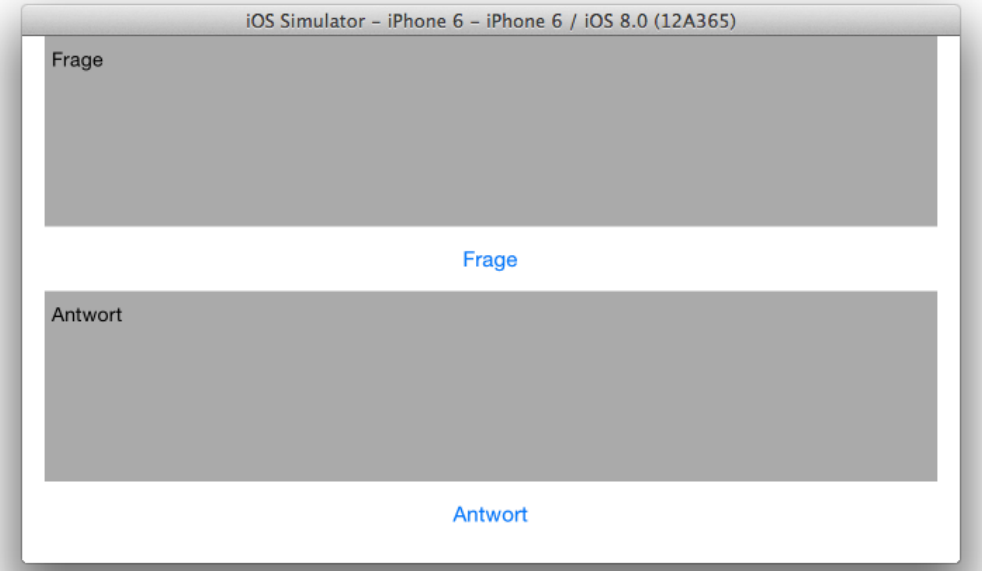
- Ausführen bringt bisher nicht das gewünschte Ergebnis...
- Problem: Im Storyboard werden durch das Platzieren der UI-Elemente nicht-adaptive Layout-Constraints vergeben



DEMO: MyQuiz

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





DEMO: MyQuiz

Hinzufügen eines Model

- File → New → File → Cocoa Touch Class
- QuestionPool: dient der Erzeugung und dem Zugriff auf Fragen und Antworten

DEMO: MyQuiz

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
```

DEMO: MyQuiz

1. Weg: Implementierung unter direkter Verwendung von Instanzvariablen

```
#import "QuestionPool.h"

@implementation QuestionPool

- (id)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*)quesions {
    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

-(id)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“,
                   @“1,5 Millionen“,
                   @“98,67 Meter“];
   }
   return self;

@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 `typdef` 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
[…]
```

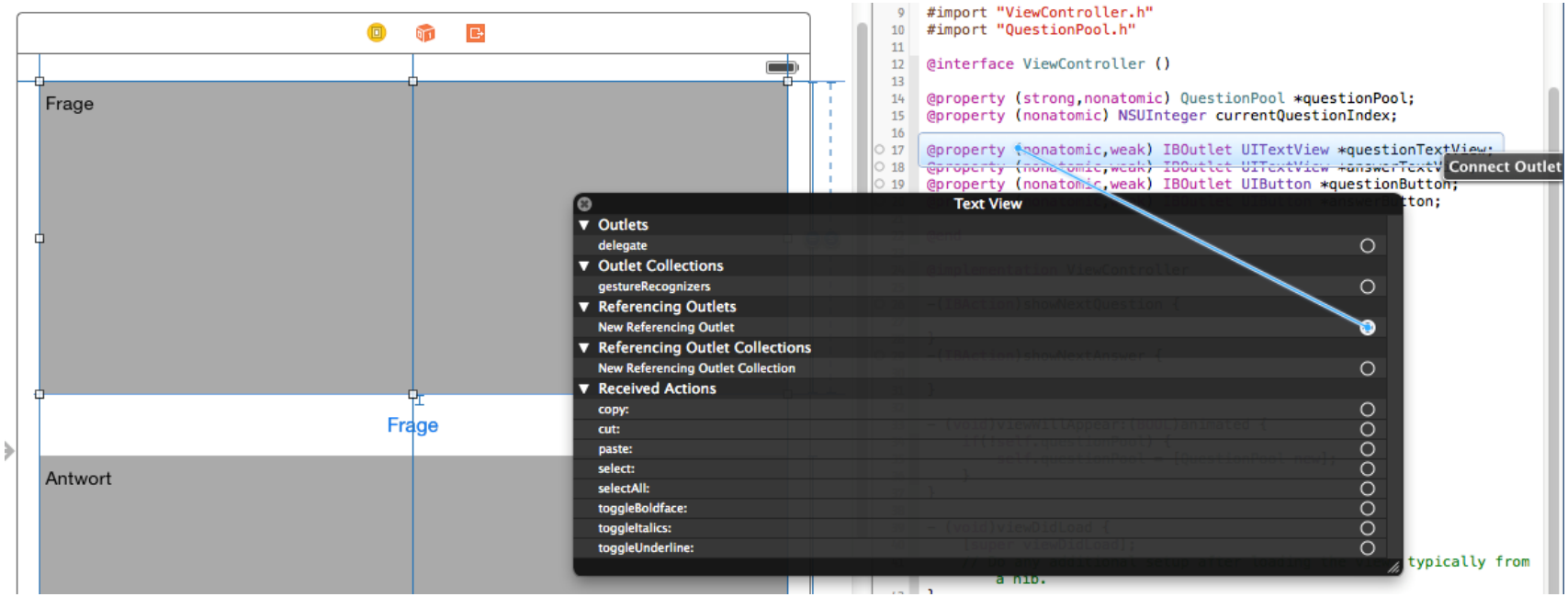
Deklaration der Methoden (Actions)

