Beta 2.0

Full Stack iOS Entwicklung mit Swift

WPF im MIM - WS 17/18

Alexander Dobrynin, M.Sc.

Heute

TableView
UITableViewController
TableViews und MVC
Datenstrukturen für TableViews

Demo
Assignment

TableView

- TableViews werden verwendet, um große Mengen an Daten in einer vertikal scrollbaren Liste zu präsentieren
- Die Daten können Gruppiert werden, um diese visuell oder semantisch aufzuteilen
    - Jede Gruppe präsentiert eine bestimmte Anzahl von Daten
    - Jede Gruppe kann eine eigene Über- und/oder Unterschrift (String) oder jeweils eine eigene View für beides haben
- Viele TableViews werden auch für bestimmte Zwecke missbraucht, weil sie
    - das flexible Layouting einer StackView innerhalb einer ScrollView mit
    - der Mächtigkeit von heterogenen CustomViews vereinen und
    - das Einfügen, Löschen, Verschieben und Aktualisieren der Inhalte animieren
- Zudem lassen sich TableViews besonders gut mit CoreData (ORM) nutzen nicht im FSIOS Scope
    - Die Daten einer CoreData-Query werden in einer TableView angezeigt
    - Sobald sich die Daten in CoreData ändern, wird die TableView aktualisiert
    - Unter Umständen lassen sich diese Änderungen sogar innerhalb der TableView animieren
- Des Weiteren verfügt jede TableView über ein RefreshControl und einer Pull-To-Refresh Funktion
- Einer Alternative zu TableViews sind CollectionViews, welche beliebige Daten in einem vertikal scrollbaren Grid anordnen

TableView

- Im Storyboard findet sich ein UITableViewController, welcher eine UITableView als seine SuperView hat
- Es gibt statische und dynamische TableViews
    - Statische TableViews haben statische Daten (z.B. Settings) und können lediglich auf Interaktionen der Daten reagieren
    - Dynamische TableViews erhalten ihre Daten zur Laufzeit (z.B. Datenbank oder Netzwerk)
- Dynamische TableViews können nichts ohne ihre DataSource, welche die Daten zur Laufzeit liefert
- Interaktionen mit dynamischen oder statischen TableViews haben keinen Effekt ohne ihr Delegate
- Eine TableView präsentiert TableViewCells, die das UI für die Daten darstellen
    - Jede Cell wird über einen IndexPath, bestehend aus einer Section und einer Row, identifiziert
    - Die TableView fragt über die DataSource u.a. nach einer fertig konfigurierten Cell für einen IndexPath
- Demnach ist eine TableView nach Sections und Rows aufgeteilt wobei
    - jede Row eine Cell hat und
    - jede Section entweder einen Titel (String) oder eine CustomView als Footer und/oder Header haben kann
- Grundsätzlich haben TableViews zwei unterschiedliche Styles, nämlich .grouped und .plain
- Zuletzt kann eine TableView als solches eine Header- und/oder eine FooterView haben

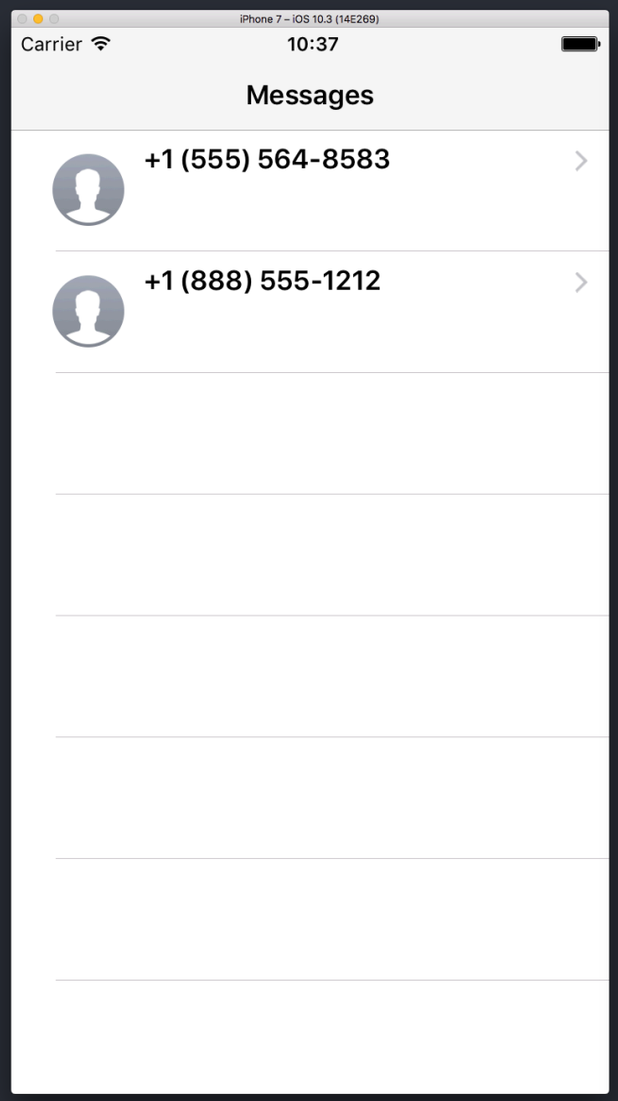
TableView

TL;DR: TableViews kategorisieren sich grob nach

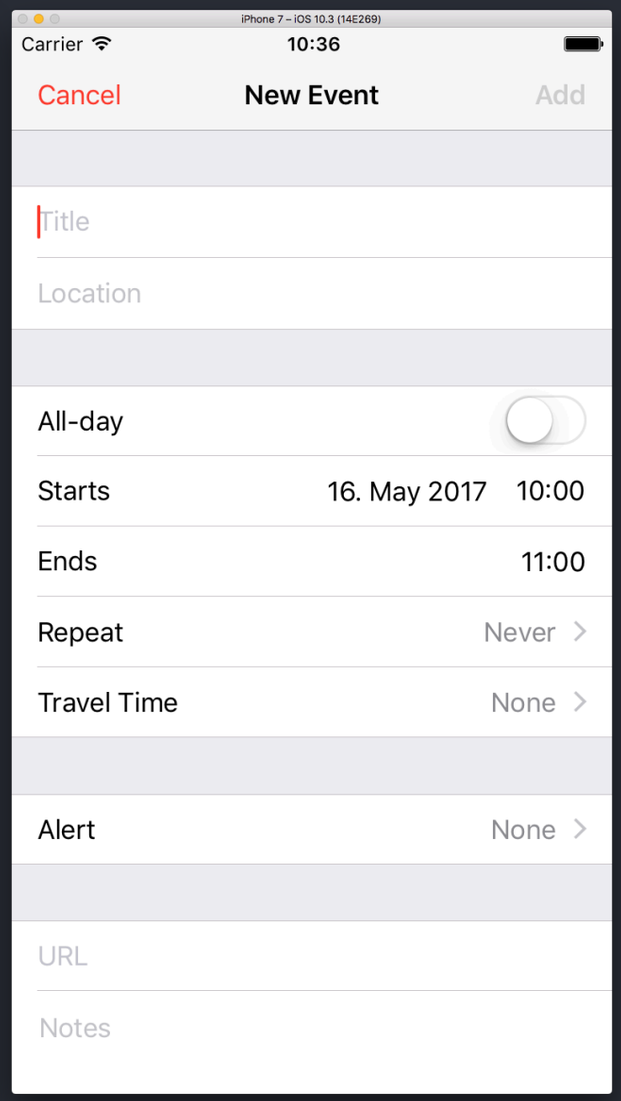
1. Style (grouped, plain)
2. Art der Daten (statisch, dynamisch)
3. Gruppierung der Daten (rows, sections und rows)
    1. Zellen (stock, custom; heterogen, homogen)
    2. Titel oder Subtext der Gruppierung (string, custom view)
4. View über- oder unterhalb der TableView

