1 Mehrere ViewController

Wenn man mehrere View-Controller verwenden will, so kann man aus mehreren Varianten wählen. Für alle gilt aber, dass die Verknüpfung per Ctrl-Taste und Drag&Drop bestimmt wird. Damit sind diese nicht so leicht einsehbar. Prinzipiell kann man die Verknüpfung auch per Code-Behind eintragen.

1.1 Tab-Bar-Controller

Der Tab-Bar-Controller ist die einfachste Version. Es wird ein Projekt „Tabbed-Application“ mit zwei eingebauten View-Controllern gewählt.

Neues Projekt:

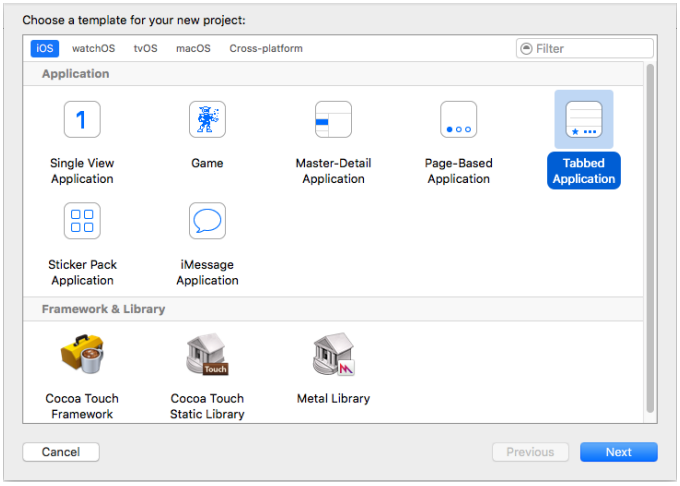


Abbildung 1 Projekt Tabbed-Application

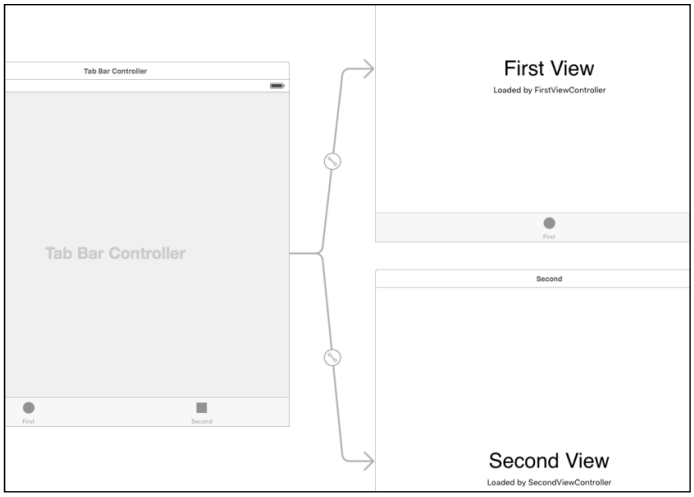


Abbildung 2 Aufbau eines Tabbed-Application

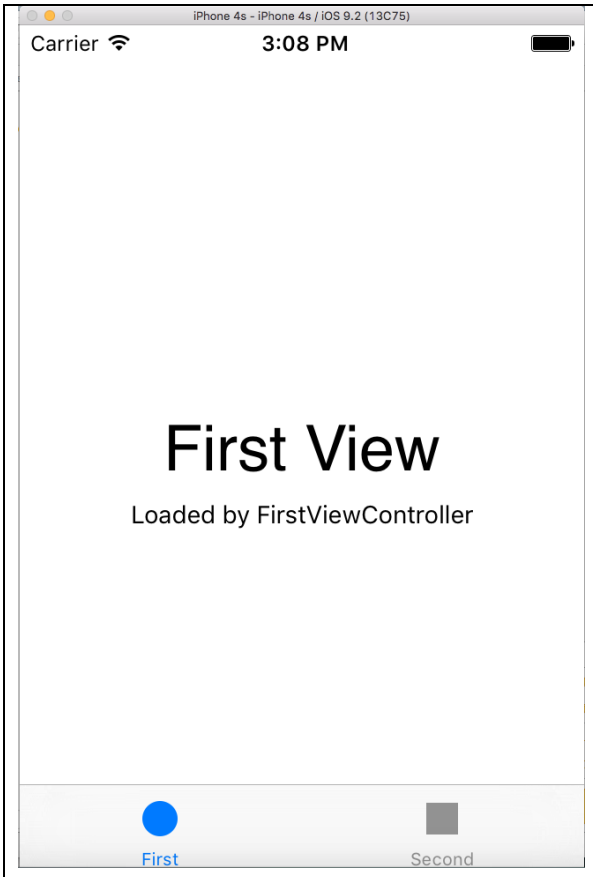


Abbildung 3 Tab-Bar-Seite 1

Abbildung 4 Tab-Bar-Seite 2

Die beiden obigen Abbildungen zeigen den Original-Zustand nach dem das Projekt erstellt wurde.

Im Projekt sind nun zwei ViewController und ein Tab-Bar-Controller. Die beiden ViewController haben die beiden Controller-Dateien „FirstViewController.swift“ und „SecondViewController.swift“.

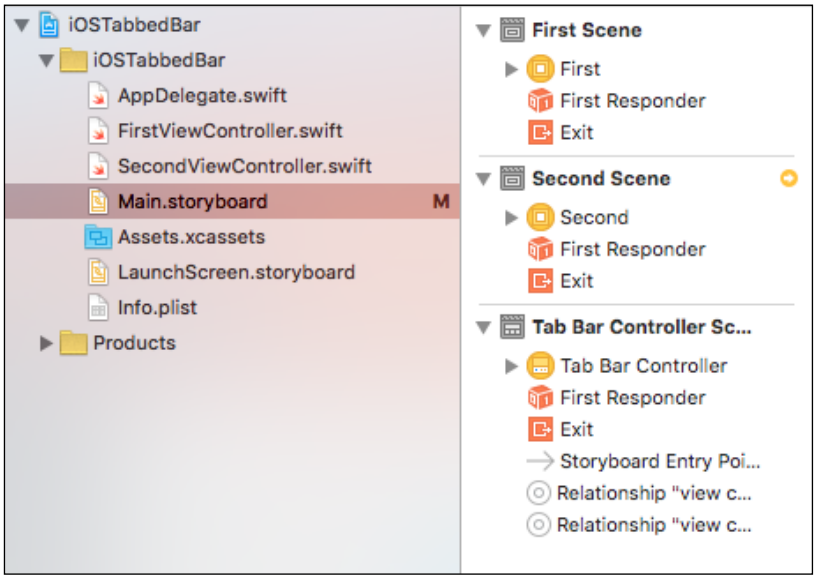


Abbildung 5 Anzeige der beiden ViewController und des TabBar-Controller im Projekt

Das Problem ist nun, wie bekommt man weitere ViewController ins Projekt bzw. in dem Tab-Bar-Controller.

1.1.1 Neuer uiViewController

1. Aus der ui-Galerie wird ein uiViewController in den Mainboard.story geschoben. Er wird nicht in den Tab-Bar-Controller geschoben, sondern als „Nachbar“ platziert.

2. Aktivieren Sie den Tab-Bar-Controller. Drücken Sie die Ctrl-Taste und ziehen Sie nun die linke Maustaste auf den neuen ViewController. Damit ist die Verbindung erstellt. Beachten Sie dabei, dass der Zoomfaktor auf 100% steht.

Abbildung 6 Verknüpfung des neuen ViewControllers mit dem TabBar -Controller

3. Wählen Sie beim Dialog den Eintrag „view-controllers“ aus.

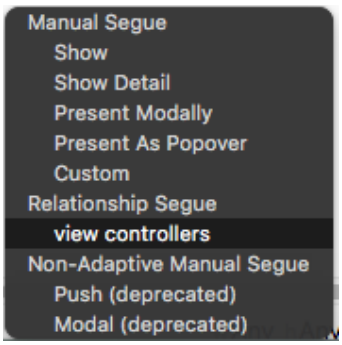


Abbildung 7 Neuer View mit einem Tabbar verbinden

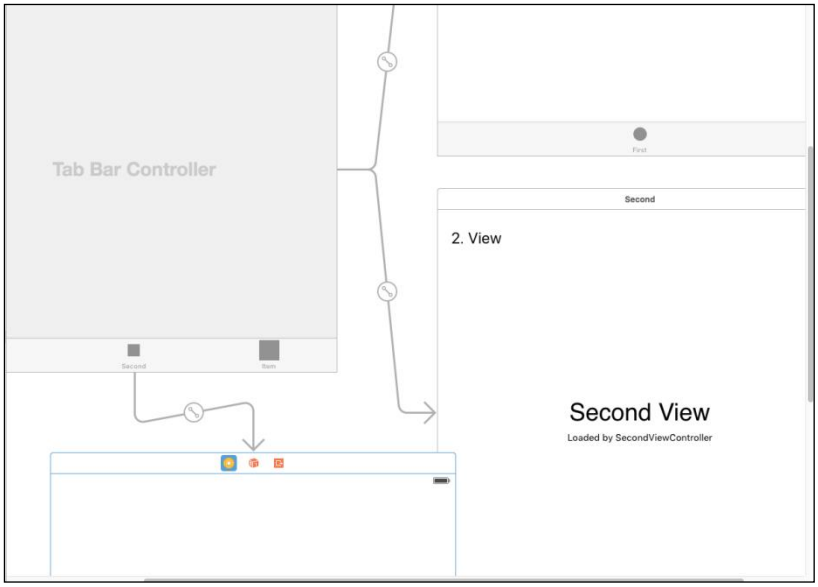


Abbildung 8 Anzeige der Verknüpfungen eines Tab-Bars

4. In den Eigenschaften können Sie nun wählen, ob Sie ein vorhandenen Symbol oder ein eigenes Symbol verwenden.

Abbildung 9 Vorgegebenes Symbol für einen Tab-Bar-Item auswählen

- Im oberen Abschnitt können Sie vorgefertigte Symbole auswählen.
- Für untere Variante müssen Sie ein Symbol im Ordner „Images.xcassets“ eintragen und registrieren. Leider ist ein einfaches Kopieren nicht möglich.