```
//  ViewController.m

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

@interface ViewController ()
[…]
@end

@implementation ViewController

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

@end
```

DEMO: MyQuiz

Verbinden von Controller und View (Outlets und Actions)

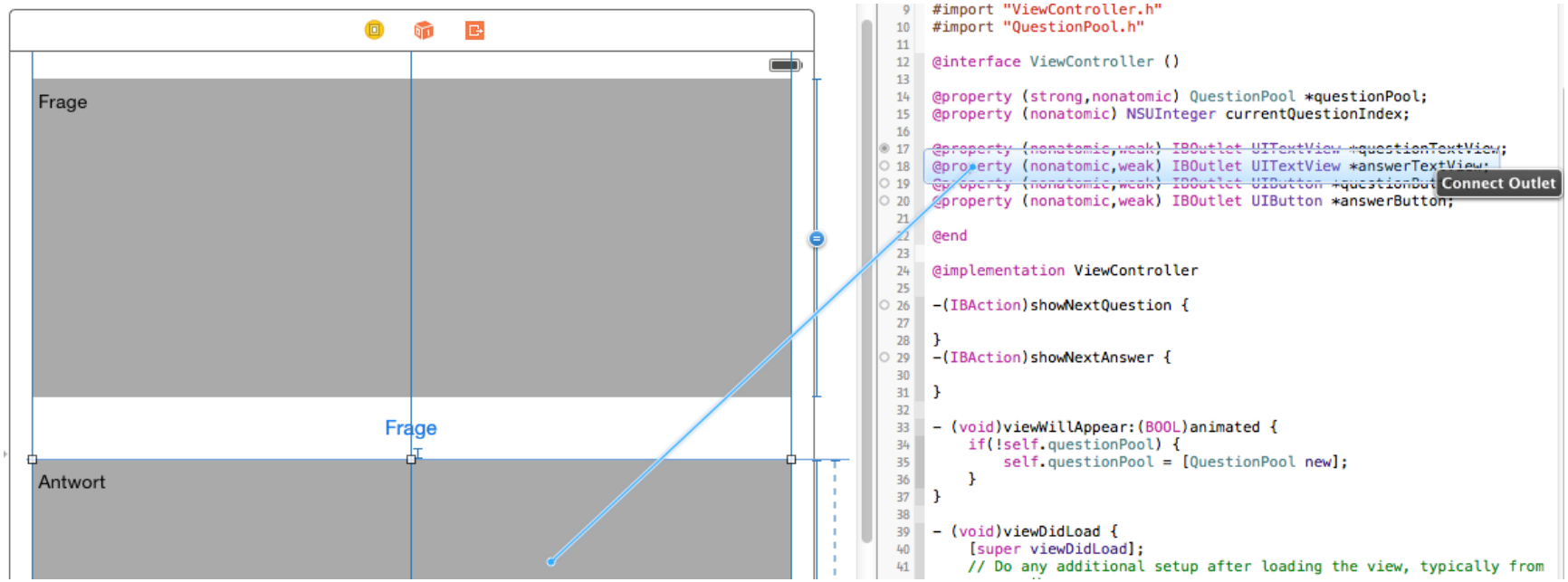
- Mehrere Methoden möglich



DEMO: MyQuiz