TableView

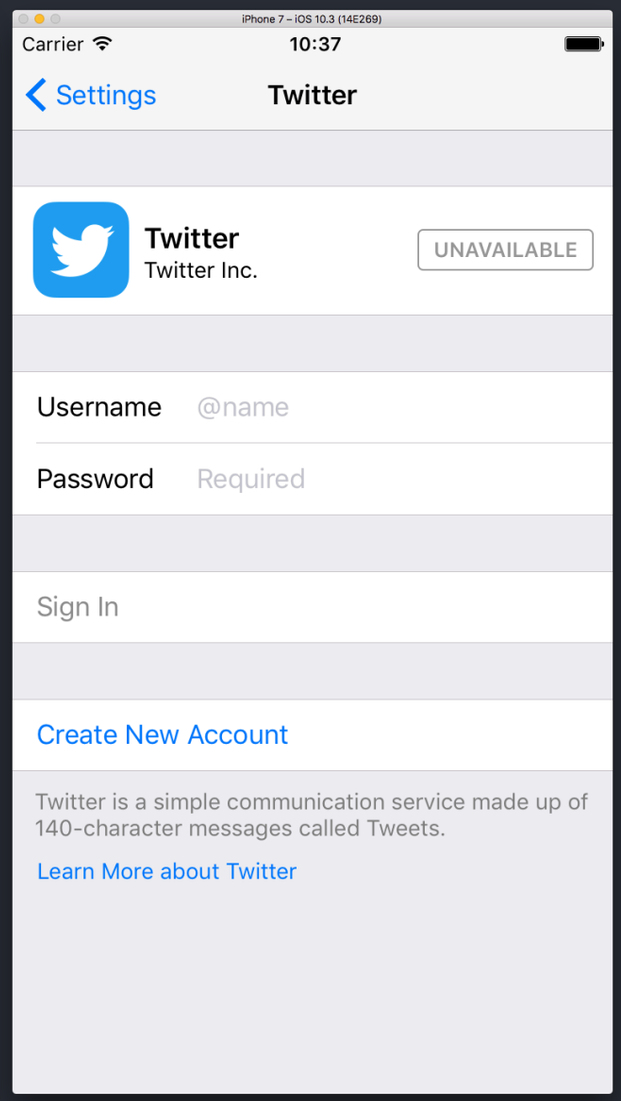
UITableViewController



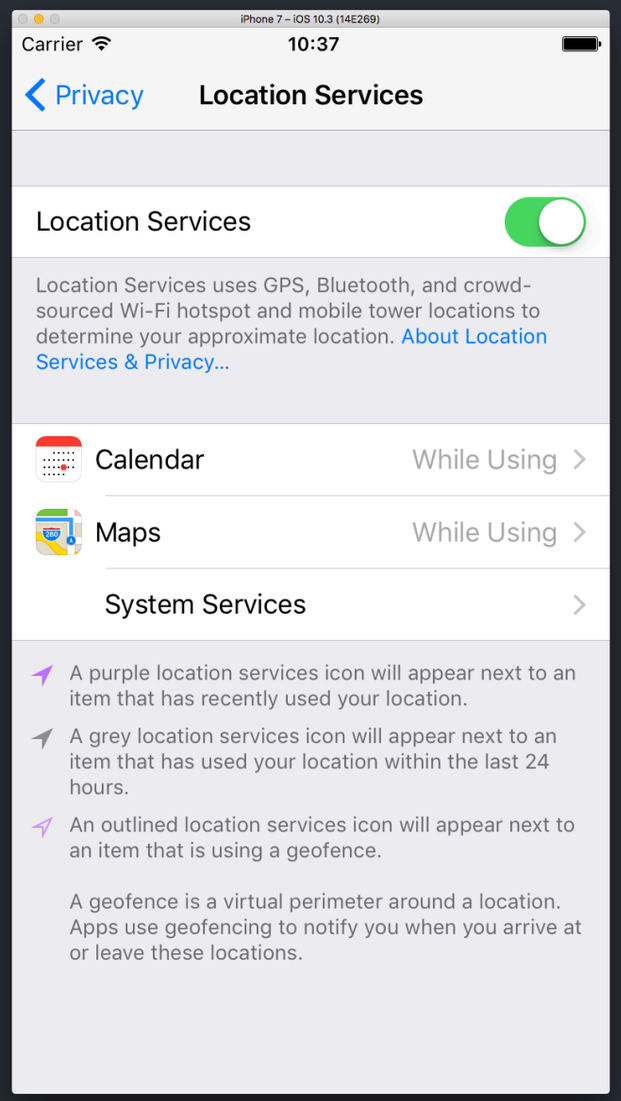
1. plain
2. dynamic
3. stock cell, homogen, no sections
4. no header or footer views



1. grouped
2. static (probably)
3. stock + custom cells, heterogen, sections without title
4. no header or footer views (probably)

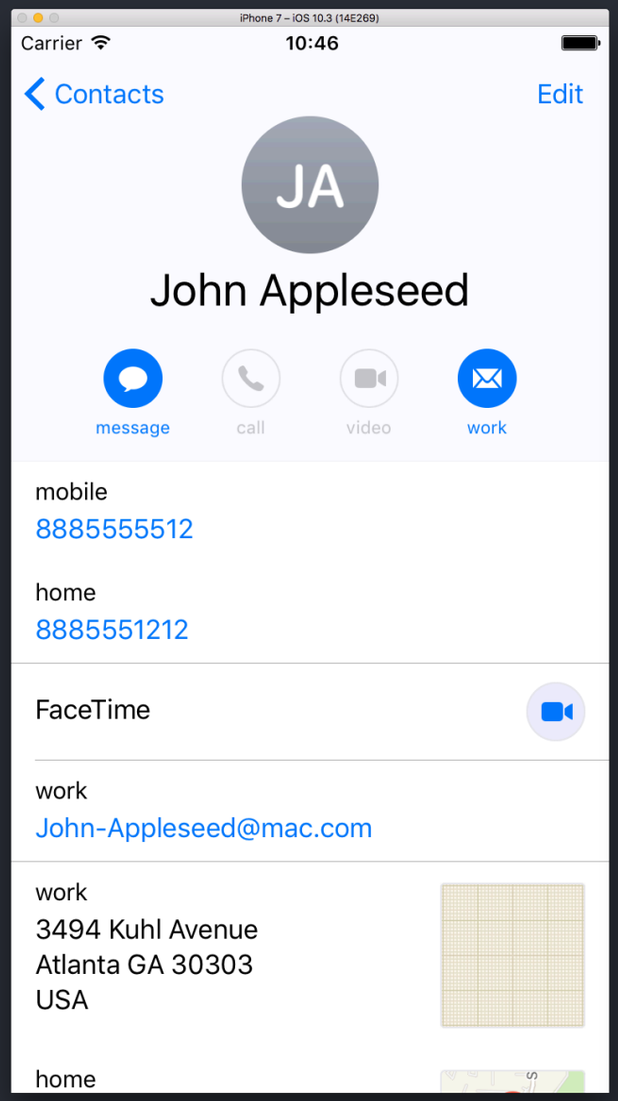


1. grouped
2. static (probably)
3. stock + custom cells, heterogen, one section with subtitle or custom footer view
4. no header or footer views (probably)

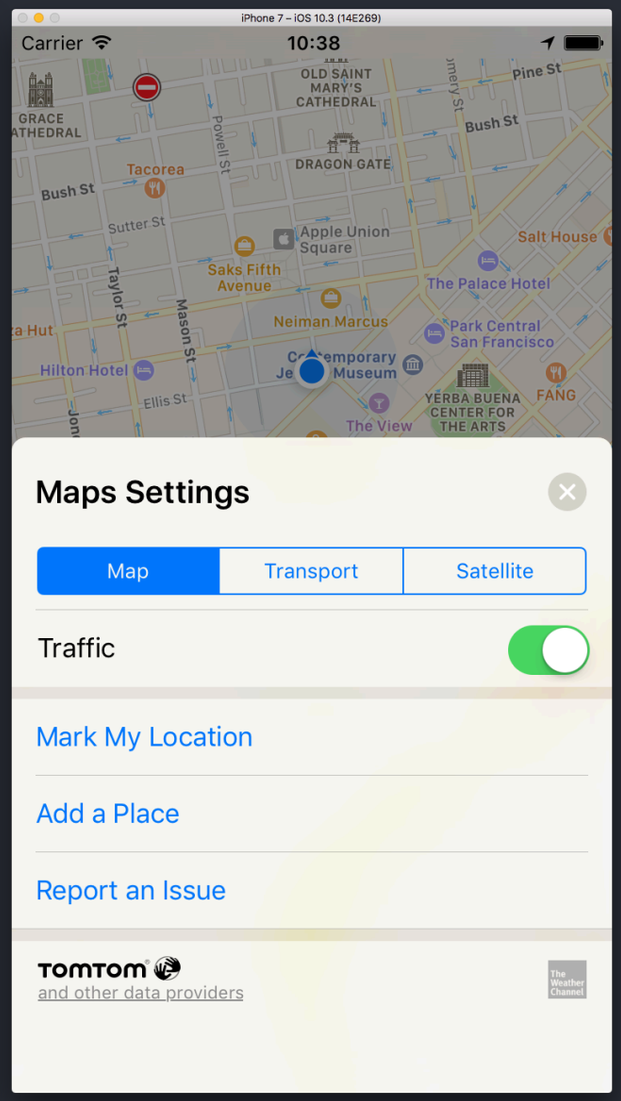


1. grouped
2. static (probably)
3. stock + custom cells, heterogen, each section with subtitle or custom footer view
4. no header or footer views (probably)