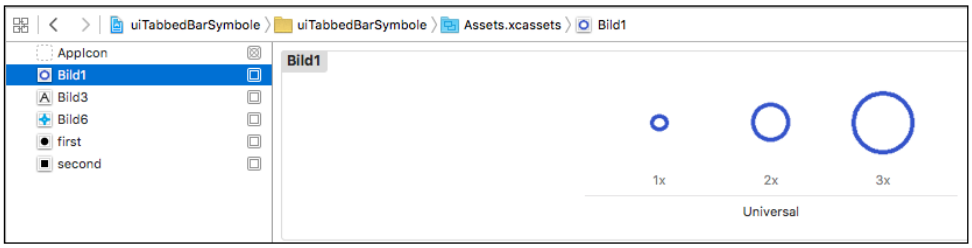


Abbildung 10 Symbole für die Views einer Tab-Bar-Application

- Öffnen Sie das Fenster für den Eintrag „Assets.xcassets“ bzw. Images.xcassets..
- Öffnen Sie den Finder und den Ordner, in dem die Bilder eingetragen sind.

    - **Die Bilder müssen einfarbig sein.**
    - **Die Farbe der Symbole spielt keine Rolle, es wird immer die blaue Farbe verwendet.**
    - **Der Hintergrund muss transparent sein.**
    - **Es sollten drei Bilder vorhanden sein:**
        - 25×25 Pixel
        - 50×50 Pixel
        - 75×75 Pixel
- Ziehen Sie das **erste** Bild in den Assets.xcassets bzw. Images.xcassets.. Der linke Bereich in der oberen Abbildung.
- Ziehen Sie die beiden restlichen Bilder in den Bereichen 2x und 3x.
- Alternativ können Sie auch eine PDF-Datei mit einer SVG-Grafik einfügen. Diese wird dann skaliert.
- Damit können im Eigenschaftsdialog das/die Bilder auswählen. Beachten Sie, dass man die **untere** Eingabe verwendet.

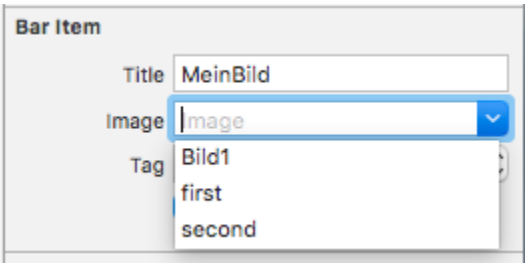


Abbildung 11 Auswahl eines eigenen Bildes im neuen ViewController

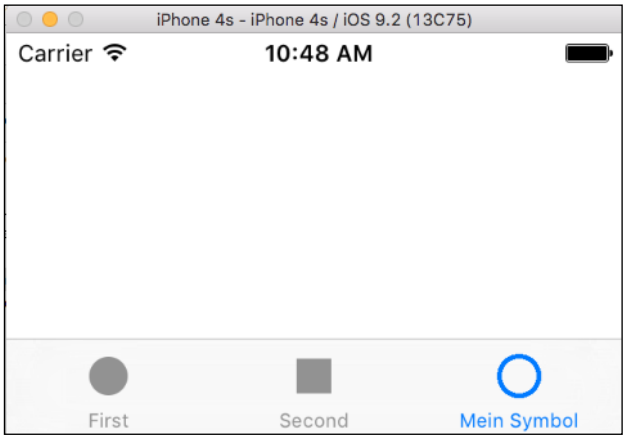


Abbildung 12 Anzeige des Symbol im neuen ViewController, teilweise geschrumpft

Im Projekt-Baum sieht man nun die drei ViewController:

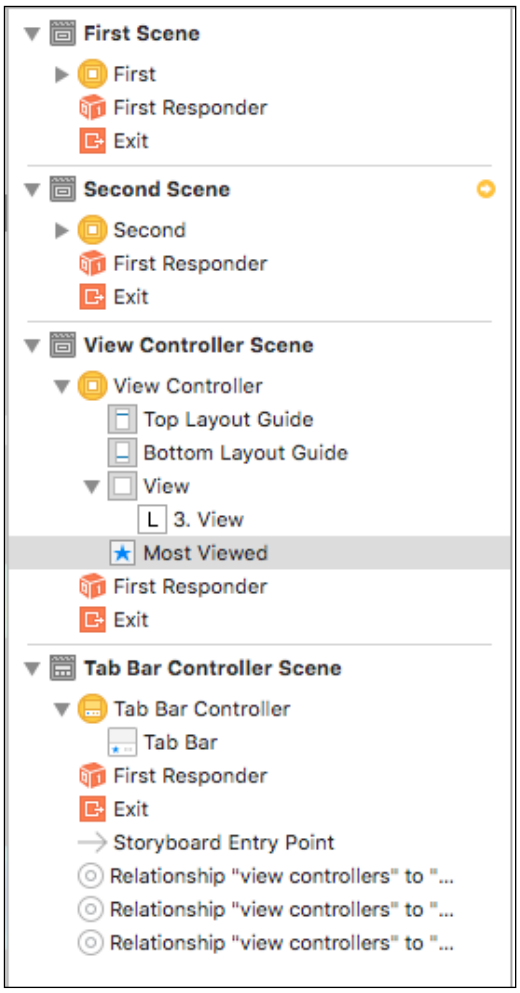
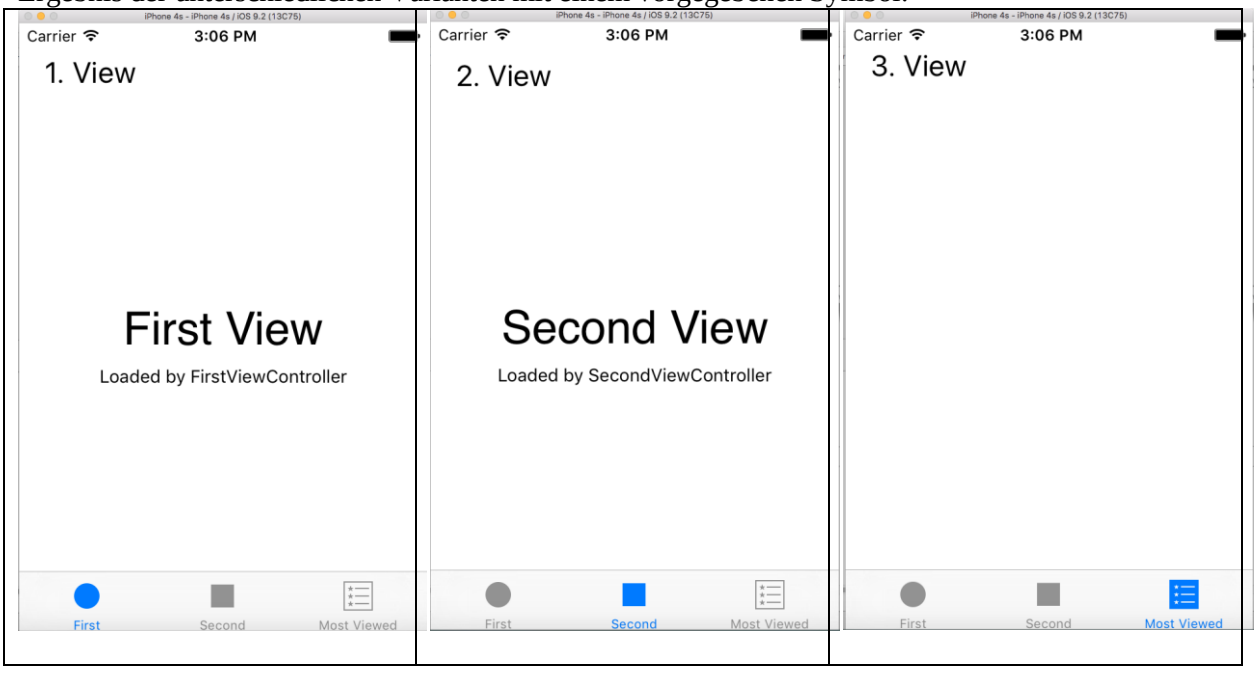


Abbildung 13 Drei ViewController im Projekt-Baum

Ergebnis der unterschiedlichen Varianten mit einem vorgegebenen Symbol:



Man kann ab Version 8.x nun fast beliebig viele ViewController in das Hauptfenster einfügen.

Abbildung 14 Tabbed-Bar mit zehn ViewController

1.2 Segues

Bei der Verwendung der „Segues“-Technik, Nachfolger, muss man die Verwaltung teilweise selbst vornehmen. Auch die „Zurück-Technik“ ist aufwändiger.

1. Gestartet wird mit einer Single-View-Application.

2. Nun wird ein neuer ViewController per Drag&Drop hinzugefügt. Beide liegen nun nebeneinander.

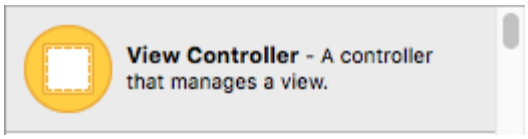
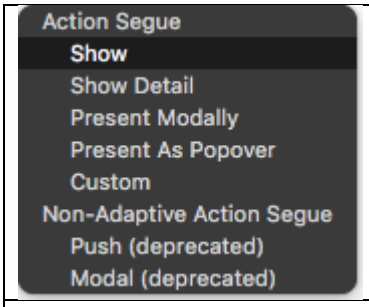


Abbildung 15 ViewController hinzufügen

3. Nun wird auf jedem ViewController ein Button eingefügt. Die Verknüpfung erfolgt aber **nicht** mit Quellcode, kann aber.
4. Nun wird der Button des ersten ViewControllers angeklickt und bei gedrückter Ctrl-Taste die linke Maustaste per Drag&Drop auf den zweiten ViewController gezogen. Xcode zeigt nun einen kleinen Dialog, siehe unten, in dem man die gewünschte Segue-Action auswählt.



Ausgewählt wird der Eintrag „show“

Abbildung 16 Einen Button mit einem ViewController automatisch verbinden

Typen der Segue-Action:

- Show
  - iOS verwaltet den kompletten View-Wechsel (**Beste Wahl**).
- Show-Detail
  - Show-Detail ändert die Details-Ansicht im zweiten Controller in einem Split-Controller.
- Present Modality
  - Ist der Show-Action ähnlich, nur hier kann man die Animation zum zweiten View beeinflussen. **Der ViewController ist modal**.
- Present as PopOver
  - Anzeige des Views als „Popup-Menü“. Die aktuelle Ansicht wird nur teilweise verdeckt. Bei iPads einfach. Bei iPhones benötigt man etwas Code.
  - Diese Technik hat nicht funktioniert.
- Custom
  - Hier kann man Übergänge und Effekte selber definieren.