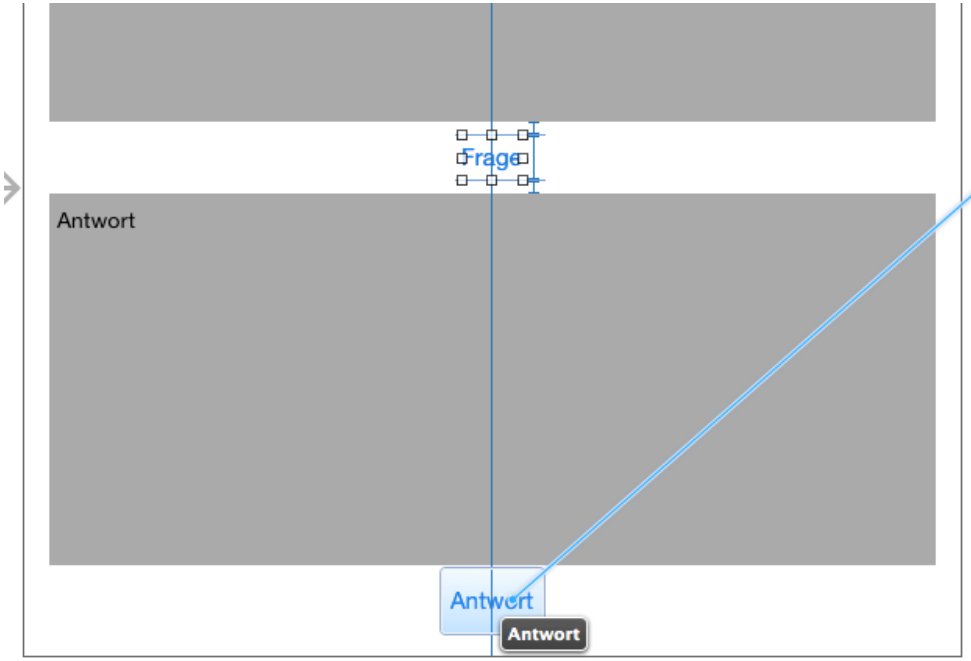
Verbinden von Controller und View (Outlets und Actions)

- Mehrere Methoden möglich



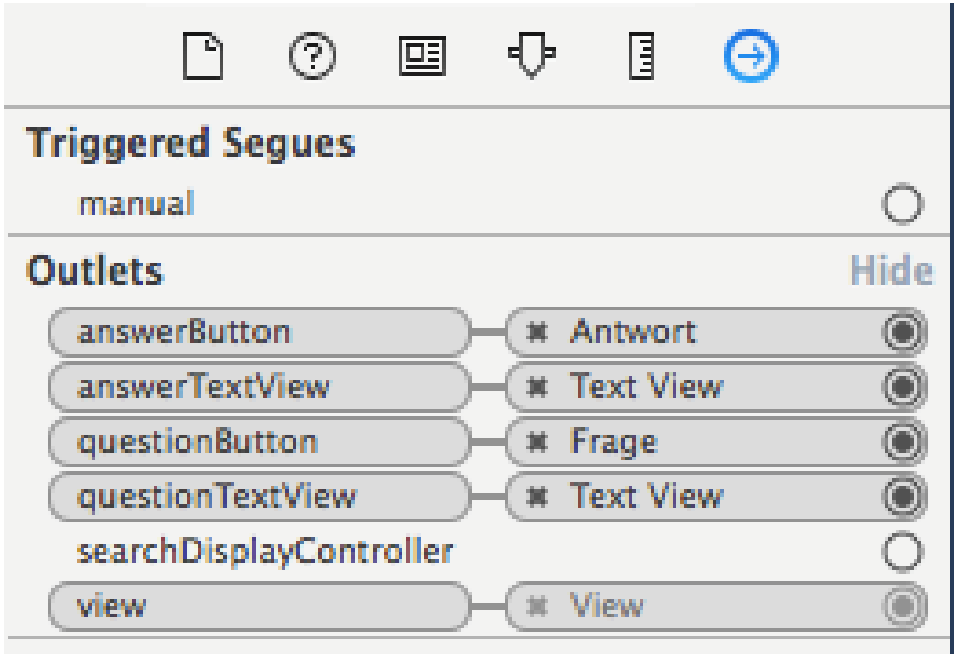
Verbinden von Controller und View (Outlets und Actions)

- 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 f
           a nib.
42 }
43
44 - (void)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
```

DEMO: MyQuiz

Problem:

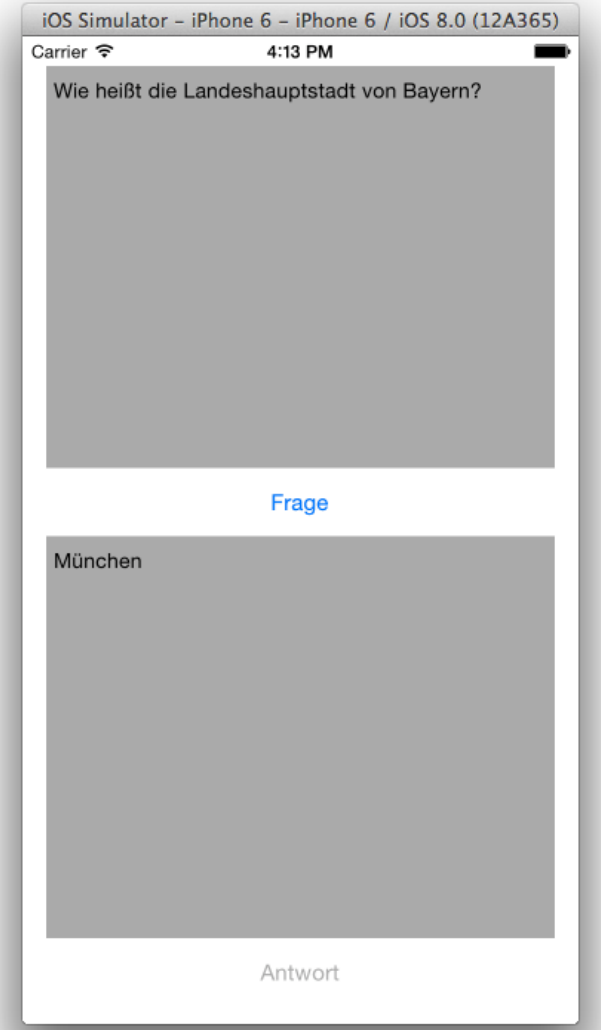
- Man kann Antwort klicken, bevor die Frage gestellt wurde

Lösung:

```
//  ViewController.m
[…]

@implementation ViewController

[…]

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

[…]

@end
```

DEMO: MyQuiz

Ergebnis:



Anwendungen ohne Storyboard

Storyboards sind schön, aber...

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

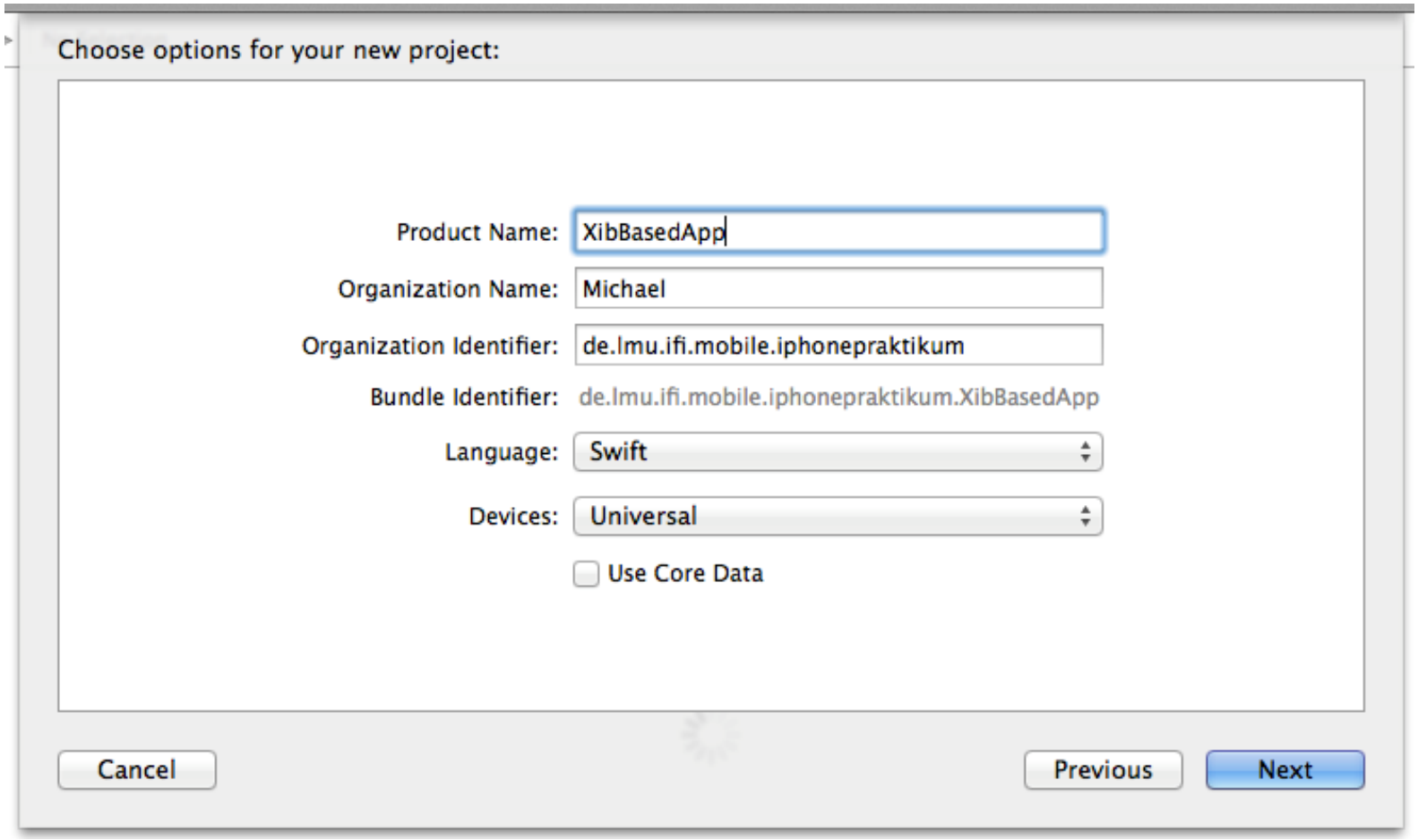
Alternativ kann man die UI mit Hilfe von XIB-Dateien implementieren