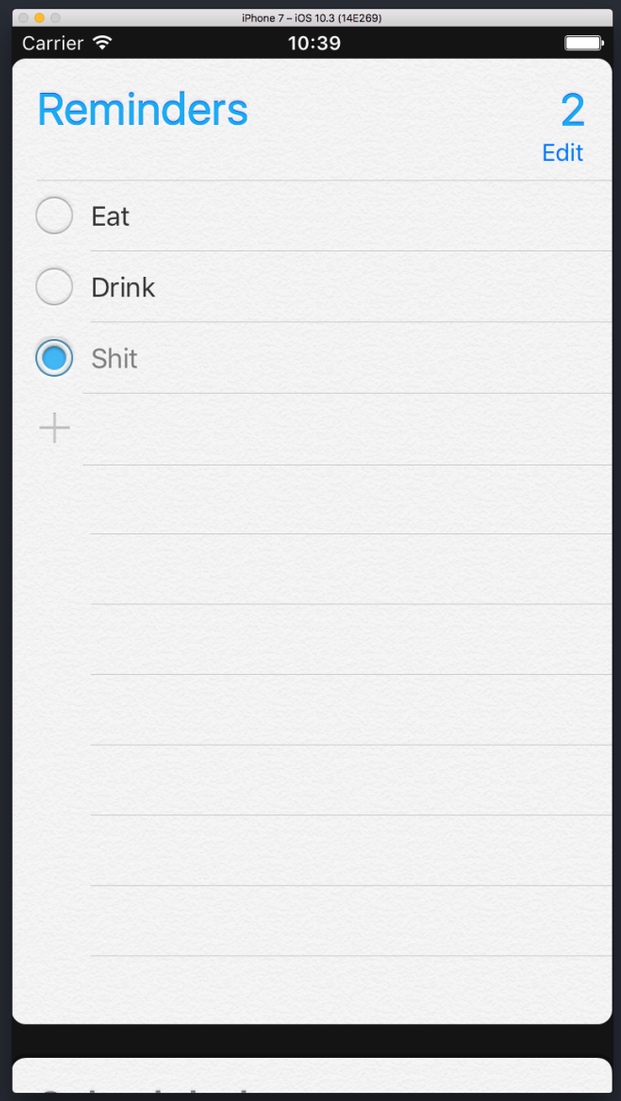
UITableViewController



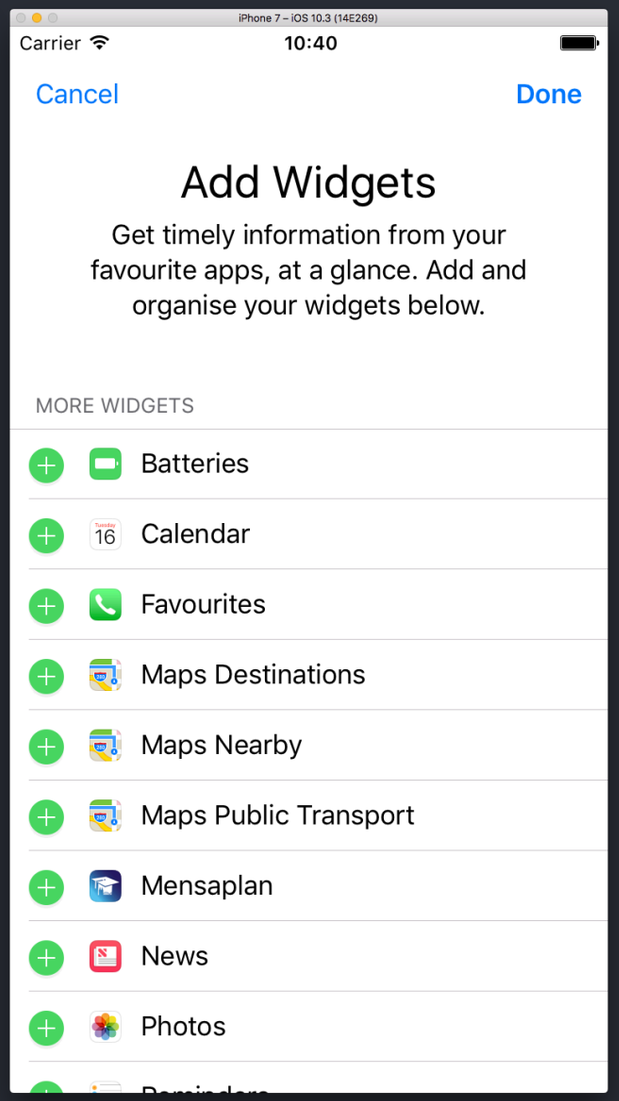
1. plain
2. dynamic
3. custom cells, heterogen, each section with title
4. header view avatar, name and actions



1. grouped
2. static (probably)
3. stock + custom cells, heterogen, sections without title (probably)
4. no header or footer views (probably)



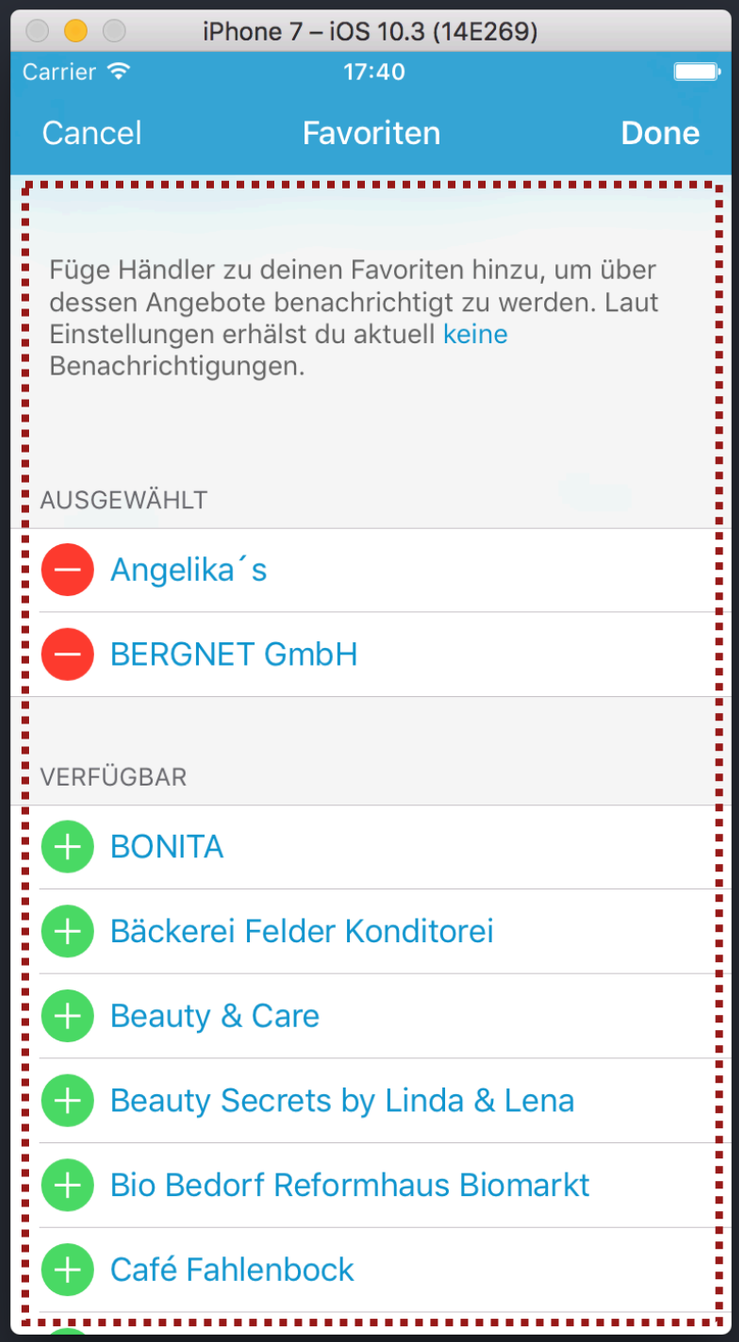
1. plan
2. dynamic
3. custom cells, heterogen, no sections
4. header view (probably)



1. grouped
2. dynamic
3. stock cells, homogen, each section with title
4. header view