Nun kann man die beiden Views testen:

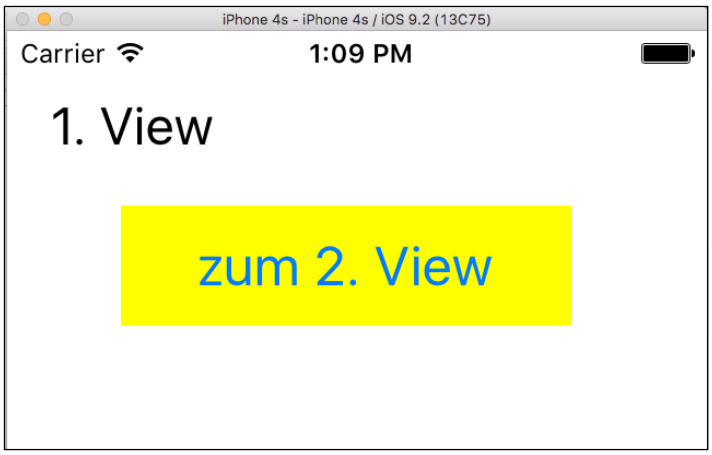


Abbildung 17 Erster ViewController

Abbildung 18 Zweiter ViewController. Der Zurcük-Schalter funktioniert aber noch nicht.

Es fehlen noch:

- Back to the roots (Rücksprung).
- Der ViewController.swift, der Texteditor.

1.2.1 Back to the roots

Das „Zurück-Springen“ ist komplizierter, da man nicht die erste Technik verwenden darf. Damit würde man jedesmal einen neuen ViewController erzeugen. Das Ziel ist es, den zweiten zu beenden.

Ablauf:

1. Einfügen einer Quellcode-Datei.
2. Eintragen einer Delegate-Funktion.
3. Verbinden des „Zurück“-Schalters zum Exit-Schalter.

1.2.2 Einbau eines neuen ViewController.swift, Quellcode-Datei

1. Aufruf des Einfüge-Dialogs mit Cmd+N.
2. Auswahl des Symbols „Cocoa Touch Class“.

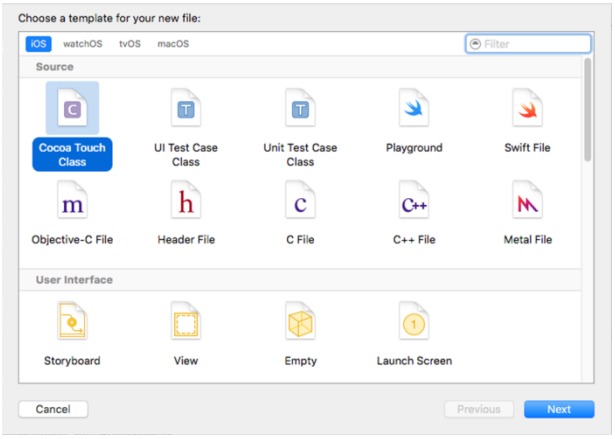


Abbildung 19 Neuer Eintrag "Cocoa Touch Class"

3. Speichern der Klasse

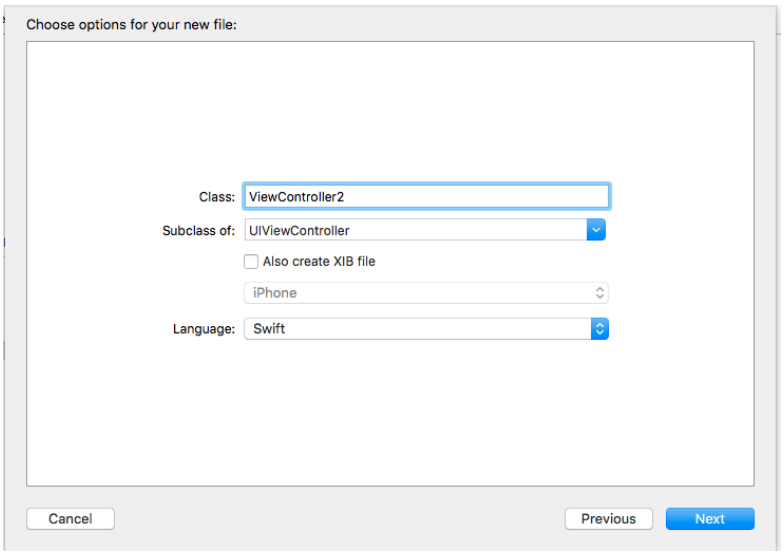


Abbildung 20 Speichern der neuen Klasse

Anzeige der neuen Klasse im Projektbaum

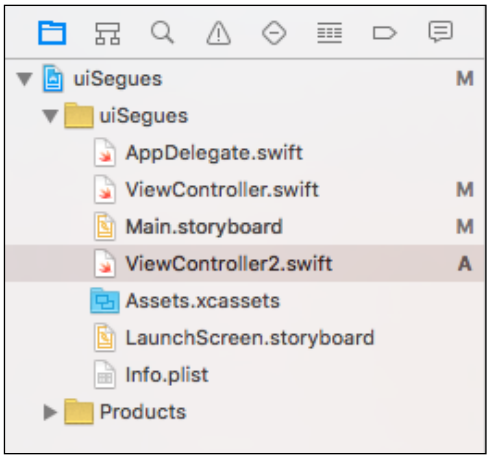


Abbildung 21 Anzeige der neuen Klasse im Projektbaum

1.2.3 Eintragen einer Delegate-Funktion

1. Eintragen einer Delegate-Funktion im neuen Quellcode.

```
@IBAction override func unwindForSegue(
  unwindSegue: UIStoryboardSegue,
  towardsViewController subsequentVC: UIViewController) {
    //  hier fehlt kein Quellcode
}
```

1.2.4 Verknüpfen des Schalters mit dem Exit-Button

Nachdem im Quellcode die Methode „unwindForSegue“ eingetragen wurde, funktioniert auch die Verknüpfung mit dem „Exit“-Schalter.

- Bei gedrückter Ctrl-Taste zieht man die linke Maus auf den Exit-Schalter im View des Projekts.
- Nun erscheint ein kleiner Dialog, in dem man die oben eingefügt Methode auswählt.

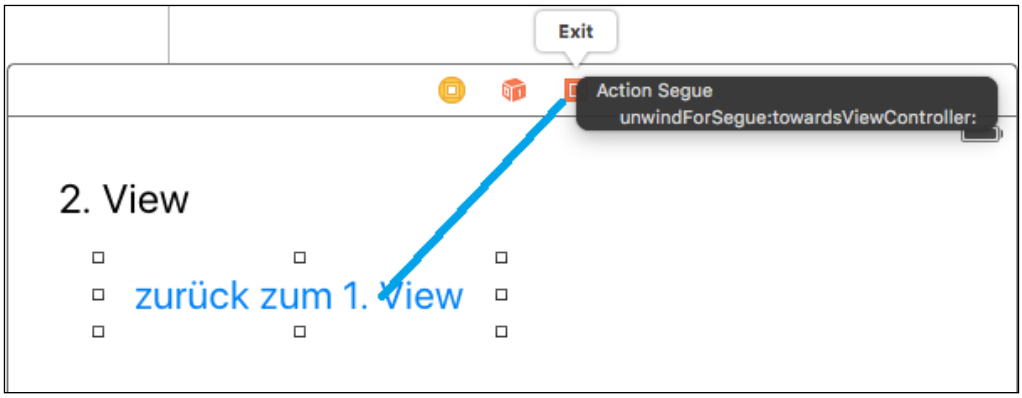


Abbildung 22 unWind-Aktion in einem Segue-View einbauen

1.3 Navigations-Controller

Der Navigations-Controller fungiert im Prinzip als Hauptmenü. Mit seiner Hilfe kann man weitere Views aufrufen.

Ablauf:

1. Erstellen eines Projektes „Single View Application“
2. Einfügen eines Navigations-Controllers.
3. Nun muss man den Navigations-Controllers als StartView definieren. Dazu öffnet man die Eigenschaften des Navi-Controllers und setzen die Eigenschaft „Is Initial View Controller“ (ganz unten im Bild).

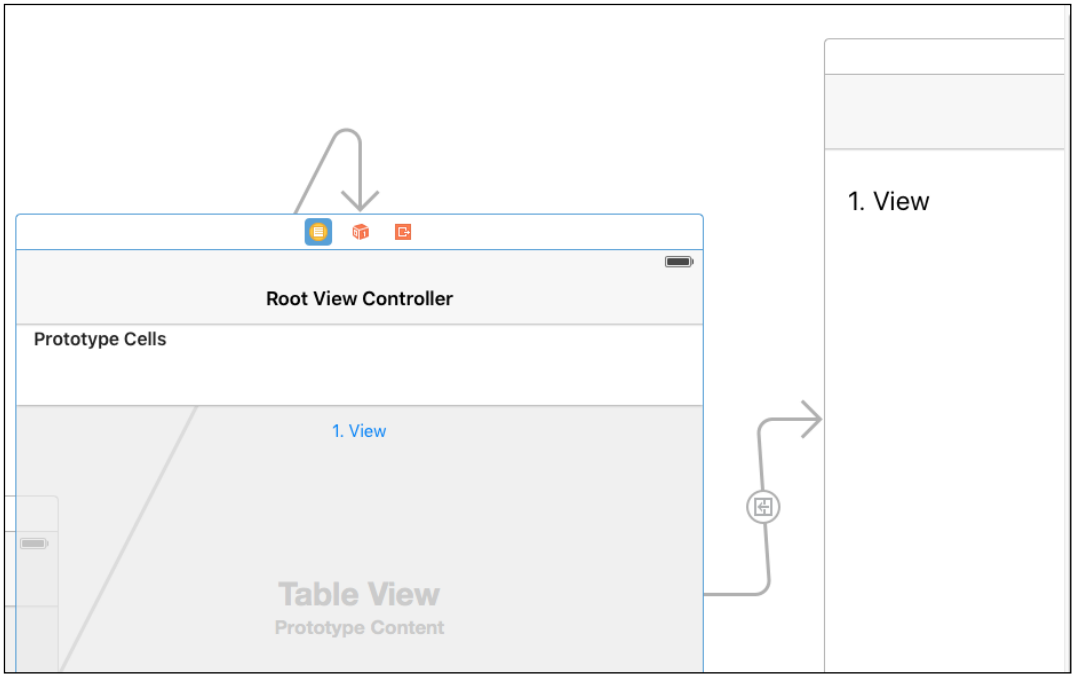


Abbildung 23 Struktur des Projektes "Navigations-Controller“

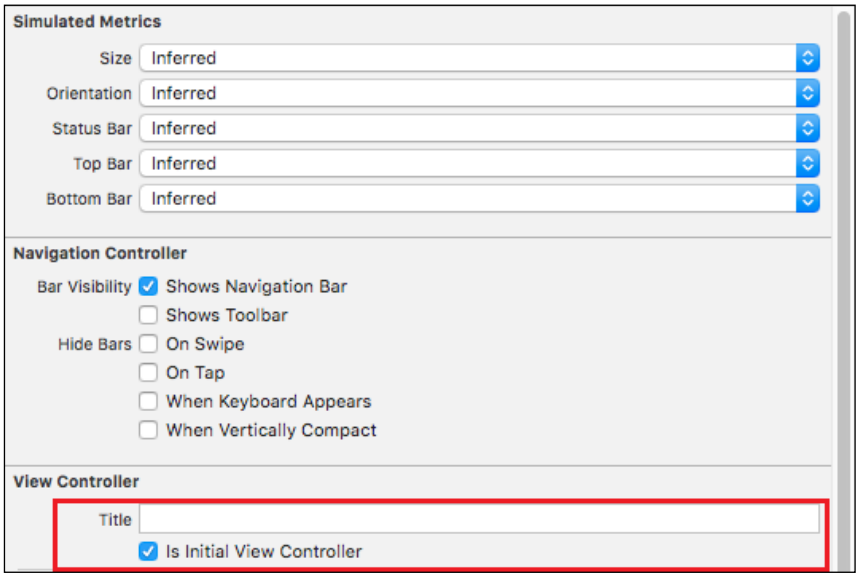


Abbildung 24 Setzen eines Navigations-Controller "Is Initial View Controller"