- Pro View eine XIB-Datei

Anwendung ohne Storyboard Beispiel

Erzeugen eines neuen Projekts:

- File → New → Project → Single View Application



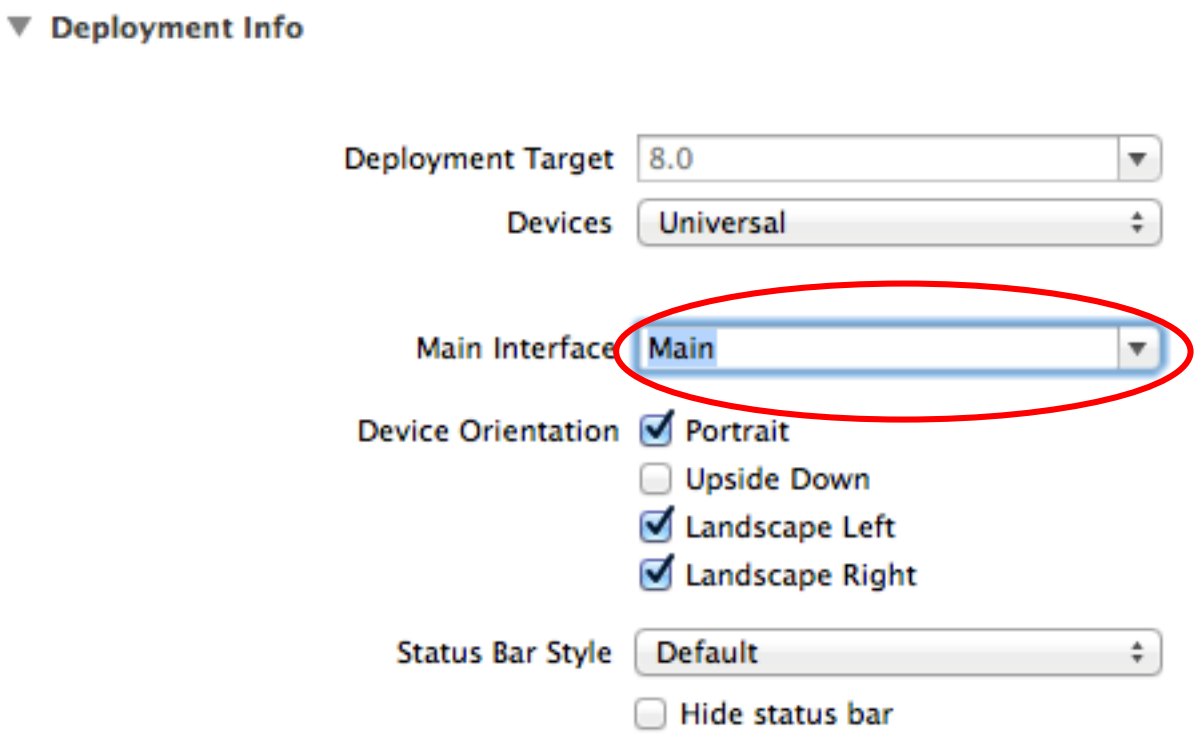
Anwendung ohne Storyboard Beispiel

Project Navigator

- Löschen der Datei "Main.storyboard"