UITableViewController

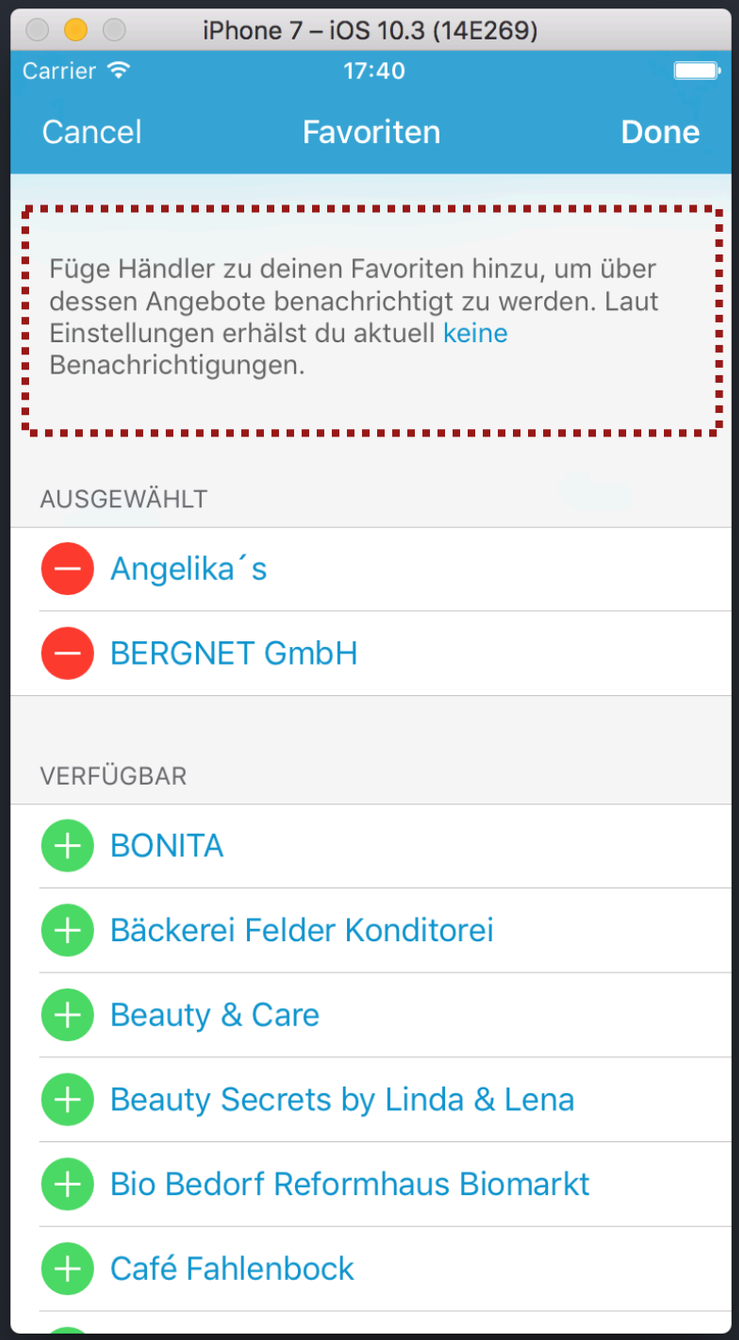
UITableViewController



UITableView

- Innerhalb eines UITableViewController
- Dynamische Daten aus der Datenbank

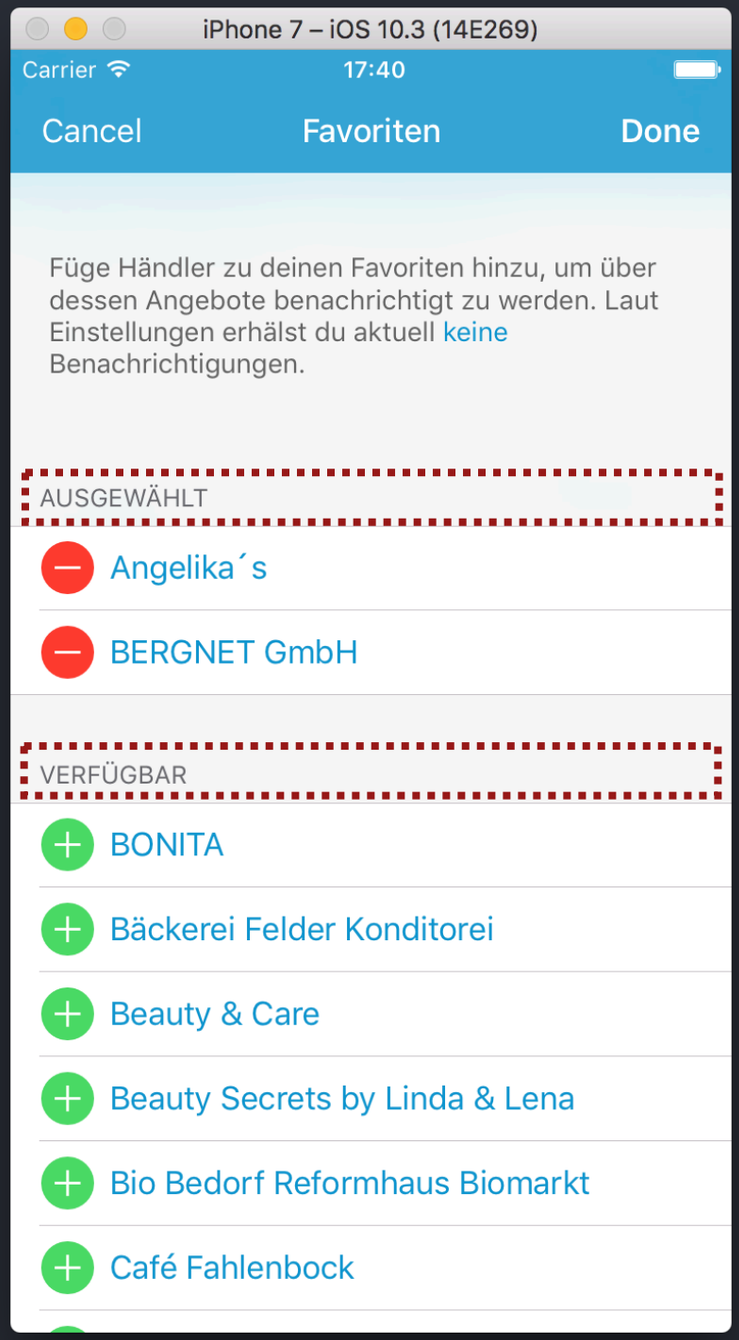
UITableViewController



Custom UITableHeaderView

- Inhalt ist ein UITextLabel mit einem NSAttributedText

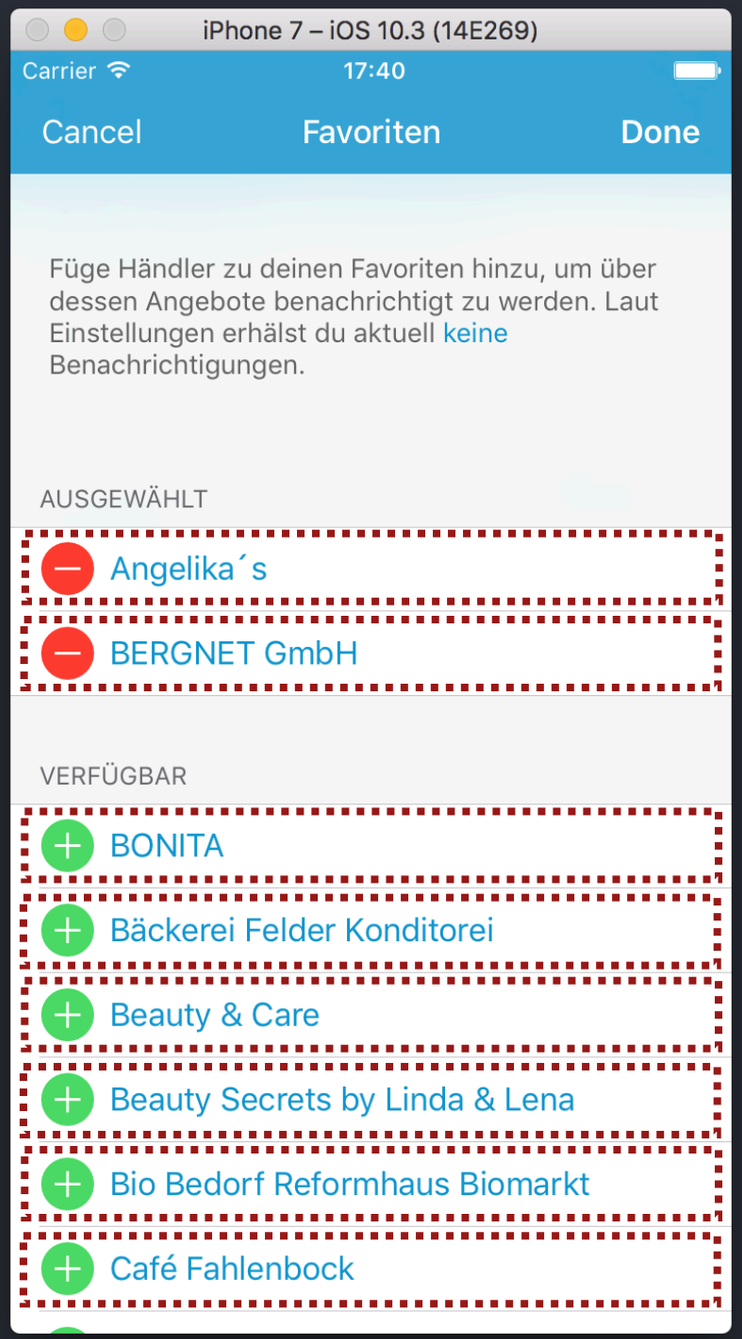
UITableViewController



Section

- Style ist grouped
- Title (String)
- 2 Sections
  - Stelle 0 mit 2 Cells
  - Stelle 1 mit min. 6 Cells