4. Nun zieht man einen Schalter auf den Navi-Controller.

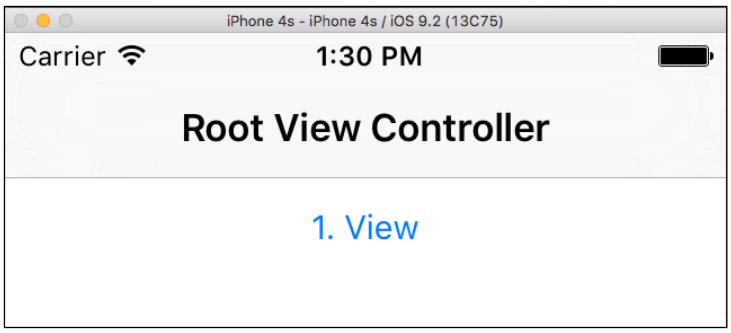


Abbildung 25 Schalter einfügen in einen Navigations-Controller

Test:

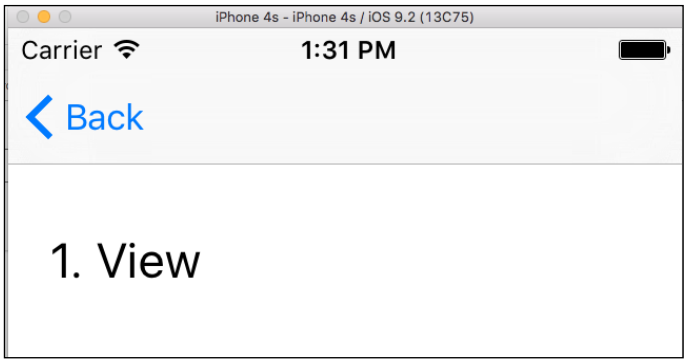


Abbildung 26 Anzeige "back 2 the root"

Der Vorteil dieser Variante ist der wesentlich geringere Aufwand.

1.4 TableView mit eigenen Navigationsschaltern

Der erste Teil ist etwas kürzer gehalten, da die verwendete Technik weitgehend bekannt sein sollte.

1. Vorlage „Single View Application“
2. Klasse Stadt anlegen

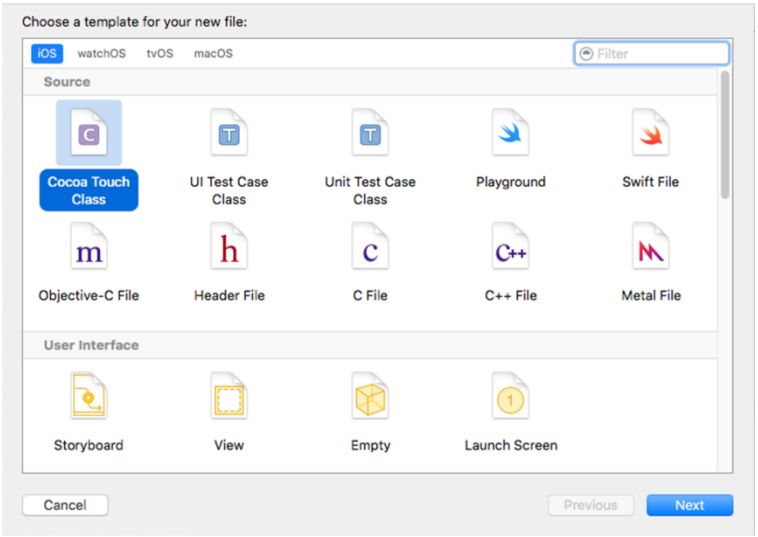
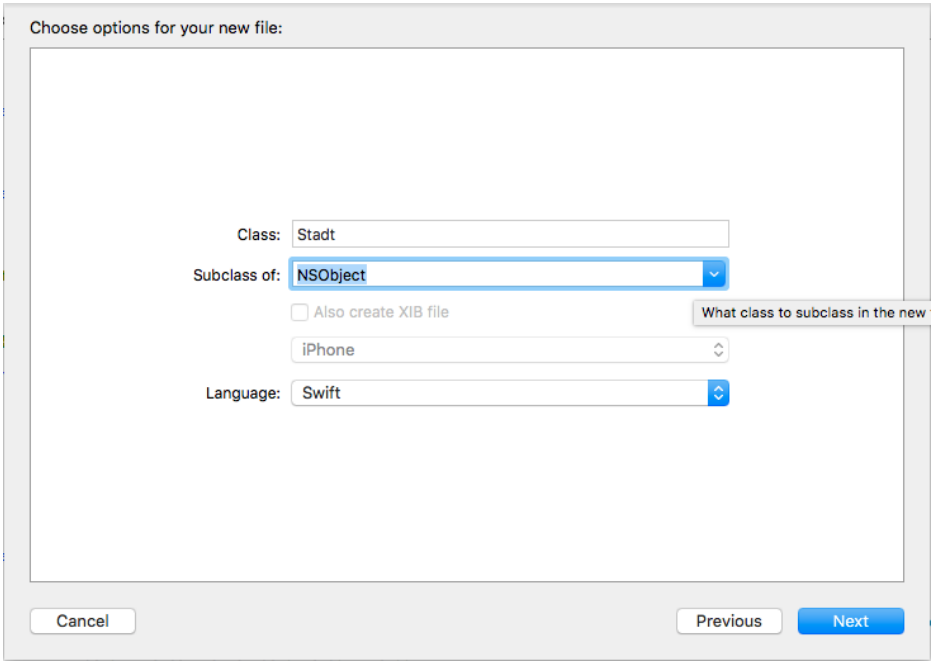


Abbildung 27 Neuer Eintrag "Cocoa Touch Class"

Basis-Klasse auswählen: NSObject



Speichern unter dem Klassennamen, z. B. Stadt

Eintragen der Klassenstruktur:

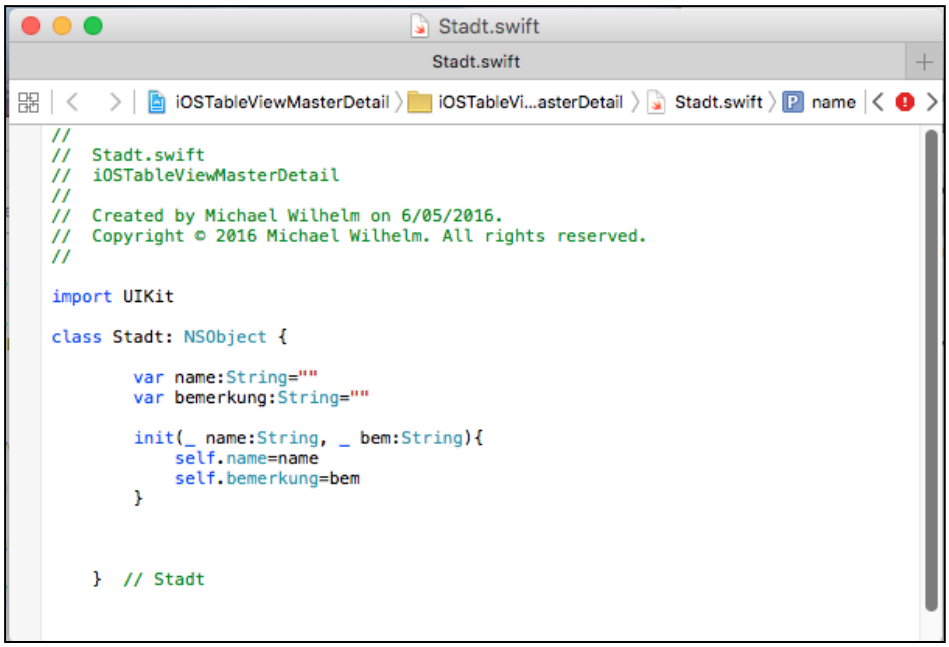
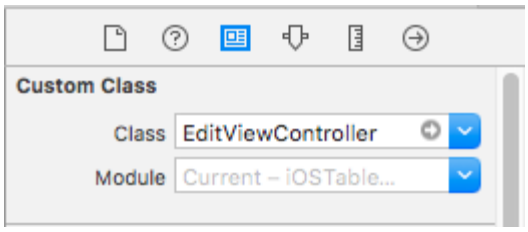


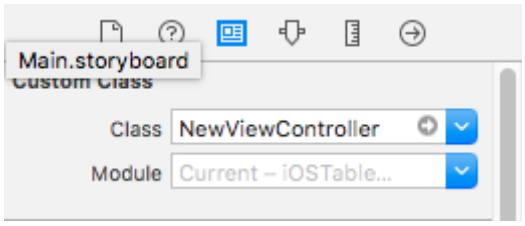
Abbildung 28 Projekt-Klasse „Stadt“

Damit kann man die obige Klasse im gesamten Projekt benutzen. Im nächsten Schritt wird der Master-View so verändert, dass er mehrere Städte in einer Tabelle anzeigen kann.