Project Navigator → General → Deployment Info

- Löschen des Inhalts von "Main Interface"



Anwendung ohne Storyboard Beispiel

Project Navigator

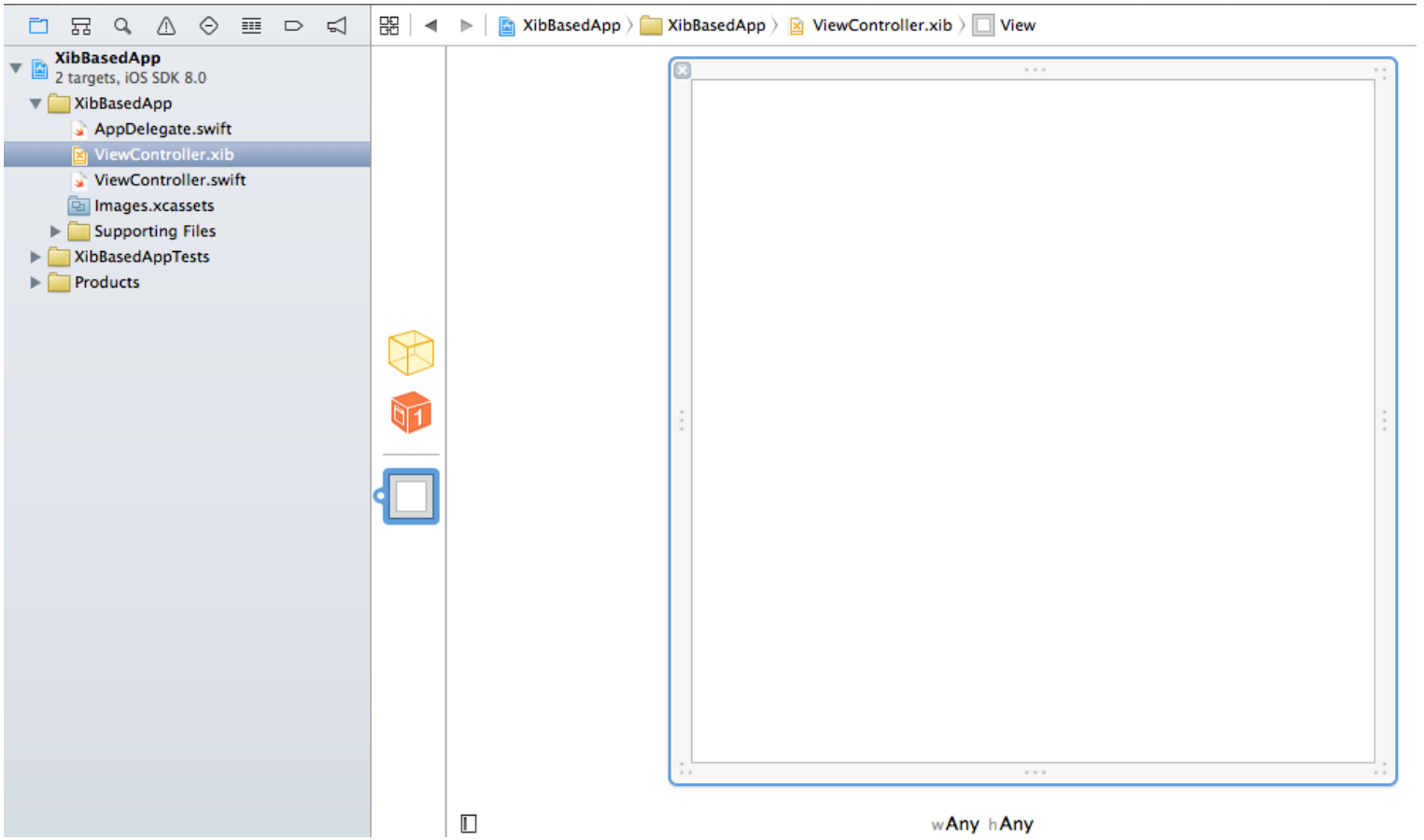
- Anpassen der Datei "AppDelegate"

```
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) {}
}
```

Anwendung ohne Storyboard Beispiel

Erzeugen einer XIB-Datei

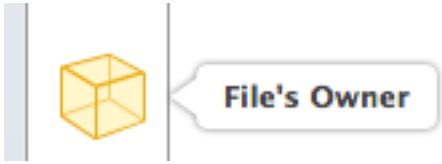
- File → New → File → User Interface → View
- Name "ViewController"