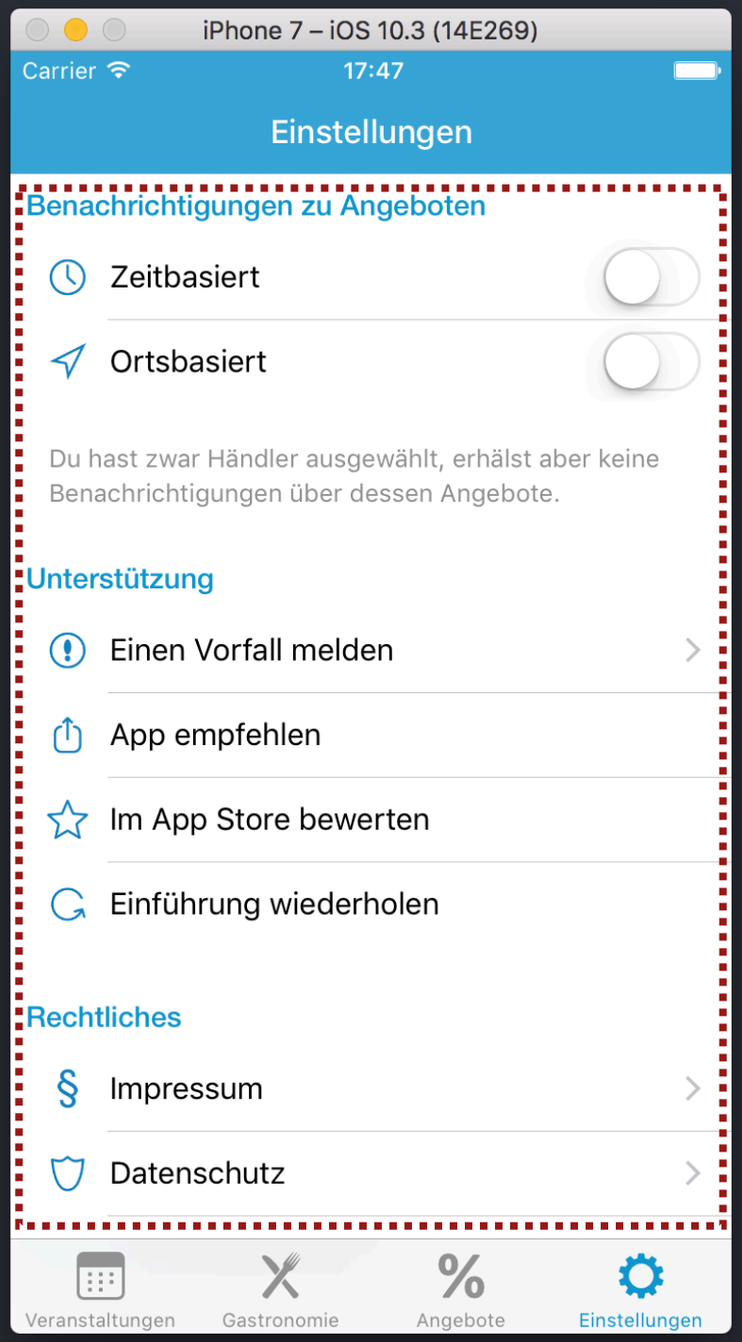
UITableViewController



Custom UITableViewCell

- Homogene Cells
- Innerhalb von Sections
- 2 in der ersten Section
  - section: 0, row: 0
  - section: 0, row: 1
- < 6 in der zweiten Section
  - section: 1, row: 0
  - section: 1, row: 1
  - section: 1, row: 2
  - …

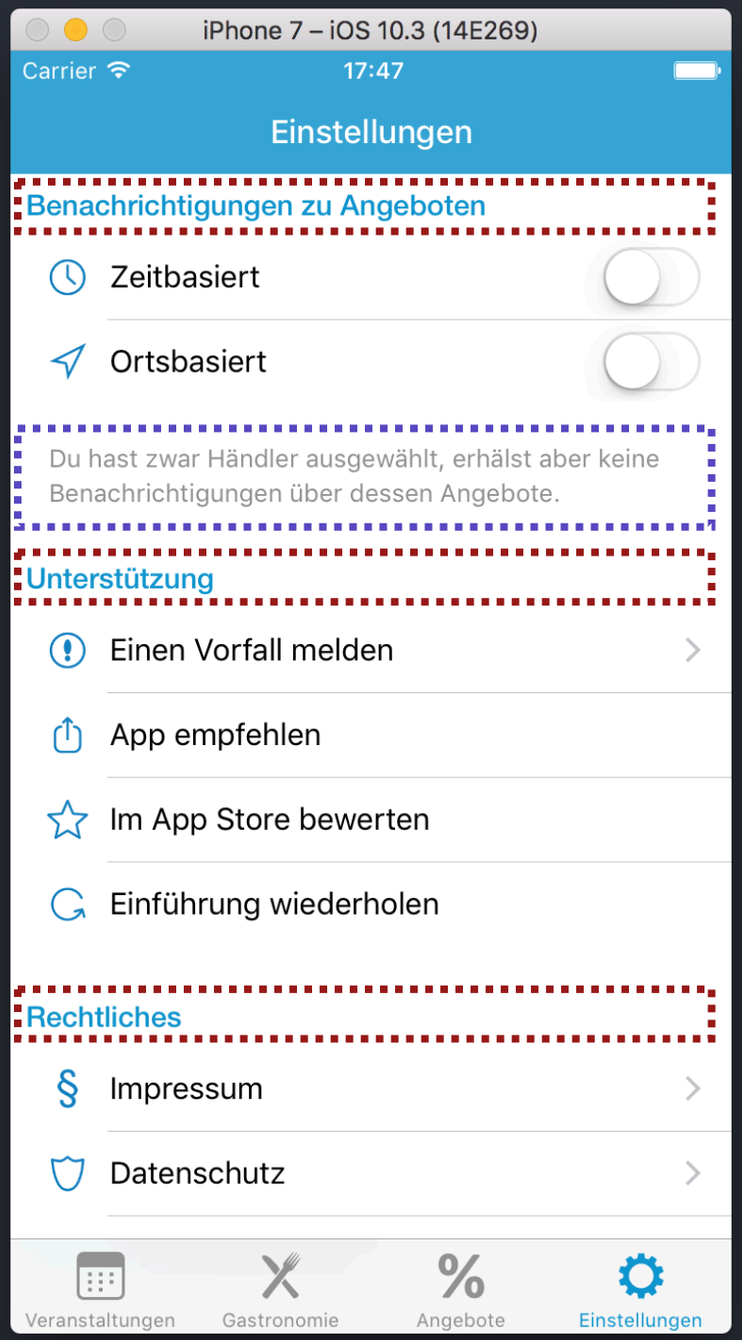
UITableViewController



UITableView

- Innerhalb eines
  - UITableViewController
  - UITabbarController
- Statische Daten

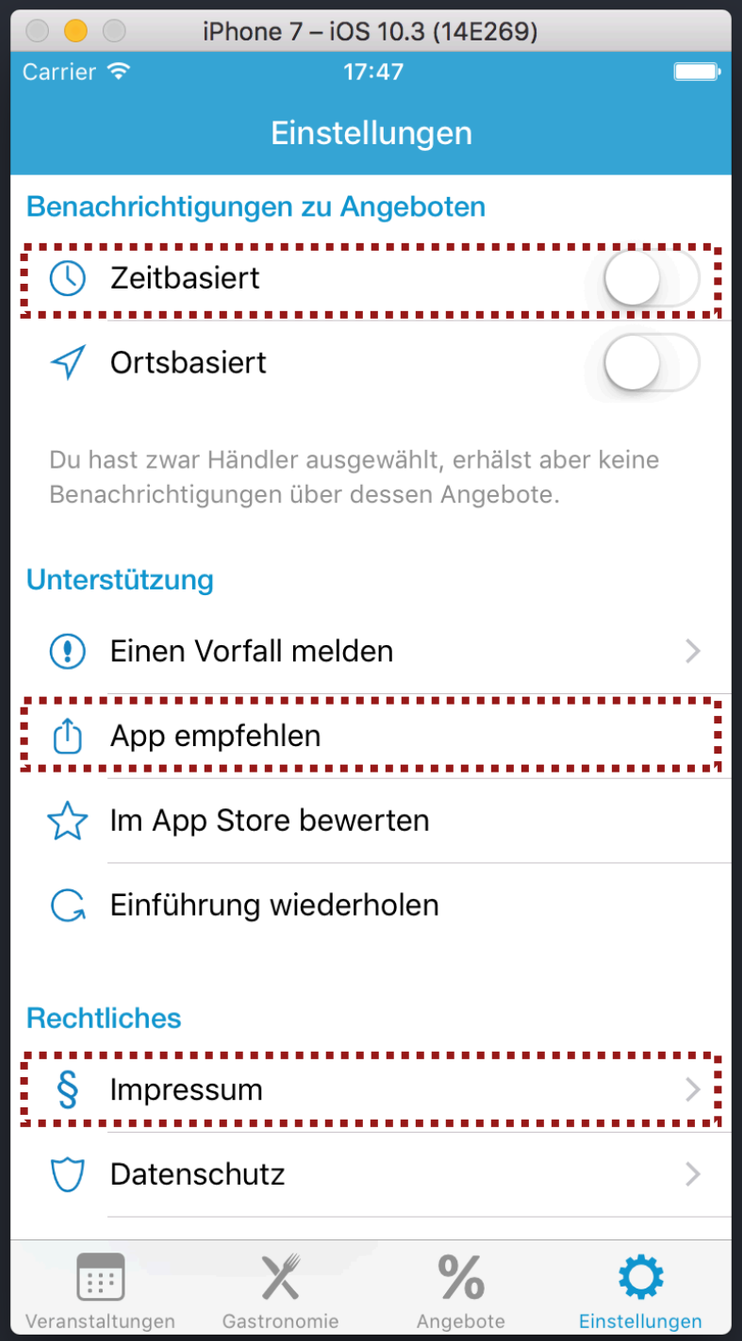
UITableViewController



Section

- Style ist plain
- Custom TitleView
- 3 Sections
  - Stelle 0 mit 2 Cells
  - Stelle 1 mit 4 Cells
  - Stelle 2 mit 3 Cells
- Section 0 mit Footer Subtitle (String)

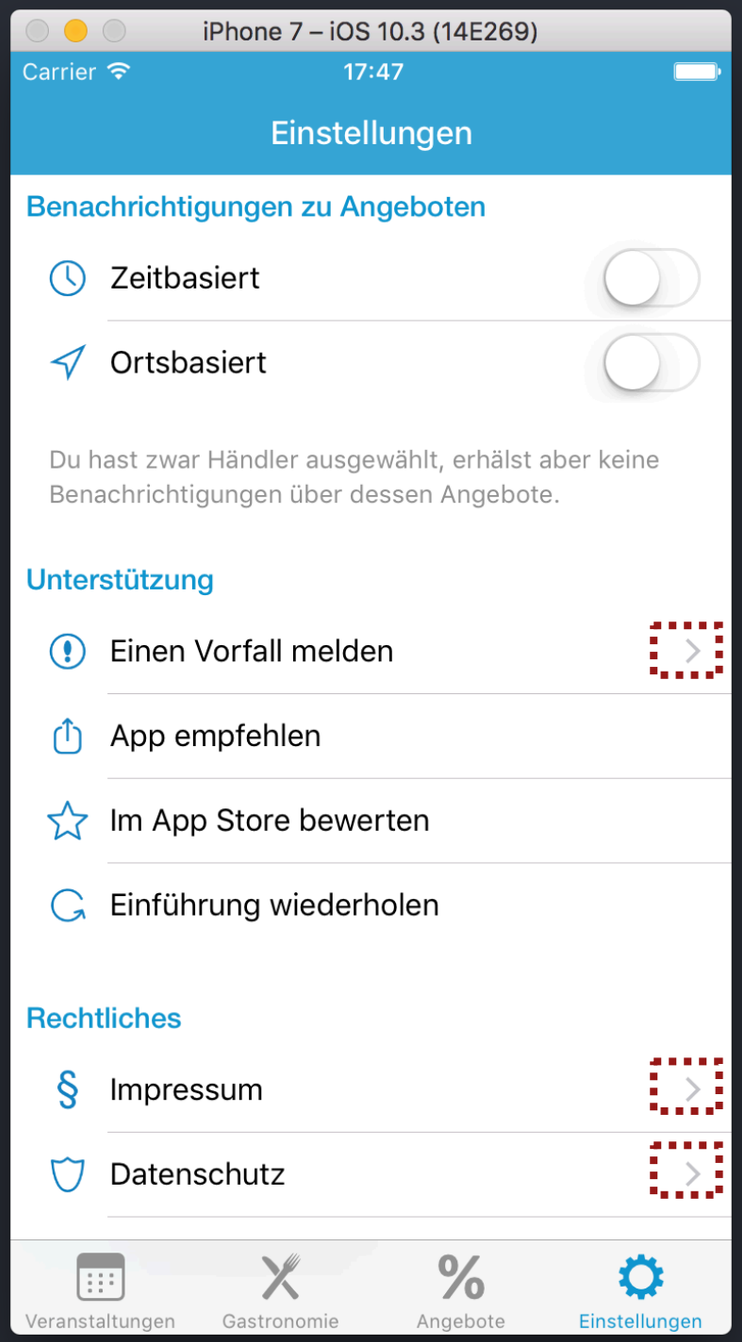
UITableViewController



Heterogene Cells

- Icon - Text - Switch
- Icon - Text
- Icon - Text - Accessory

UITableViewController



UITableViewCell Accessory

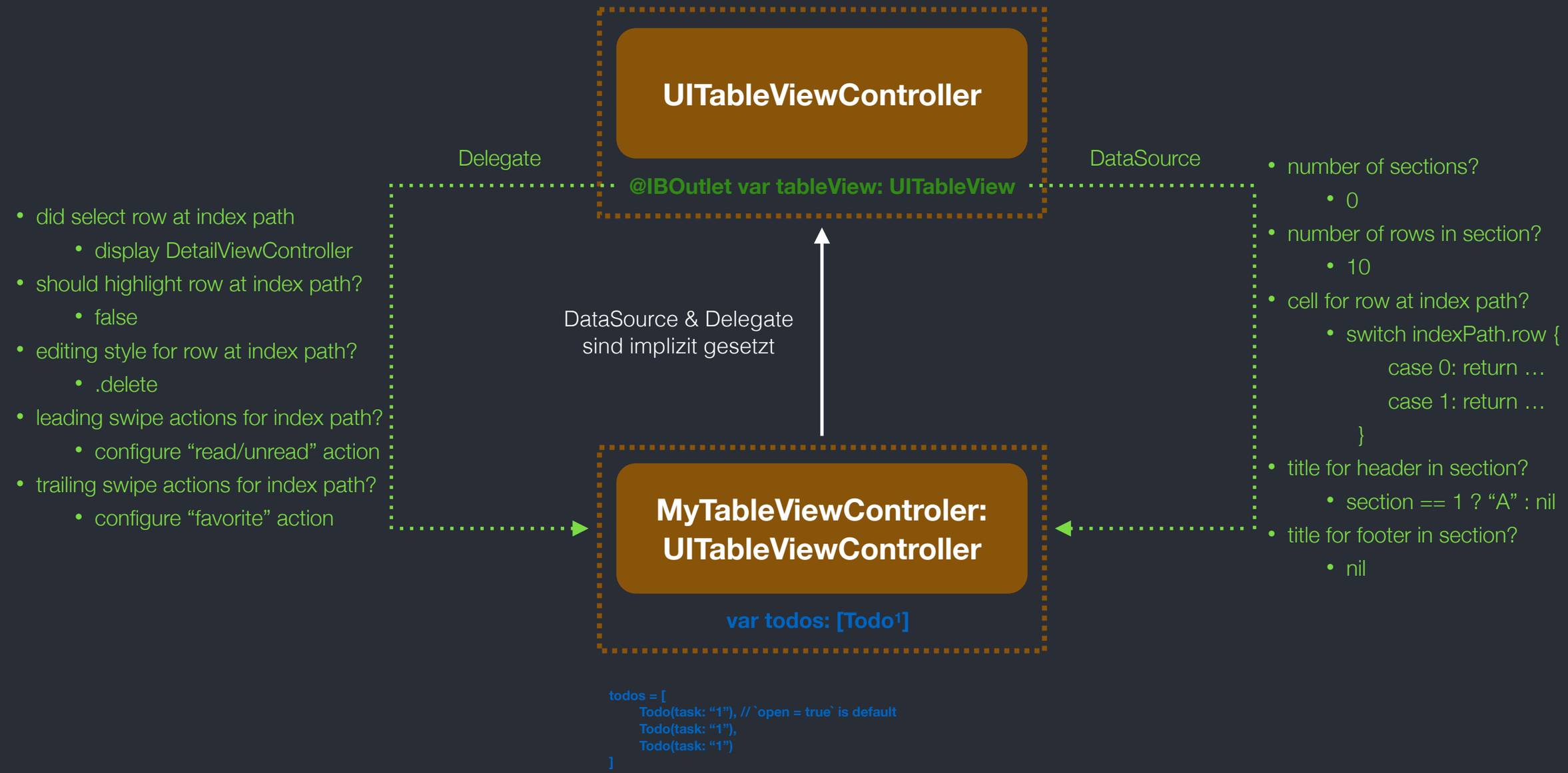
- Disclosure Indicator deutet auf einen Detail ViewController hin

Heute

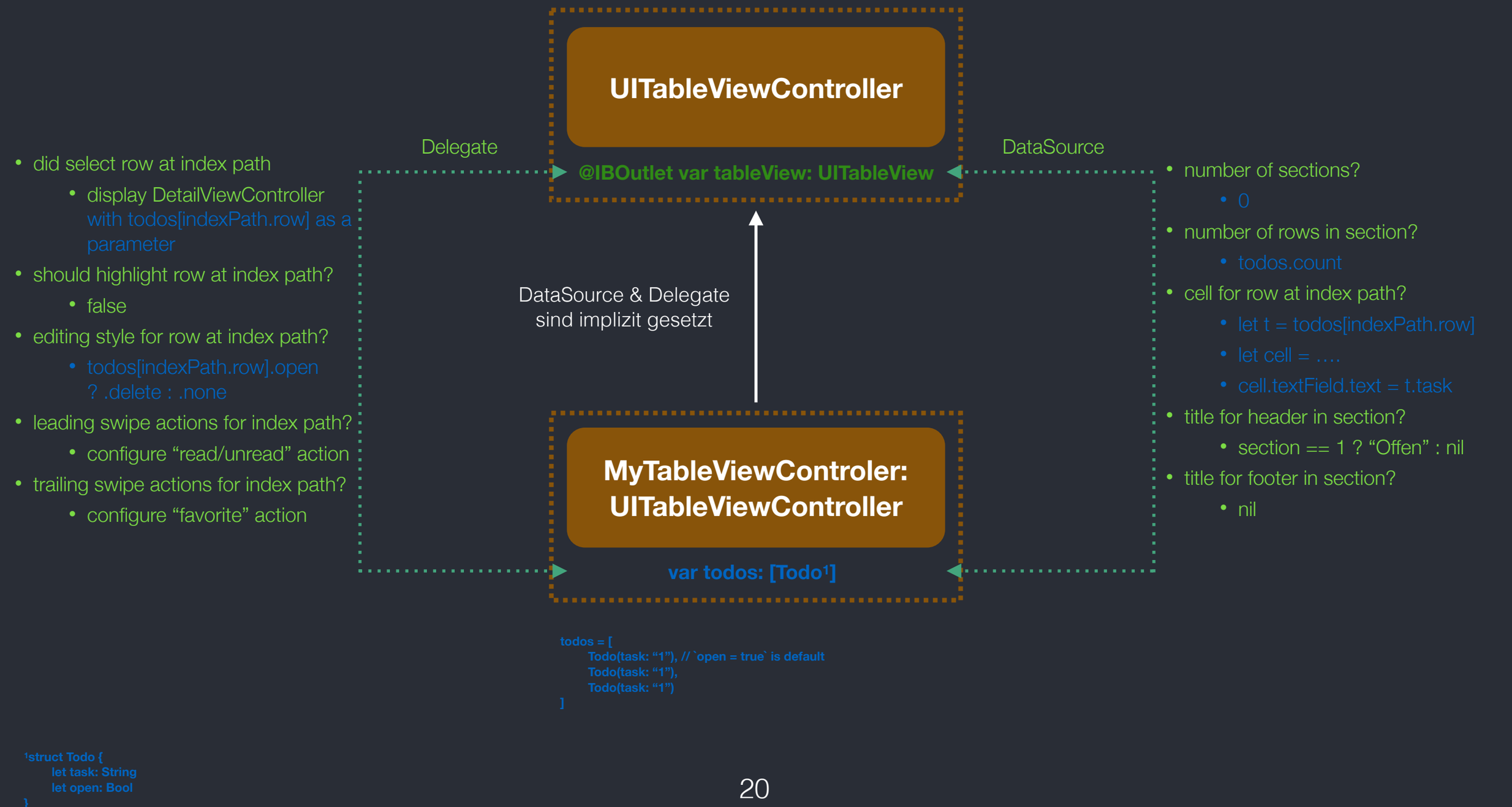
TableView
UITableViewController
TableViews und MVC
Datenstrukturen für TableViews

Demo
Assignment

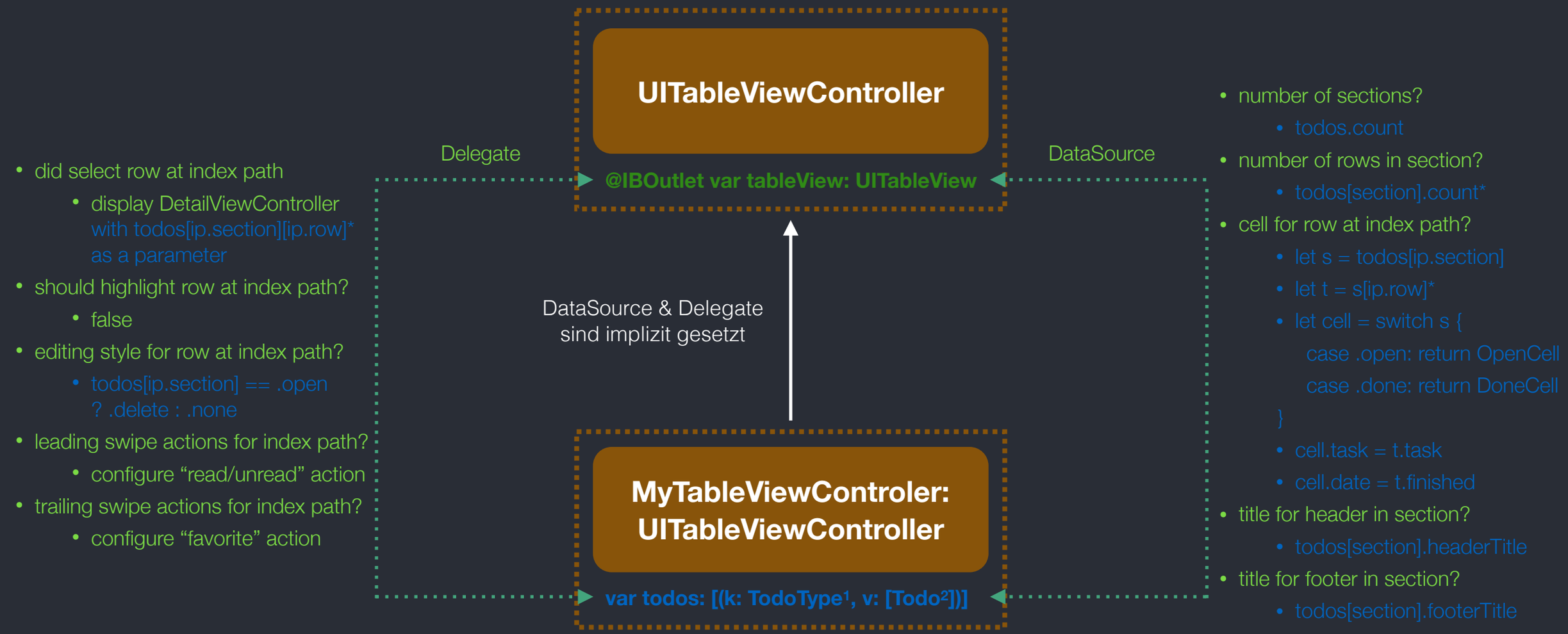
TableViews und MVC



TableViews und MVC



TableViews und MVC




```
todos = [
    (
        k: .open,
        v: [Todo(task: “1”), Todo(task: “2”), Todo(task: “3”)]
    ),
    (k: .done, v: [Todo(task: “4”), Todo(task: “5”)])
]
```

```
²struct Todo {
    let task: String
    let finished: Date?
}
```

```
¹enum TodoType {
    case open, done
    var headerTile: String { … }
}
```


*Die Syntax ist nicht 100% korrekt. Der Zugriff soll exemplarisch dargestellt werden

Heute

TableView
UITableViewController
TableViews und MVC
Datenstrukturen für TableViews

Demo
Assignment

Datenstrukturen für TableViews

- Je geeigneter die Datenstruktur des Models ist, desto einfacher ist die Implementierung des TableViewControllers
- Enums eignen sich, um heterogene Daten in einer Collection zu modellieren
    - Zudem können berechnete Properties die Fragen von DataSource und Delegate beantworten
    - Mit Pattern Matching und IndexPath kann der Typ eines Eintrags in der Collection aufgelöst werden
- High-Order-Functions ermöglichen jegliche Art von Manipulation auf Collections
    - groupBy*, filter, map, sortedBy, reduce, zip, …
    - Dadurch lässt sich ein Array<T> in jede beliebige Form bringen
    - Achtung: Sobald man ein Dictionary sortieren möchte, erhält man ein Array<(K, [V])>, weil Dictionaries kein Ordering haben

*Diese Funktion gibt es nicht in der Standard-Collection-Library.

Datenstrukturen für TableViews

Datenstruktur	Gruppiert	Reduziert	Sortiert
Array<V>	X	✓	✓
Dictionary<K: AnyHashable, Array<V>>	✓	✓	X
Array<(K, Array<V>)>	✓	✓	✓

Datenstrukturen für TableViews

Datenstruktur	Gruppiert	Reduziert	Sortiert
Array<V>	X	✓	✓
Dictionary<K: AnyHashable, Array<V>>	✓	✓	X
Array<(K, Array<V>)>	✓	✓	✓

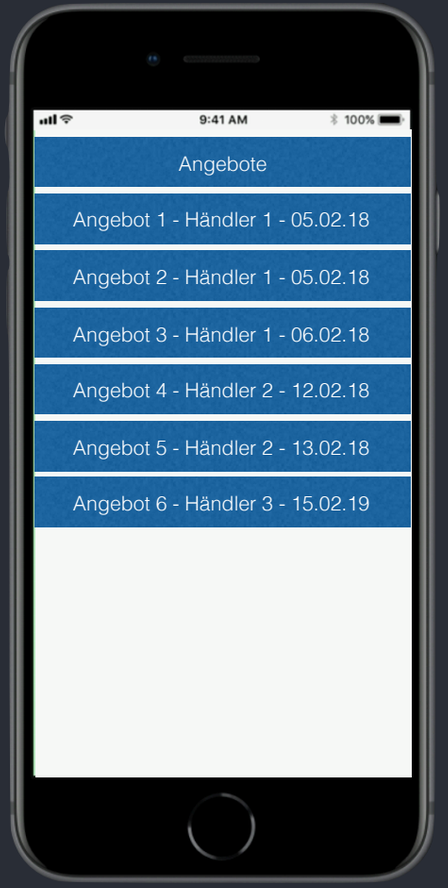
Given

```
struct Offer {
    let id: Int
    let retailerId: Int
    let name: String
    let oldPrice: Double?
    let newPrice: Double?
    let discount: Double?
    let fromDate: Date
    let fromTime: Date
    let toDate: Date
    let toTime: Date
}
```

[V]

```
var offers: [Offer] = [] {
    didSet { tableView.reloadData() }
}

private func updateModel() {
    self.offers = offers.filter { offer -> Bool in
        return offer.fromDate >= Date()
    }.sorted { (left, right) -> Bool in
        return left.fromDate < right.fromDate
    }
}
```



Datenstrukturen für TableViews

Datenstruktur	Gruppiert	Reduziert	Sortiert
Array<V>	X	✓	✓
Dictionary<K: AnyHashable, Array<V>>	✓	✓	X
Array<(K, Array<V>)>	✓	✓	✓

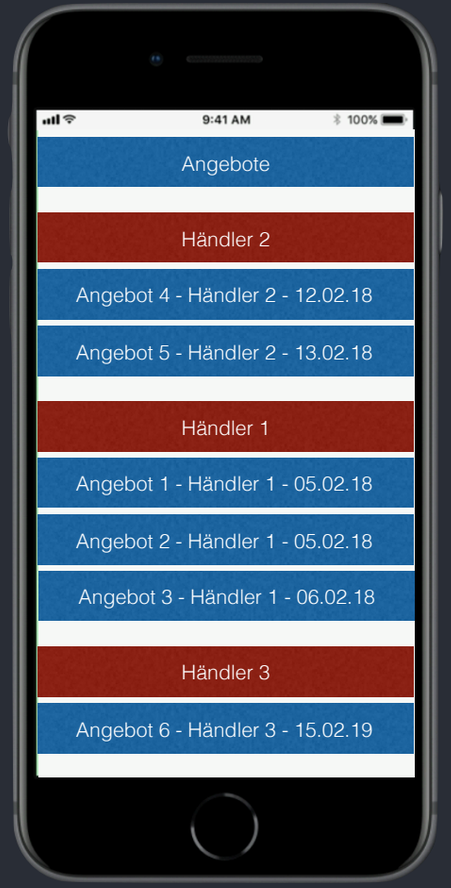
Given

```
struct Offer {
    let id: Int
    let retailerId: Int
    let name: String
    let oldPrice: Double?
    let newPrice: Double?
    let discount: Double?
    let fromDate: Date
    let fromTime: Date
    let toDate: Date
    let toTime: Date
}
```

[K: [V]]

```
var offers: [Int: [Offer]] = [:] {
    didSet { tableView.reloadData() }
}

private func updateModel() {
    self.offers = offers.filter { offer -> Bool in
        return offer.fromDate >= Date()
    }.sorted { (left, right) -> Bool in
        return left.fromDate < right.fromDate
    }.groupBy { offer -> Int in
        return offer.retailerId
    }
}
```



Datenstrukturen für TableViews

Datenstruktur	Gruppiert	Reduziert	Sortiert
Array<V>	X	✓	✓
Dictionary<K: AnyHashable, Array<V>>	✓	✓	X
Array<(K, Array<V>)>	✓	✓	✓

Given

```
struct Offer { var status: OfferStatus }

enum OfferStatus: Hashable {
    case active, valid, upcoming, expired, unknown

    var hashVal: Int {
        switch self {
        case .active, .valid: return 0
        case .upcoming: return 1
        case .expired: return 2
        case .unknown: return 3
        }
    }

    var string: String {
        switch self {
        case .active, .valid: return "Aktuell gültig"
        case .upcoming: return "Bevorstehend"
        case .expired: return "Nicht mehr gültig"
        case .unknown: return "Gültigkeit unbekannt"
        }
    }
}
```

[(K, [V])]

```
var offers: [(k: OfferStatus, v: [Offer])] = [] {
    didSet { tableView.reloadData() }
}

private func updateModel() {
    self.offers = offers.filter { offer -> Bool in
        return offer.status != .expired
    }.sorted { (left, right) -> Bool in
        return left.fromDate < right.fromDate
    }.groupBy { offer -> OfferStatus in
        return offer.status
    }.sorted { (left, right) -> Bool in
        return left.k.hashVal < right.k.hashVal
    }
}
```

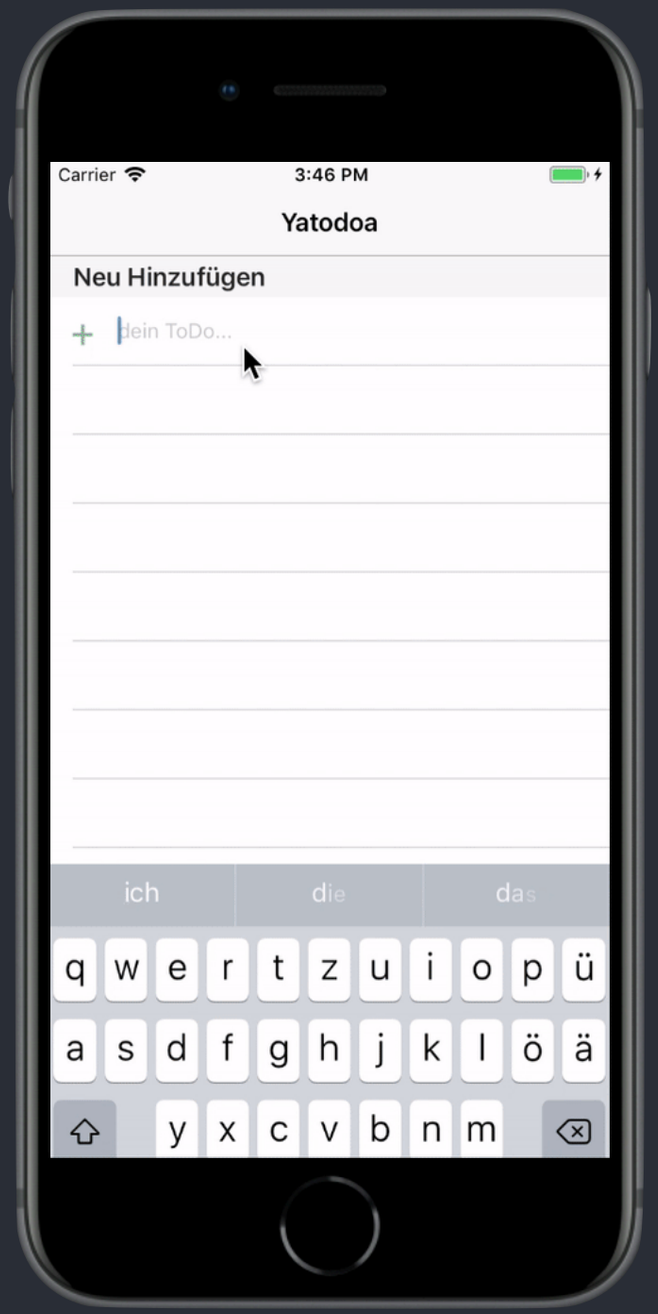
Heute

TableView
UITableViewController
TableViews und MVC
Datenstrukturen für TableViews

Demo
Assignment

Yatodoa - Demo

- UITableViewController
  - UITableViewCells und Custom-Cells
  - UITableView-Delegate und -DataSource
  - UITableView Animationen
  - UITableView Trailing Gesten
- Arrays, Dictionaries und Tuple
- High-Order-Functions
- Enums und Algebra Design
- Computed Properties



Heute

TableView
UITableViewController
TableViews und MVC
Datenstrukturen für TableViews

Demo
Assignment

Yatodoa - Assignment

1. Dritte Section namens “Erledigt”
    - Austausch zwischen “Offen” und “Erledigt”, ohne Animationen!
    - Anpassungen an TodoType und allen DataSource u. Delegate Funktionen
    - Vergegenwärtigt euch, wie “viel” ihr hierfür anpassen müsst
2. Swipe von Rechts zum Löschen eines Todos (siehe Demo davor)
    - Es dürften nur Todos vom Typ .open gelöscht werden
    - Funktion tableView(editingStyle: forRowAt:) in der Doku nachschauen
    - Zunächst nur mit reloadData (ohne Animation) implementieren
    - Danach ruhig mit tableView.deleteRowAtIndexPath arbeiten.
      Achtung: Was passiert, wenn die letzte Zeile gelöscht wird?
- Sonstige Änderungen und Verbesserungen sind Willkommen
- Bis zum 28:11, 13:59 Uhr per Pull-Request einreichen