3. ein uiLabel-Element in den ViewController einfügen.
4. uiButton neben dem Label einfügen.
    a. Text: Edit
5. uiButton neben dem Label einfügen.
    a. Text: New
6. uiButton neben dem Label einfügen.
    a. Text: Remove
7. ein uiTableView-Element in den ViewController einfügen.
8. Die Constraints bearbeiten.
9. Einfügen eines neuen ViewControllers.
    a. ein uiLabel-Element in den neuen ViewController einfügen.
        i. Inhalt: EditViewController
    b. ein uiLabel-Element in den neuen ViewController einfügen.
        i. Inhalt: Name
    c. ein uiTextfield-Element in den neuen ViewController einfügen.
        i. Name: tName als Outlet
    d. ein uiLabel-Element in den neuen ViewController einfügen.
        i. Inhalt: Bemerkung
    e. ein uiTextView-Element in den neuen ViewController einfügen.
    f. Name: tBemerkung als Outlet
    g. uiButton für dem Ok-Schalter
    h. uiButton für dem Abbruch-Schalter
10. Constraints für den edit-ViewController einbauen (optional).
11. Einfügen einer neuen Quellcode-Datei für den ViewController.
    a. Aufruf des Einfüge-Dialogs mit Cmd+N.
    b. Auswahl des Symbols „Cocoa Touch Class“.
    c. Class „EditViewController“
    d. SubClass: „UIViewController“
    e. Im Property Inspector des EditViewController die Quellcode eintragen.



12. Einfügen eines neuen ViewControllers (New)
    a. ein uiLabel-Element in den neuen ViewController einfügen.
        i. Inhalt: NewViewController
    b. ein uiLabel-Element in den neuen ViewController einfügen.
        i. Inhalt: Name
    c. ein uiTextfield-Element in den neuen ViewController einfügen.
        i. Name: tName als Outlet
    d. ein uiLabel-Element in den neuen ViewController einfügen.
        i. Inhalt: Bemerkung
    e. ein uiTextView-Element in den neuen ViewController einfügen.
    f. Name: tBemerkung als Outlet
    g. uiButton für dem Ok-Schalter
13. Constraints für den new-ViewController einbauen (optional).
14. Einfügen einer neuen Quellcode-Datei für den ViewController.
    a. Aufruf des Einfüge-Dialogs mit Cmd+N.
    b. Auswahl des Symbols „Cocoa Touch Class“.
    c. Class „NewViewController“
    d. SubClass: „UIViewController“
    e. Im Property Inspector des NewViewController die Quellcode eintragen.



15. Eintragen der unwind-Methode in beide neuen ViewController:

```
@IBAction override func unwindForSegue(
        unwindSegue: UIStoryboardSegue,
        towardsViewController subsequentVC: UIViewController) {
         // hier fehlt Code
    }
```

16. Eintragen der unwind-Methode in beide neuen ViewController:
17. Nun per Drag&Drop die beiden Verbindungen von mainController zu den beiden neuen Controllern verbinden.
    a. Schalter „Edit“ anklicken
    b. Ctrl-Taste drücken
    c. Mit der linken Maustaste zum EditViewController ziehen
18. Quellcode des mainControllers:
    a. Delegate eintragen: UITableViewDelegate, UITableViewDataSource
    b. staedte-Array deklarieren und erzeugen.
    c. Outlet erzeugen

i. tableview
ii. Schalter edit
iii. Schalter new

d. init-Methode setzen (staedte-Array und delegate-self)
e. Setzen der tableView-Delegate-Methoden.

```
import UIKit
```

MainController:

```
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {

    var staedte = [Stadt]()
    var neueStadtNr=0

    @IBOutlet var tableview: UITableView!

    @IBOutlet var bnEdit: UIButton!

    @IBOutlet var bnNew: UIButton!


    override func viewDidLoad() {
        super.viewDidLoad()
        staedte.append( Stadt("New York", "Manhattan")  )
        staedte.append( Stadt("Mailand", "Mailander Scala")  )
        staedte.append( Stadt("Rom",  "Vatikan, Michelangelo")  )
        staedte.append( Stadt("Moskau", "Roter Platz")  )
        staedte.append( Stadt("Wernigerode", "HS Harz")  )
        staedte.append( Stadt("New York", "Manhattan")  )

        tableview.delegate=self
        tableview.dataSource=self

    }


    func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        return 1
    }

    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return staedte.count
    }

      // cell fuellen
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell  {
        let cell:UITableViewCell=UITableViewCell(
               style: UITableViewCellStyle.Subtitle,
               reuseIdentifier: "mycell")
```

```
        let stadt:Stadt = staedte[indexPath.row]  // as! Stadt
        cell.textLabel!.text = stadt.name
        cell.detailTextLabel!.text=stadt.bemerkung
//            let pictname = "Bild"+String(indexPath.row%5)
//            let image : UIImage = UIImage(named: pictname)!
//            cell.imageView!.image = image
        return cell
    }


     // wenn edit-gedrueckt, setzen der Uebergabe-Variablen
     override func prepareForSegue(segue: UIStoryboardSegue,
                sender: AnyObject?) {
        if let button:UIButton = sender as! UIButton {
           if button==bnEdit {
              if let indexPath =
                       tableview.indexPathForSelectedRow {
                    let stadt:Stadt = staedte[indexPath.row]
                    if let dest = segue.destinationViewController
                           as? EditViewController {
                        dest.stadt=stadt
                     }
                 }  // if
            }  // if
        }  // if
    }  // prepareForSegue


     // Die naechste Methode ist die "Rueckkehr-Methode".
```

```
    @IBAction func unwindToEditView(segue:UIStoryboardSegue) {
        if let src = segue.sourceViewController as?
                    EditViewController {
           if let indexPath = tableview.indexPathForSelectedRow {
             let stadt:Stadt = staedte[indexPath.row]
             stadt.name = (src.stadt?.name)!
             stadt.bemerkung = (src.stadt?.bemerkung)!

                // update der TableView
             tableview.delegate=nil
             tableview.dataSource=nil

             tableview.delegate=self
             tableview.dataSource=self

           } // if
        }
    }

}  // ViewController
```

19. Ruecksprung im EditControllers aufbauen:
    a. Eintragen der unwindforSegue

```
    @IBAction override func unwindForSegue(
           unwindSegue: UIStoryboardSegue,
           towardsViewController subsequentVC: UIViewController) {
    }
```

20. Quellcode des EditControllers (UITextViewDelegate wg. TextViewOnChange):

```
class EditViewController: UIViewController, UITextViewDelegate {

     // oeffentliche Uebergabe-Variable
    var stadt : Stadt?=nil


    @IBOutlet var tName: UITextField!
    @IBOutlet var tBemerkung: UITextView!

    override func viewDidLoad() {
        super.viewDidLoad()
        if let stadt2 = self.stadt {
            first=true
            tName.text = stadt2.name
            tBemerkung.text = stadt2.bemerkung
        }
        tBemerkung.delegate=self  // TextViewOnChange
    }


      // onChange-Event des Textfield
    @IBAction func tNameChanged(sender: UITextField) {
        if let stadt2 = self.stadt {
```

```
            stadt2.name=tName.text!
        }

    }
    // onChange-Event des TextView:

func textViewDidChange(textView: UITextView) {
    //Handle the text changes here
    if let stadt2 = self.stadt {
        stadt2.bemerkung=tBemerkung.text!
    }
}
```

Nun fehlt noch die Verbindung der Rücksprungs-Methode vom Edit-ViewController:

```
@IBAction func unwindToEditView(segue:UIStoryboardSegue) {
```

Dazu die "UnWind Segue" im **Projekt** anklicken. dann im Property Inspector die Action-Methode setzen (siehe untere Abbildung):

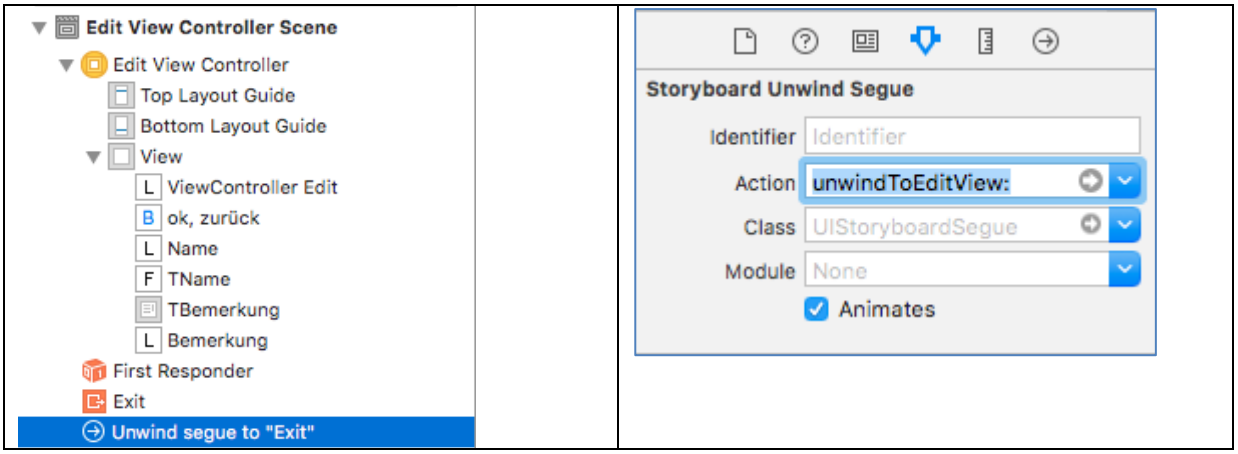


Abbildung 29 Rücksprung-Methode für ein Segue im Property Inspector setzen

Abschlusslösung:

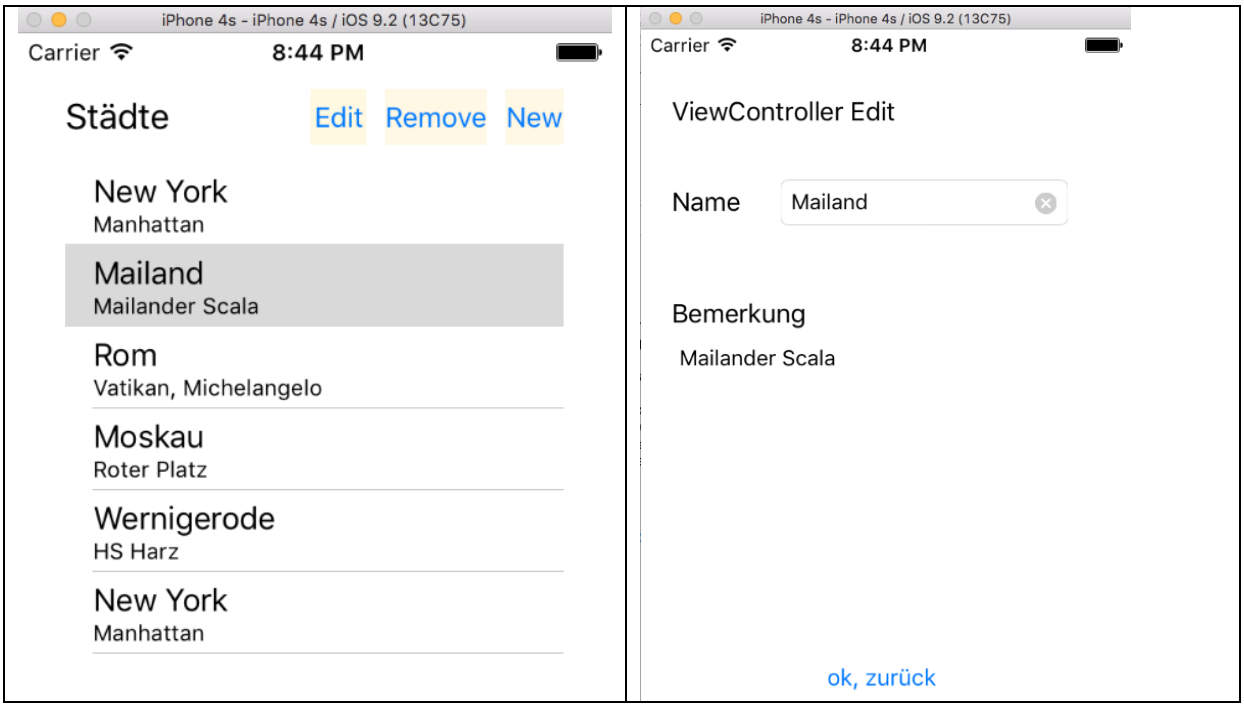


Abbildung 30 ViewController mit eigener Navigation

1.5 Master-Detail-Application

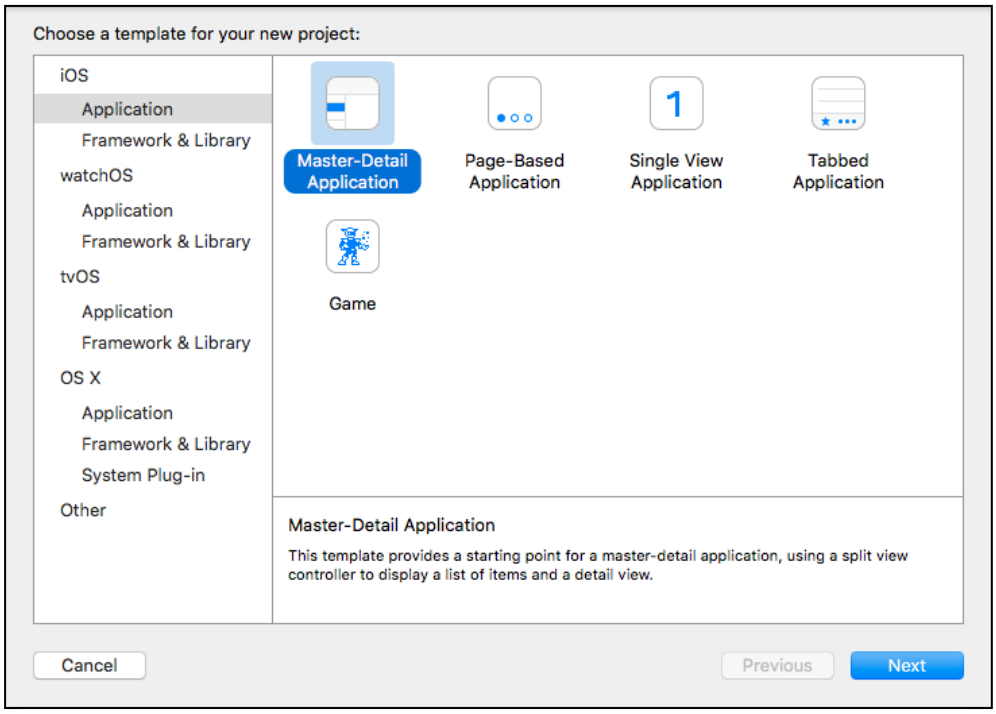


Abbildung 31 Master-Detail-Application

Dieses Framework liefert einen ansprechenden Rahmen für den TabellenView. Es hat vorgefertigt FÜNF grafische Hauptkomponten:

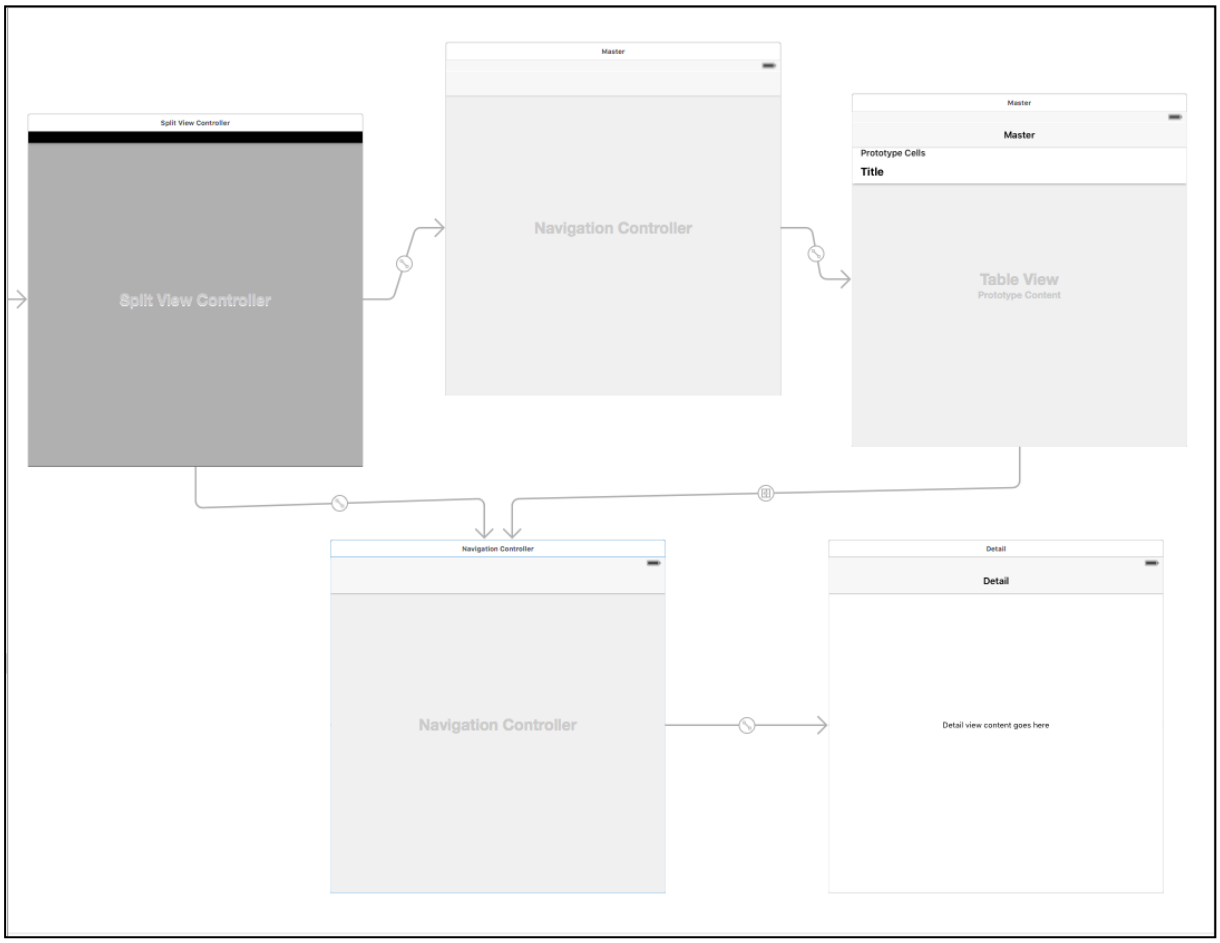


Abbildung 32 Komponenten einer Master-Detail-Application

Folgende Hauptkomponten existieren:

- Split-View-Controller
- Navigations-Controller
- **Table View**
- Navigations-Controller
- **Details View-Controller**

Für die Programmierung sind aber nur zwei Komponenten wichtig:

- **Table View**
  - Anzeige der „Menü“-Tabelleneinträge
- **Details View-Controller**
  - Anzeige der Details mit beliebigen Labeln, Editoren, Bildern etc.

Musterlösung:

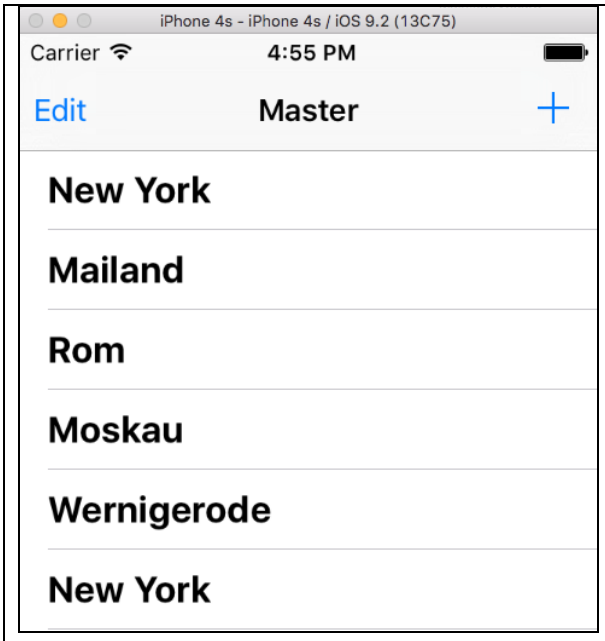


Abbildung 33 MenüView

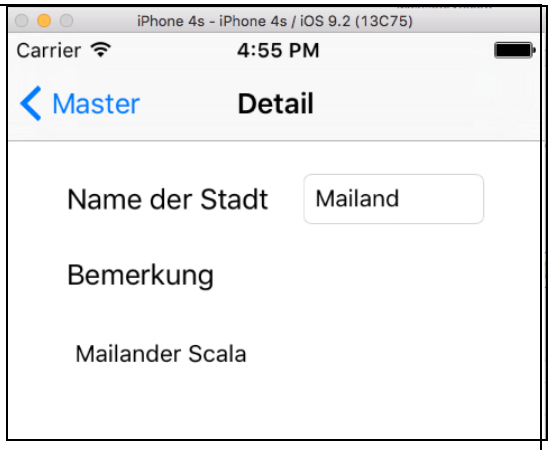


Abbildung 34 Anzeige der Details mit Editoren

Hinweis:

- Der Schalter „+“ erlaubt das Einfügen von neuen Daten

Erstellen des fertigen Projektes:

1. Anlegen einer neuen CocoaTouch Class

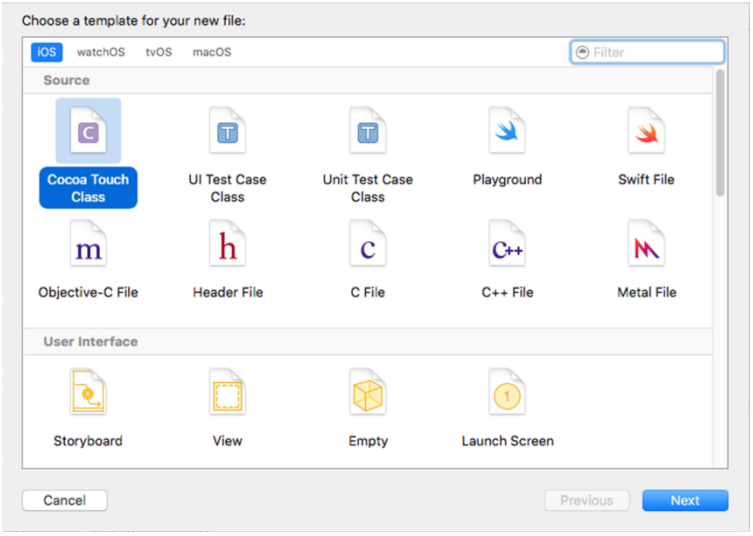
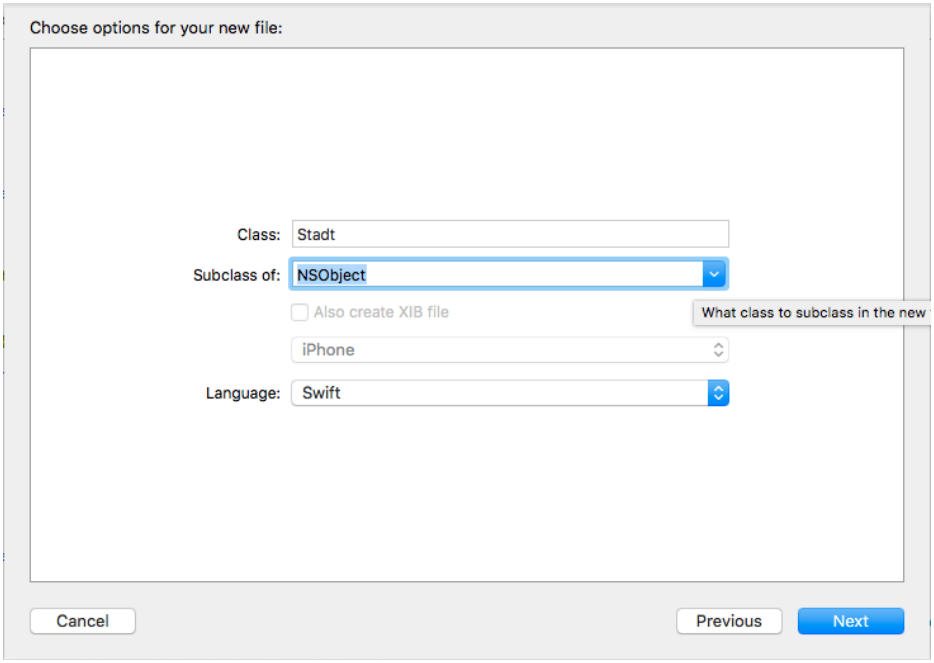


Abbildung 35 Neuer Eintrag "Cocoa Touch Class"

2. **Basis-Klasse auswählen: NSObject**



3. **Speichern unter dem Klassennamen, z. B. Stadt**

4. **Eintragen der Klassenstruktur:**

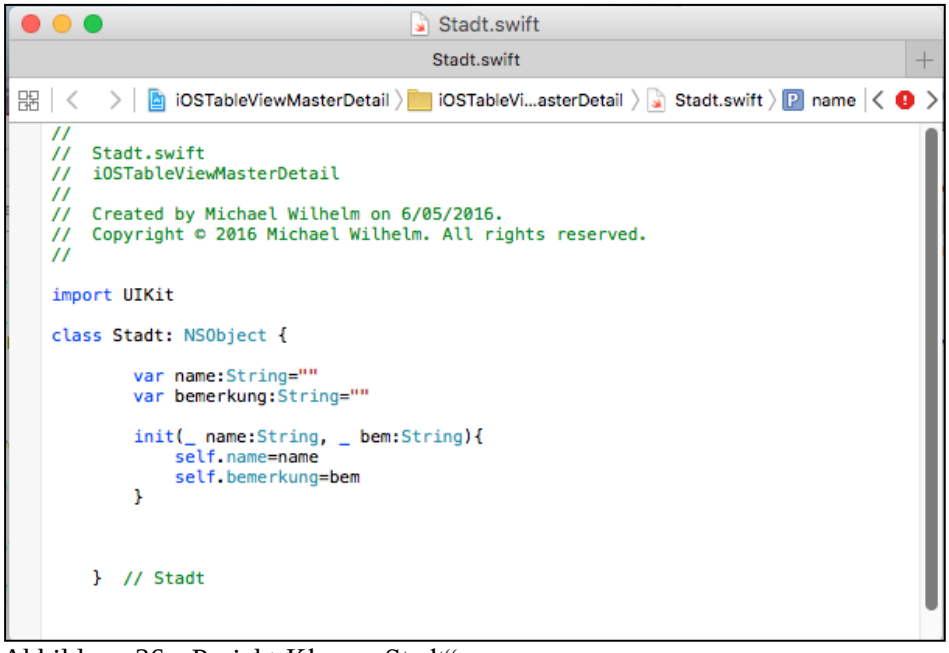


Abbildung 36 Projekt-Klasse „Stadt“

Damit kann man die obige Klasse im gesamten Projekt benutzen. Im nächsten Schritt wird der Maste-View so verändert, dass er mehrere Städte in einer Tabelle anzeigen kann.

5. **Modifizieren des MasterViewController**

Eintragen der Klassenglobalen variablen:

```
var staedte = [Stadt]()          // Liste der Städte
var neueStadtNr=0
```

Eintragen der Städte in die Liste:

```
override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after ...
        staedte.append( Stadt("New York", "Manhattan")  )
        staedte.append( Stadt("Mailand", "Mailander Scala")  )
        staedte.append( Stadt("Rom",  "Vatikan, Michelangelo")  )
        staedte.append( Stadt("Moskau", "Roter Platz")  )
        staedte.append( Stadt("Wernigerode", "HS Harz")  )
        staedte.append( Stadt("New York", "Manhattan")  )
```

Ändern von „object“ nach „staedte“:

```
~~var objects = [AnyObject]()~~  // Hauptarray  löschen

func insertNewObject(sender: AnyObject) {
    neueStadtNr++
    let stadt:String = "Stadt"+String(neueStadtNr)
    staedte.append( Stadt(stadt, stadt+stadt ) )

    ...
}

override func prepareForSegue(segue: UIStoryboardSegue,
        sender: AnyObject?) {
    if segue.identifier == "showDetail" {
      if let indexPath =
            self.tableView.indexPathForSelectedRow {
      let stadt:Stadt = staedte[indexPath.row]  // as! Stadt
      let detailviewcontroller =
         (segue.destinationViewController as!
             UINavigationController).topViewController as!
                 DetailViewController
            detailviewcontroller.detailStadt = stadt
       ...
        }
    }
}
```

Bitte beachten Sie, dass die “Übergabe-Variable” hier schon verändert wurde.

```
  detailviewcontroller.detailStadt = stadt
```

statt

```
  controller.detailItem = object
```

```
override func tableView(tableView: UITableView,
        numberOfRowsInSection section: Int) -> Int {
    return staedte.count
}

override func tableView(tableView: UITableView,
     cellForRowAtIndexPath indexPath: NSIndexPath)
        -> UITableViewCell {
let cell = tableView.dequeueReusableCellWithIdentifier(
```

```
                "Cell", forIndexPath: indexPath)
    let stadt:Stadt = staedte[indexPath.row] // as! NSDate
    cell.textLabel!.text = stadt.name
    return cell
}

override func tableView(tableView: UITableView,
    commitEditingStyle editingStyle:
    UITableViewCellEditingStyle, forRowAtIndexPath
        indexPath: NSIndexPath) {

    if editingStyle == .Delete {
       staedte.removeAtIndex(indexPath.row)
       tableView.deleteRowsAtIndexPaths([indexPath],
                 withRowAnimation: .Fade)
    }
    else if editingStyle == .Insert {
    }
}
```

Damit sind alle Änderungen im Master-View angeschlossen. Es fehlt nur noch die Änderung in den Details-Controller.

6. Modifizieren des DetailsViewController

a) Löschen des Labels

b) Eintragen von zwei Label
Eintragen eines Textfield
Eintragen eines TextView (Multi-Line-Editor)

c) Eintragen von Constraints

Label: „Name der Stadt“

- Top: 10 Pixel
- Left: 10 Pixel
- Right: 10 Pixel
- Bottom: 10 Pixel
- Ausrichtung: zentriert

TextField: Name der Stadt aus der Instanz

- Top: 10 Pixel
- Left: 10 Pixel
- Right: 10 Pixel
- Bottom: 10 Pixel

Label: Bemerkung

- Top: 10 Pixel
- Left: 10 Pixel
- Bottom: 10 Pixel

TextView: Bemerkung der Stadt aus der Instanz

- Top: 10 Pixel
- Left: 10 Pixel
- Right: 10 Pixel
- Bottom: 10 Pixel

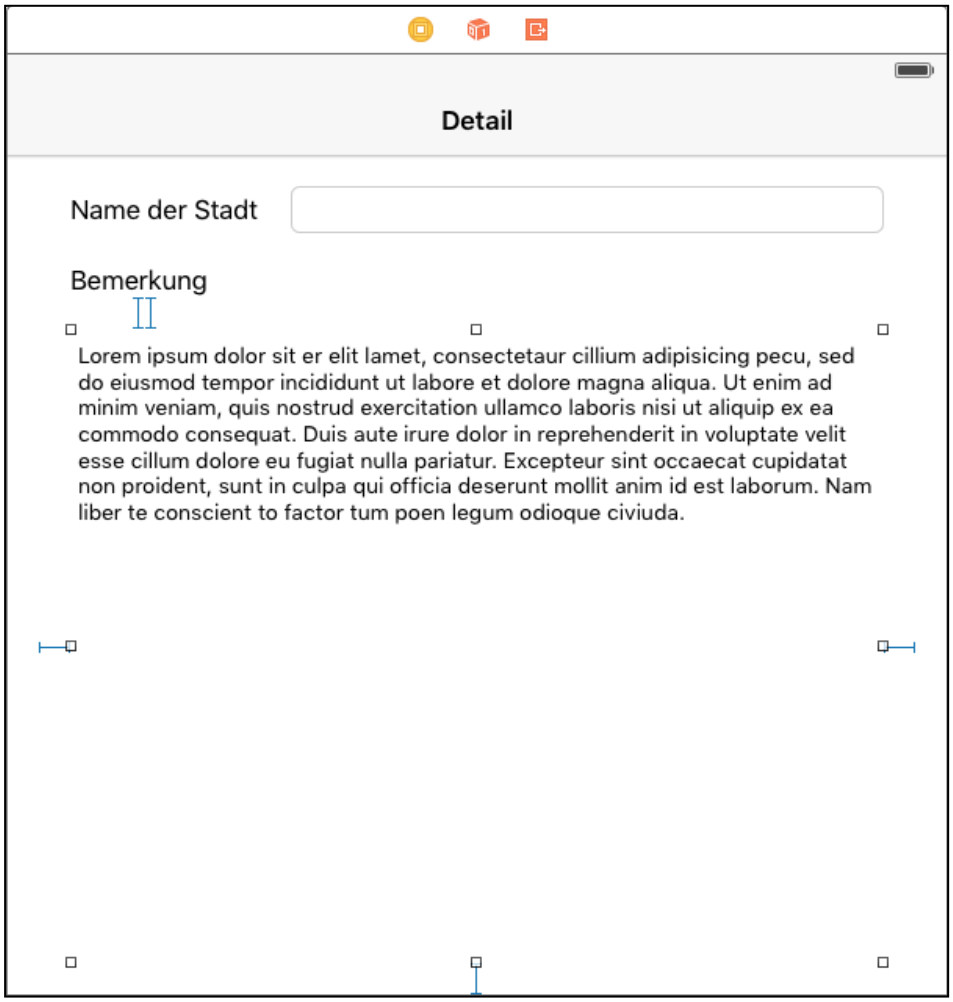


Abbildung 37 Constraints der DetailsViewController

7. Erstellen der beiden Outlets per Drag & Drop

```
@IBOutlet var tStadtName: UITextField!
@IBOutlet var tStadtBemerkung: UITextView!
```

8. Ändern der Anzeige-Programmierung (uiElemente, Klasse Stadt):

```
var detailStadt:Stadt? = nil {
        didSet {
            // Update the view.
            //self.configureView()  // wenn aktiv, Absturz
        }
    }

    func configureView() {
        // Update the user interface for the detail item.
```

```
    if let stadt:Stadt = self.detailStadt {
        tStadtName.text = stadt.name
        tStadtBemerkung.text = stadt.bemerkung
    }
}

override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup ...
    // Testanweisungen
    //tStadtName.text = "stadt.name"
    //tStadtBemerkung.text = "stadt.bemerkung"
    self.configureView()
}
```

2 PageBased

Die PageBased-Application ist eine „Vorlage“ zum Anzeigen von Seiten-Orientierten Daten. dazu gehören PDF-Dateien oder eine Auflistung von Kochrezepten.

Grundstruktur:

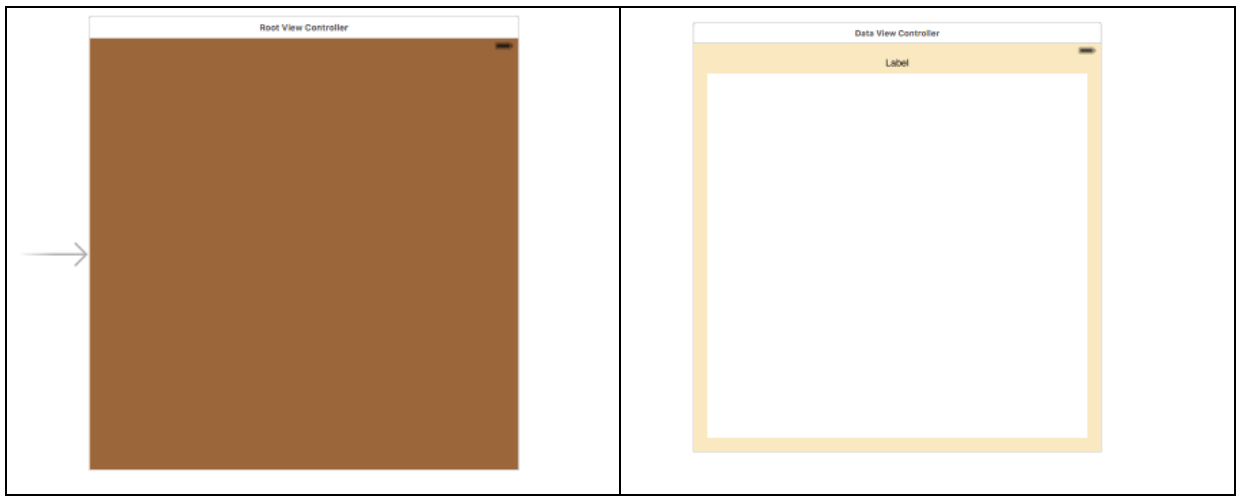


Abbildung 38 Grundstruktur einer Paged-Based Application

Insgesamt gibt es zwei UI-Elemente und drei Code-Dateien:

- Root View Controller
- Data View Controller
- RootViewController.swift
- ModelController.swift hier werden die Daten gespeichert
- DataViewController.swift hier werden die Daten angezeigt

2.1 Quellcode ModelController

ES werden zwei Array verwaltet, besser wäre aber ein Array von Instanzen.

```
class ModelController: NSObject, UIPageViewControllerDataSource {

    var pageData: [String] = []
    var pageRemark: [String] = []


    override init() {
        super.init()
        // Create the data model.
        pageData.append("AI")
        pageData.append("VW")
        pageData.append("W")
        pageData.append("Medizin")

        pageRemark.append("Automatisierung und Informatik in
Wernigerode")
        pageRemark.append("Verwaltungswissenschaften in
Halberstadt")
```

```
        pageRemark.append("Wirtschaftswissenschaften in
Wernigerode")
        pageRemark.append("Medizin auf dem Friedhof,\nHintere
Gruft neben Dracula")


    }

    func viewControllerAtIndex(index: Int,
           storyboard: UIStoryboard) -> DataViewController? {
        // Return the data view controller for the given index.
        if (self.pageData.count == 0) ||
                            (index >= self.pageData.count) {
            return nil
        }
        let dataViewController =
             storyboard.instantiateViewControllerWithIdentifier(
                 "DataViewController") as! DataViewController
        dataViewController.dataObject = self.pageData[index]
        dataViewController.remarkObject = self.pageRemark[index]
        return dataViewController
    }


}
```

2.2 Quellcode Data ViewController

Der DataViewController hat für die Übergabe zwei öffentliche Variablen:

- dataObject
- remarkObject

Um die Daten anzuzeigen, hat der DataViewController ein Label und ein TextView.

Quellcode:

```
class DataViewController: UIViewController {

     // öffentlcihe Variablen (Transport)
    var dataObject: String = ""
    var remarkObject: String = ""

     // uiAttribute
    @IBOutlet weak var dataLabel: UILabel!
    @IBOutlet var tBemerkung: UITextView!


    override func viewDidLoad() {
        super.viewDidLoad()

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
```

```
        // Dispose of any resources that can be recreated.
    }

    override func viewWillAppear(animated: Bool) {
        super.viewWillAppear(animated)
        self.dataLabel!.text = dataObject
        tBemerkung.text = remarkObject
    }


}
```

2.3 Musterlösung

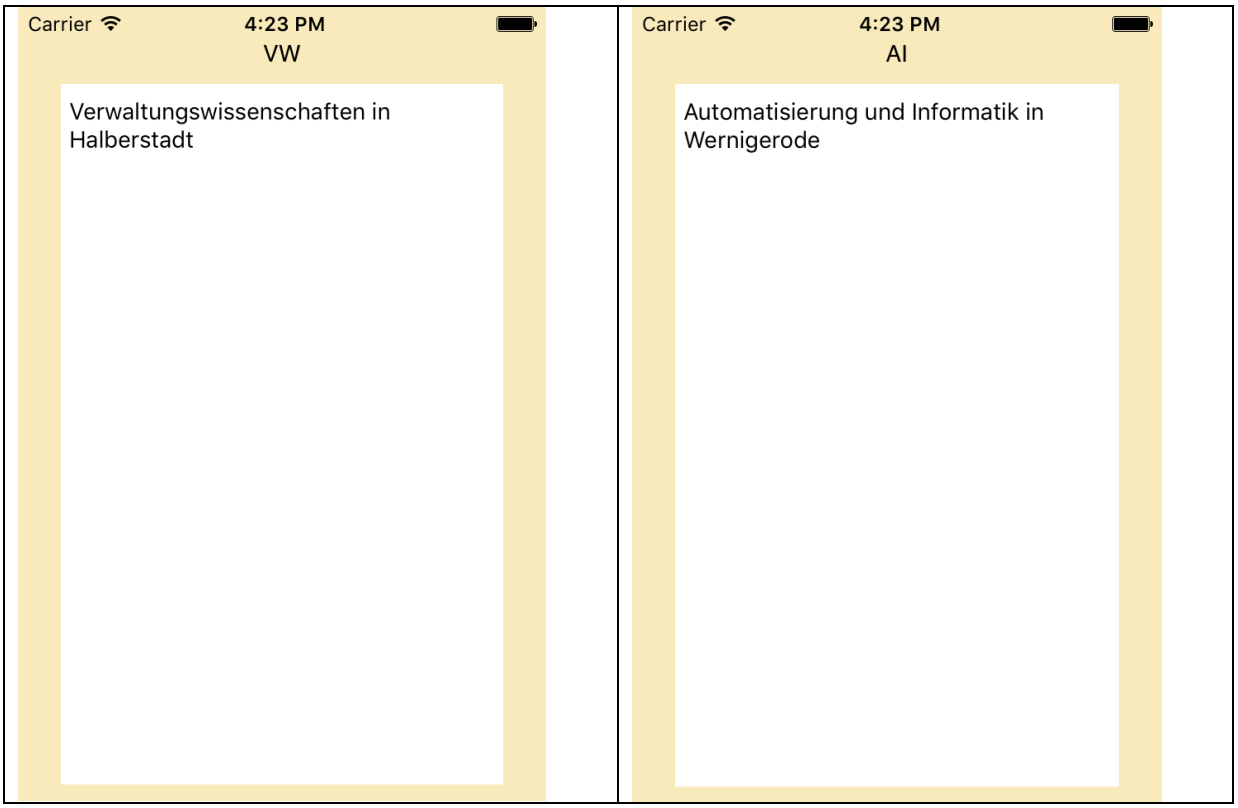


Abbildung 39 Paged-Based Application nach einer Modifikation