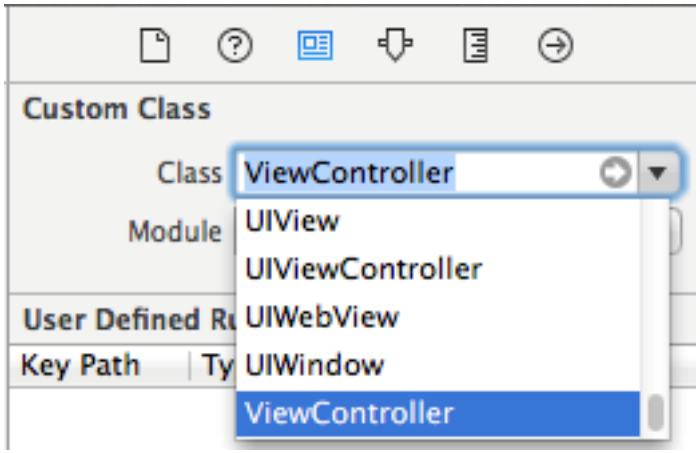
Anwendung ohne Storyboard Beispiel

Anpassen der XIB-Datei

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

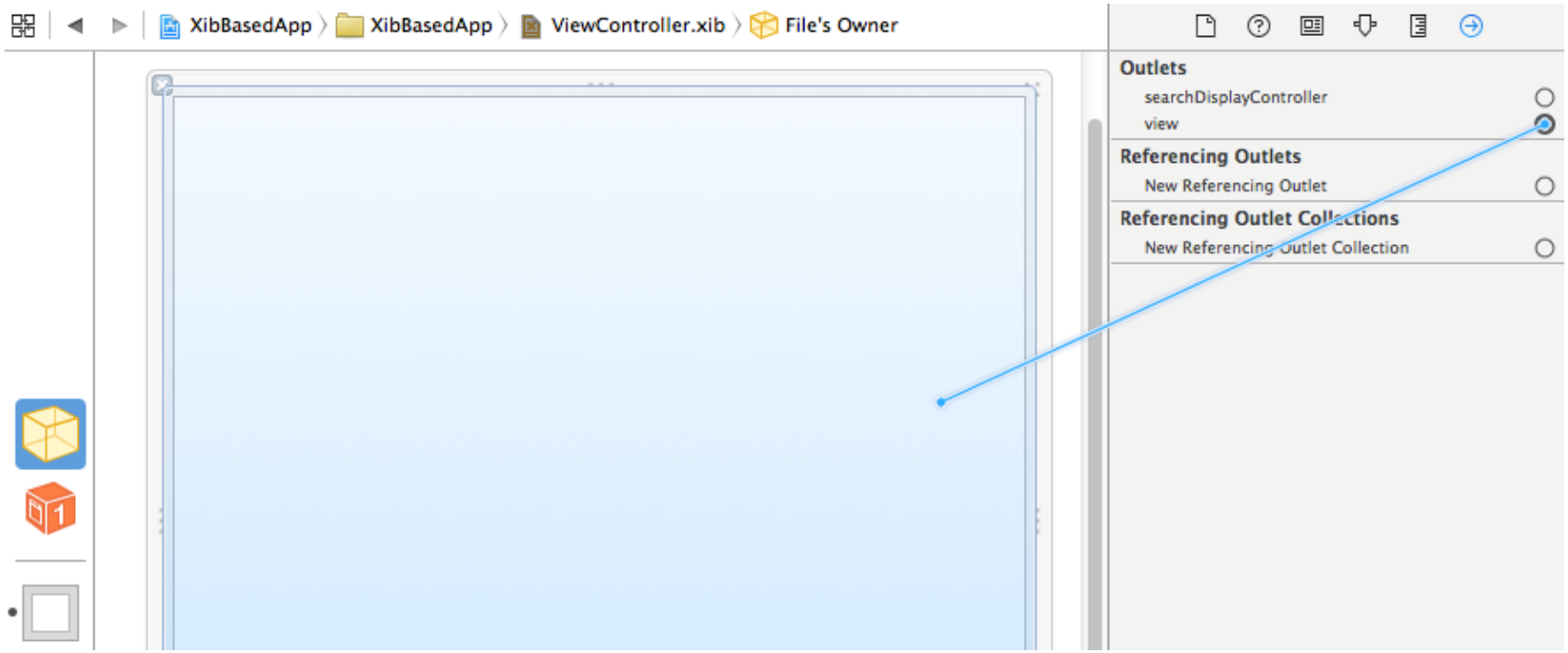


- Im "Identity Inspector"
  - Setzen der "Custom Class" auf "ViewController"

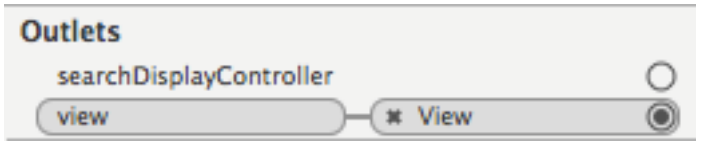


Anpassen der XIB-Datei

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

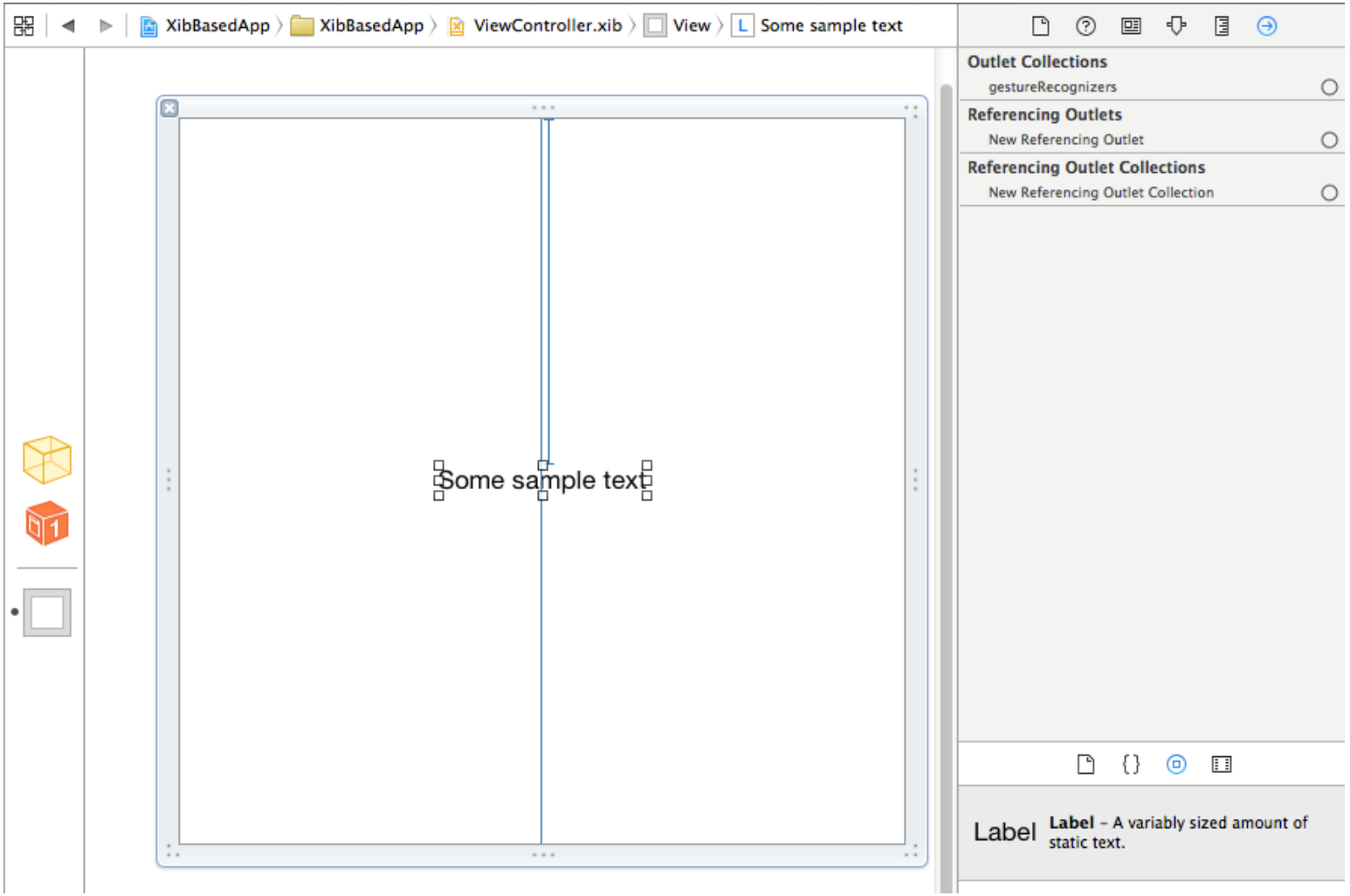


- Ergebnis:



Anwendung ohne Storyboard Beispiel

Hinzufügen von UI Komponenten wie gehabt...



Anwendung ohne Storyboard Beispiel

Ergebnis: