Full Stack iOS Entwicklung mit Swift

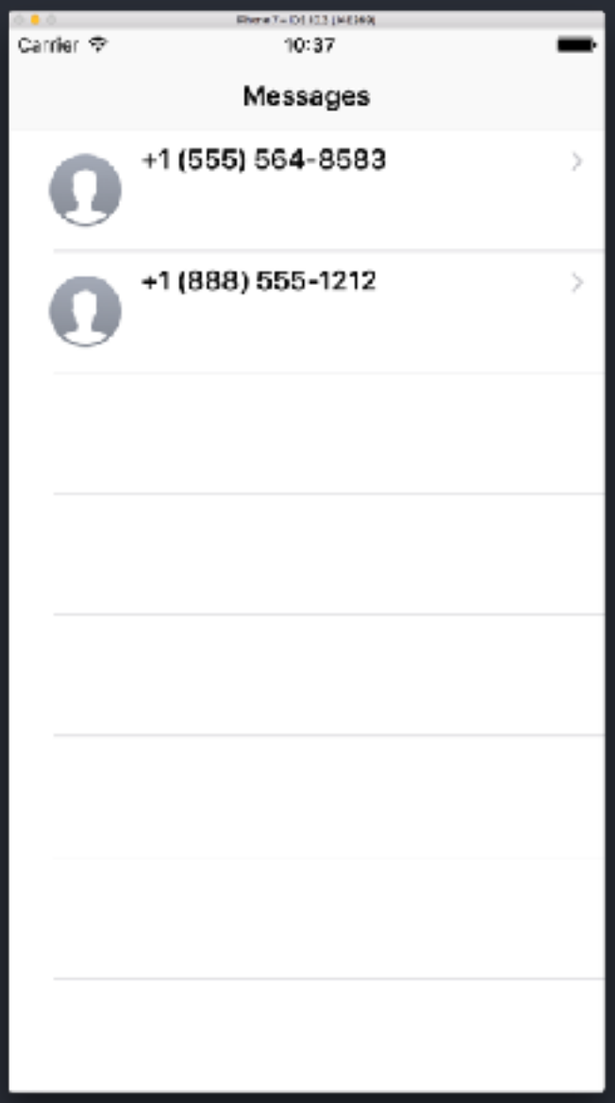
WPF im MIM - SS 17

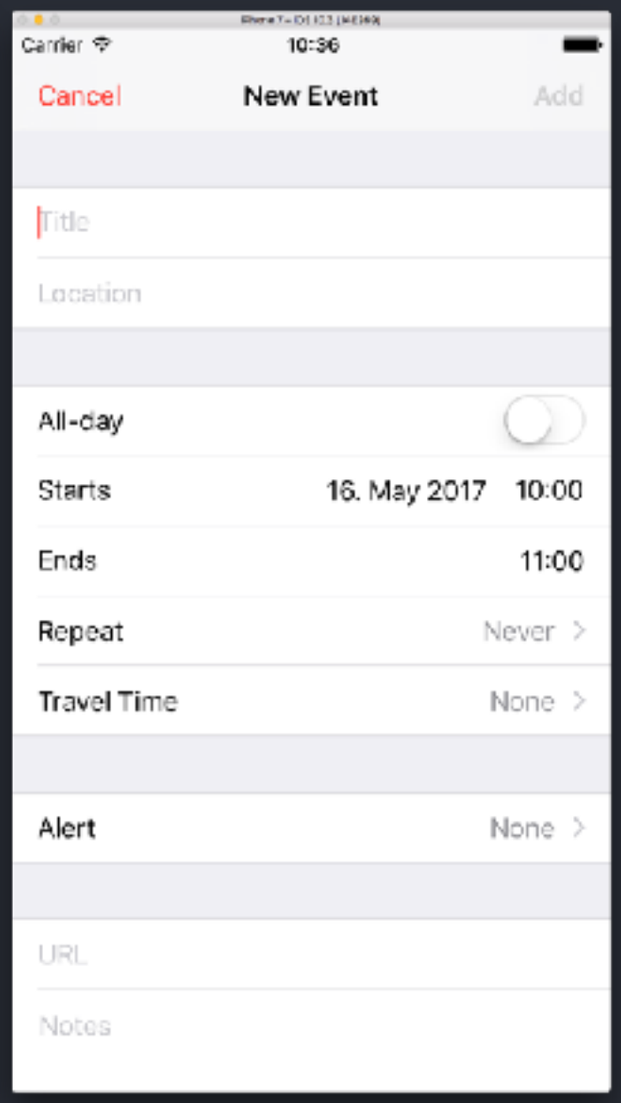
Alexander Dobrynin M.Sc.

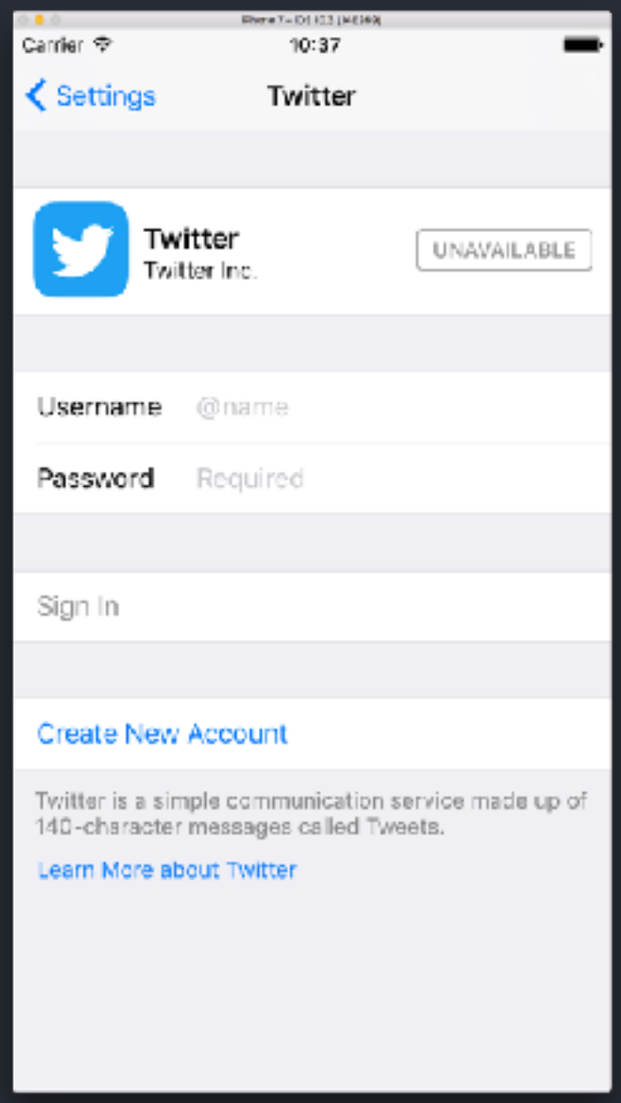
Heute

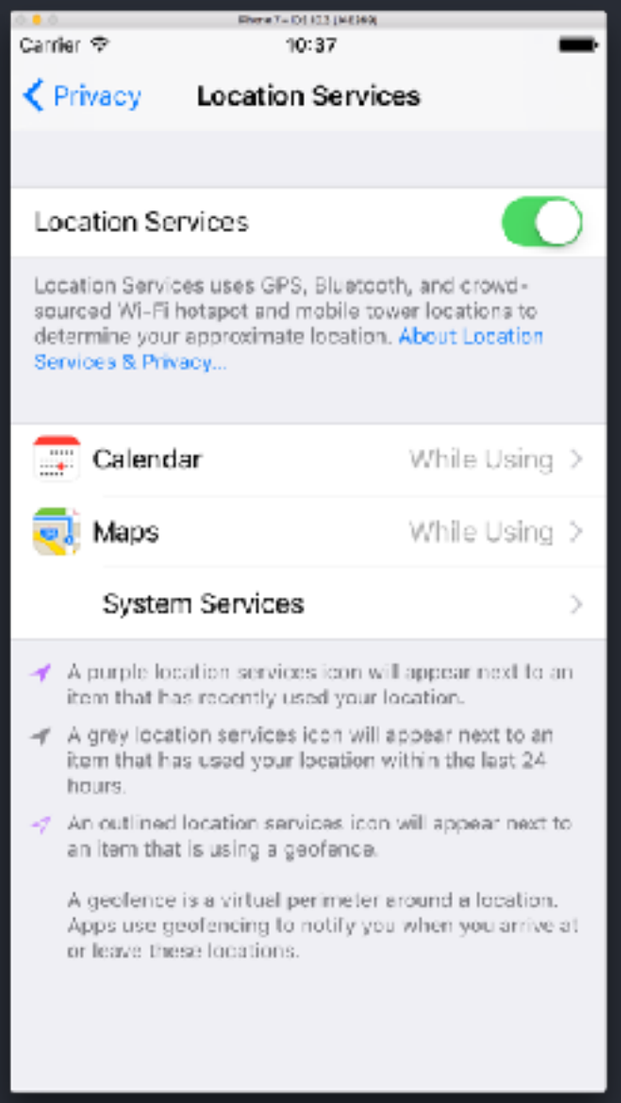
UITableView, Multithreading und Storage

UITableViewController

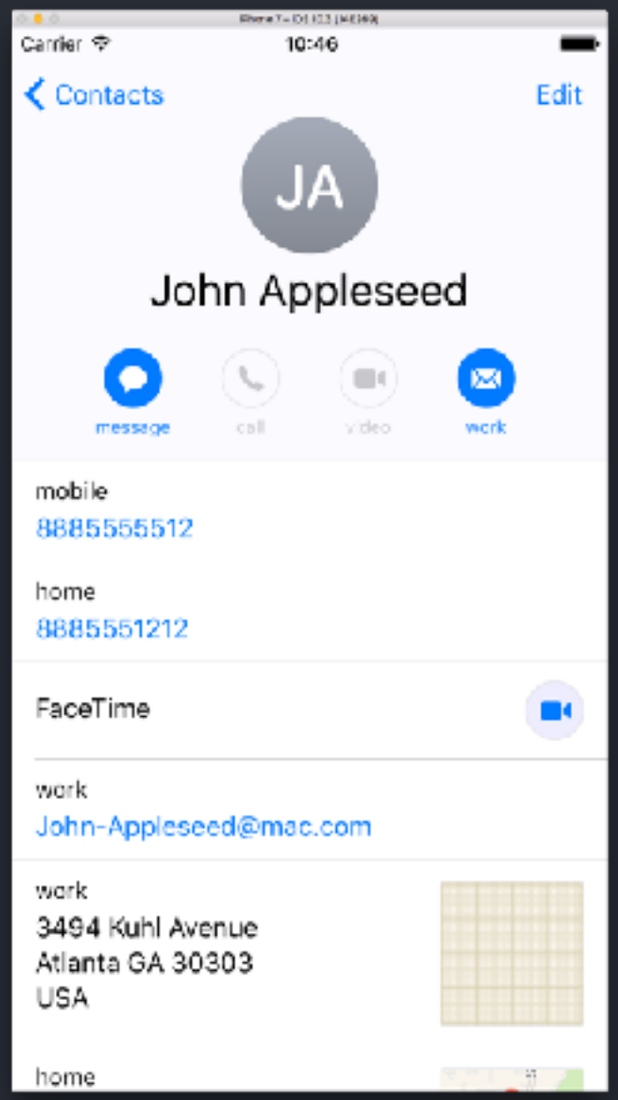


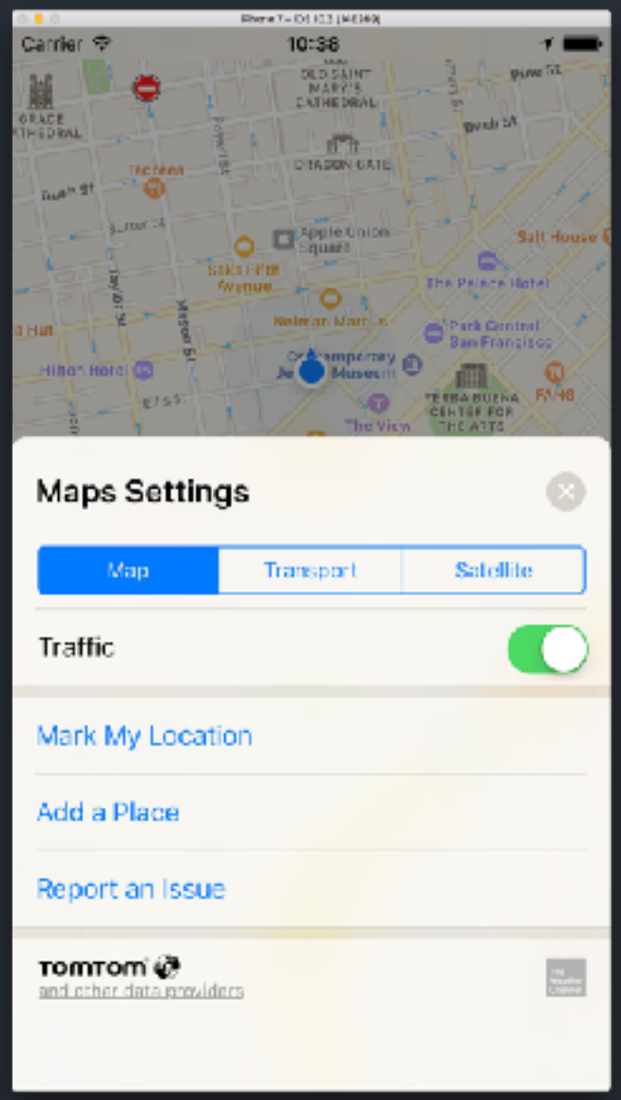


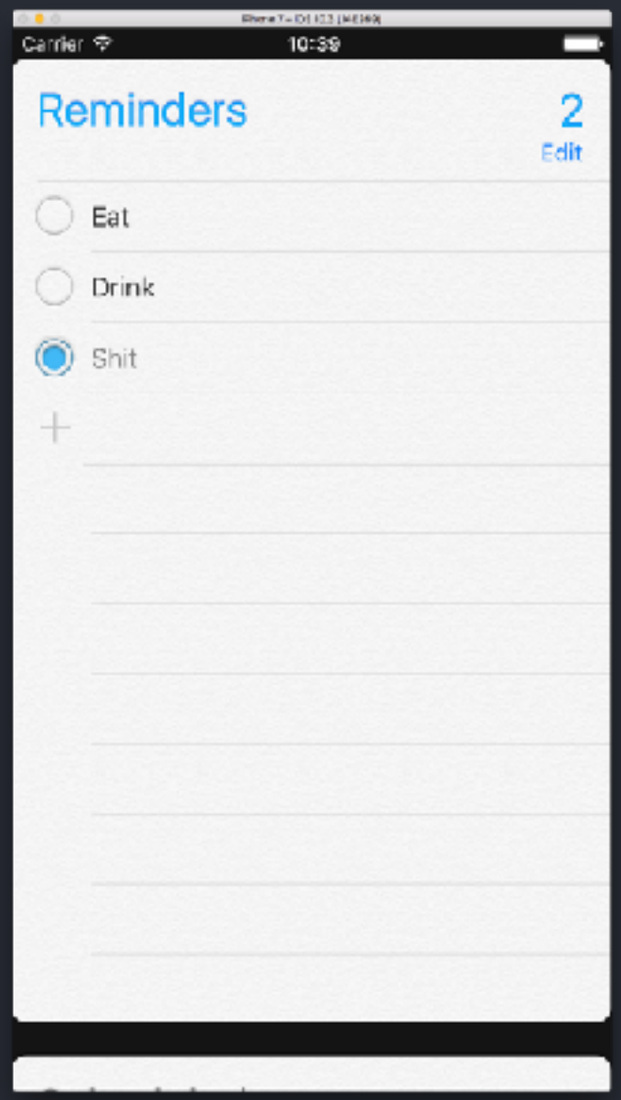


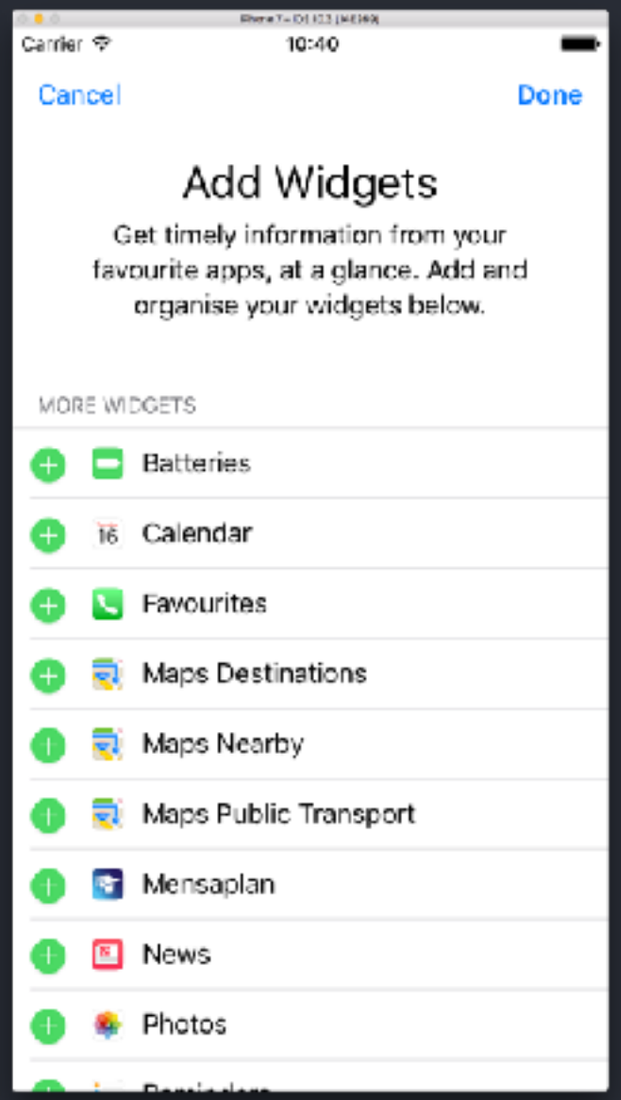


UITableViewController





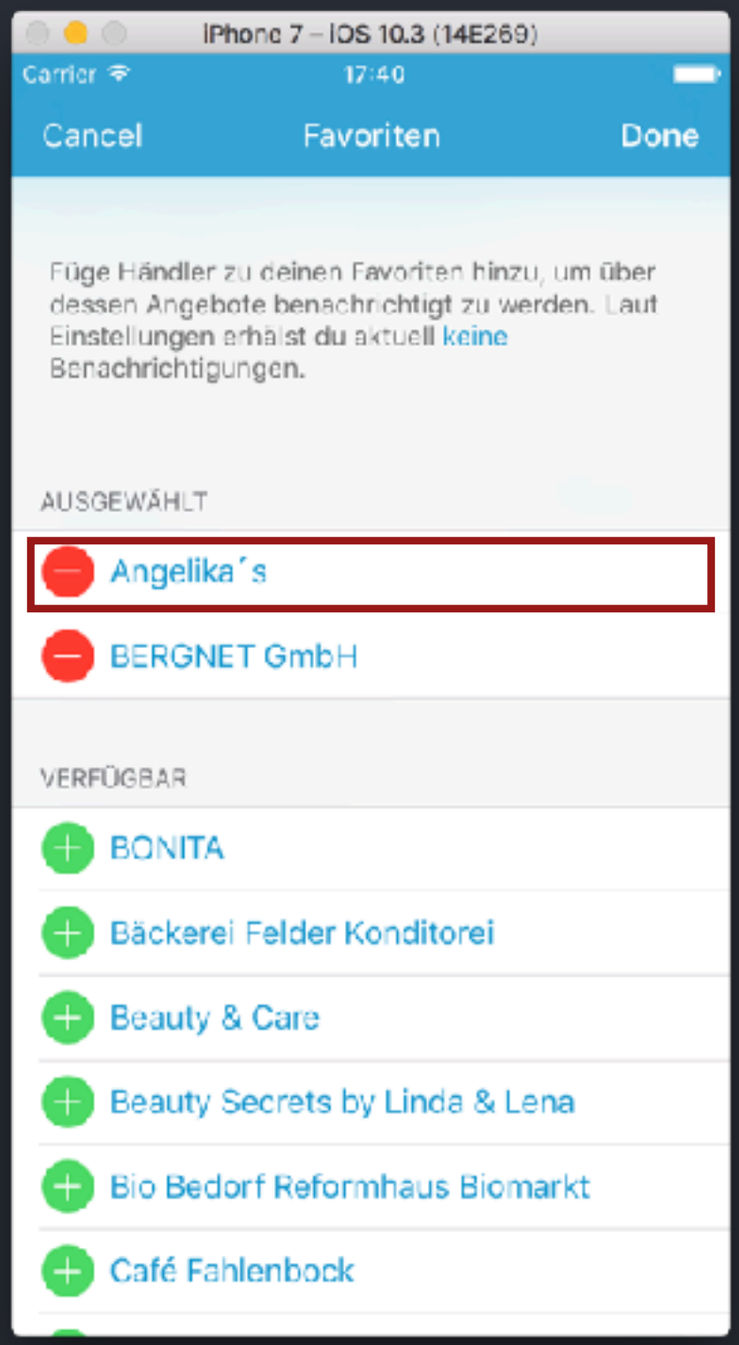




UITableViewController

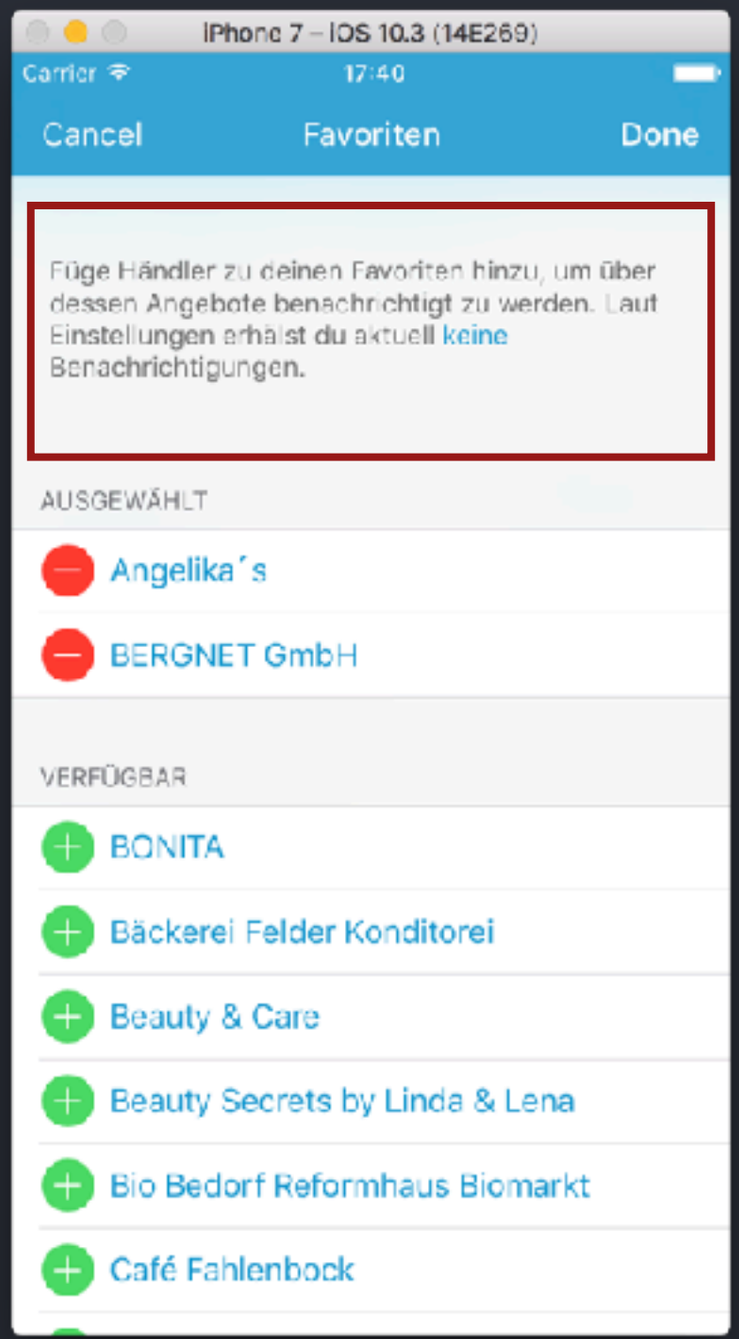
- TableViews werden verwendet, um große Mengen an Daten zu präsentieren
- Viele TableViews werden auch für bestimmte Zwecke missbraucht
- Es gibt statische und dynamische TableViews
- Dynamische TableViews können nichts ohne ihre DataSource
- Interaktionen mit TableViews haben keinen Effekt ohne ihr Delegate
- Eine TableView kann ein Header- und ein FooterView haben
- TableViews haben zwei unterschiedliche Styles, nämlich .grouped und .plain
- Der Inhalt einer TableView sind TableViewCells
- TableViews gruppieren die Cells in Sections und Rows
- Jede Section kann ein Footer und ein Header haben

UITableViewController



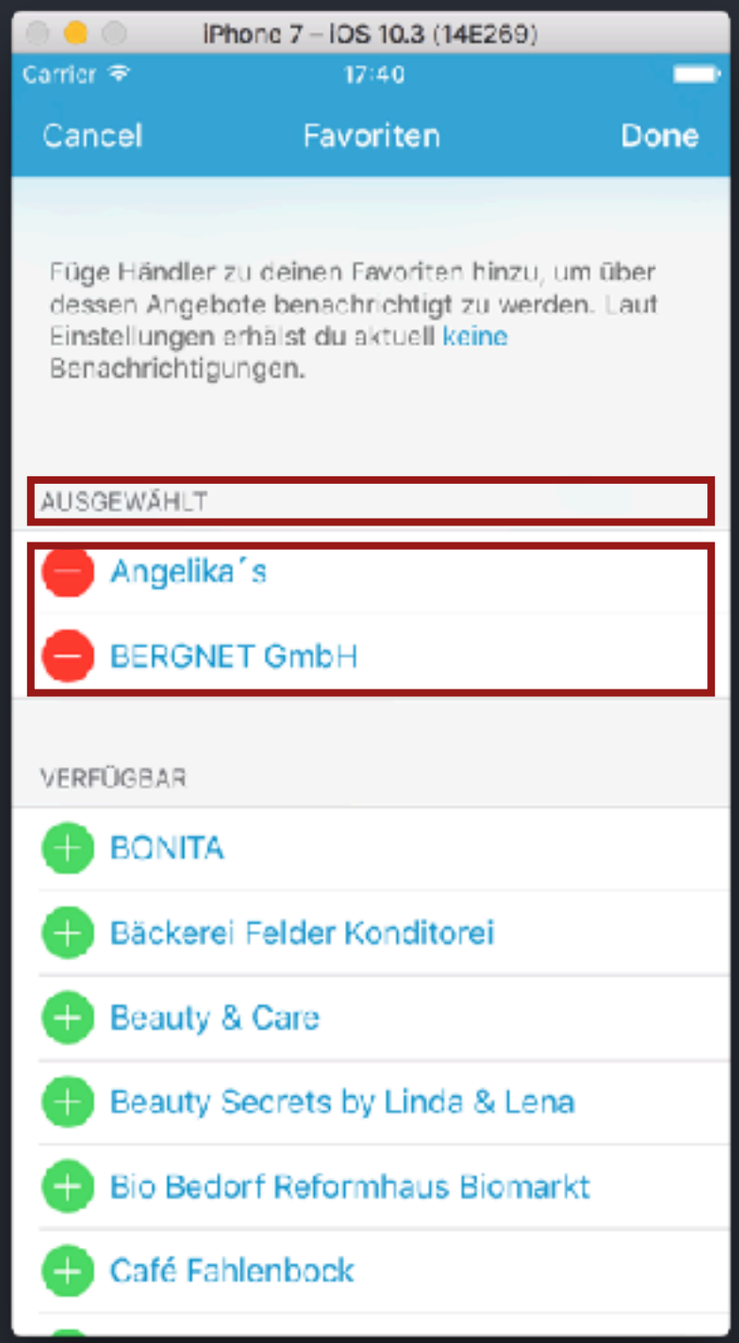
Cell

UITableViewController



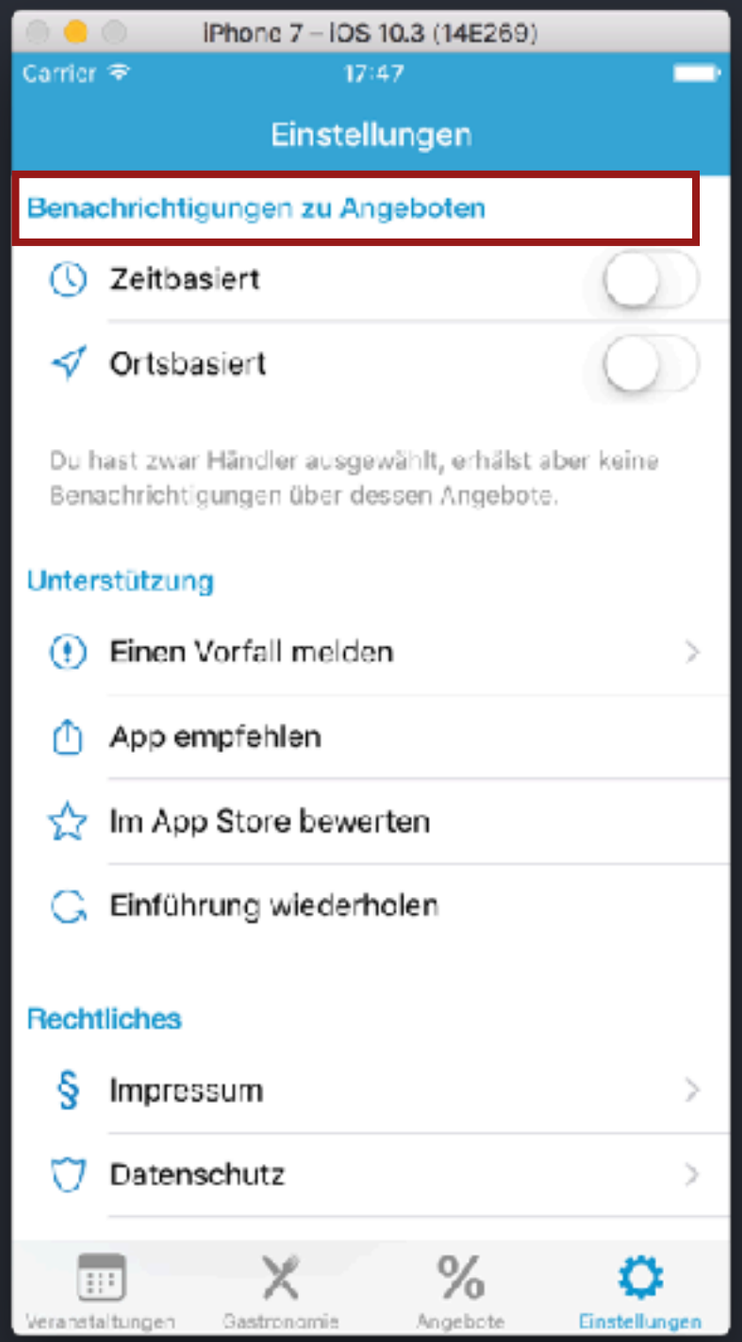
TableView's
HeaderView

UITableViewController



Section - Grouped Rows

UITableViewController

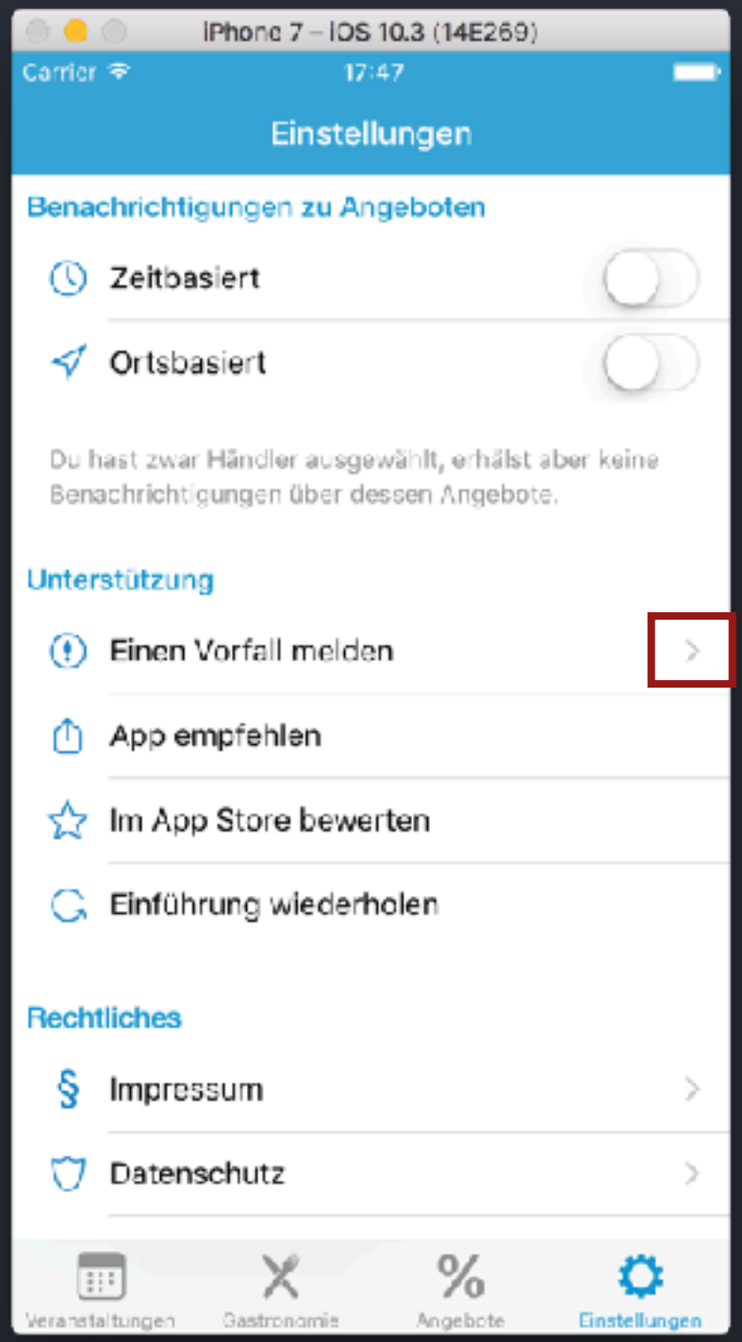


Section - Plain

UITableViewController

Section's Footer

UITableViewController

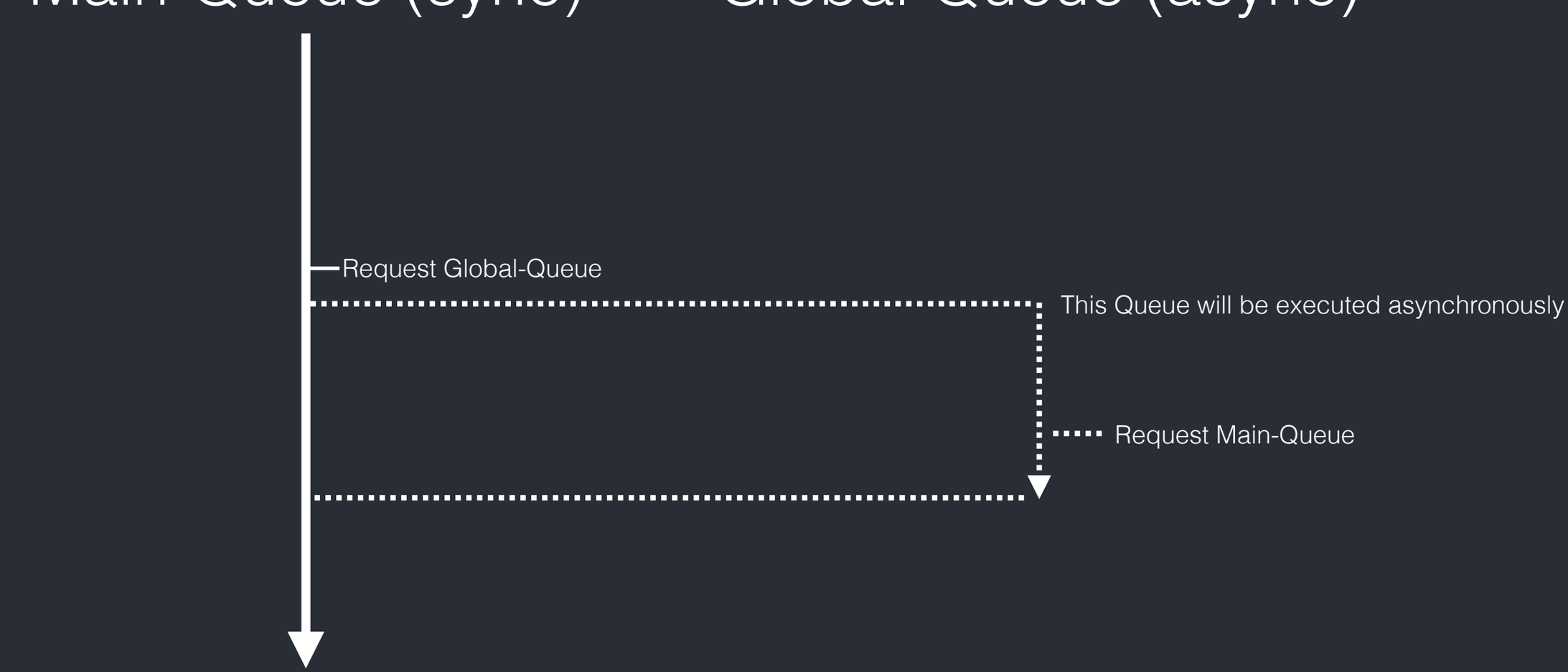


Disclosure Indicator

Multithreading

- Ein Main-UI-Thread, welcher Touch- und sonstige Events verarbeitet und der **einzige Thread ist, welcher mit der UI arbeiten sollte**
- Mobile Geräte unterliegen eingeschränkten Ressourcen
- Dennoch besteht der Bedarf Daten von Disk zu lesen, aus dem Internet zu laden oder grundsätzlich große Datenmengen im Hintergrund zu verarbeiten
- Multithreading in iOS basiert auf Funktionen, die auf unterschiedliche Queues gelegt und ausgeführt werden
- Die einfachste Verwendung von Multithreading in iOS ist Grand-Central-Dispatch (GCD), wobei es noch OperationQueue mit Operation's gibt, dessen API mächtiger ist
- Unabhängig davon existiert eine Main-Queue, auf der die "Hauptarbeit" ausgeführt wird
- Für alle nebenläufigen Aufgaben stehen Global-Queues zur Verfügung
- Unter GCD wird eine Global-Queue mit einem DispatchQoS (Quality of Service) angefragt, welcher die Priorität bestimmt
- DispatchQoS ist ein Enum mit .userInteractive, .userInitiated, .background und .utility als Cases (Priorität absteigend)
- Auf die jeweilige Global-Queue wird eine Funktion gelegt, die gemäß dem QoS in einem Background-Thread ausgeführt wird
- Sobald die Funktionen ausgeführt wurde, kann die Main-Queue angefragt werden, um das Ergebnis auf dem Main-Thread weiter zu verarbeiten
- Der Ablauf mit GCD sollte immer das folgende Pattern haben: **Request Global-Queue -> off main thread, do something asynchronously -> Request Main-Queue -> continue synchronously**

Multithreading



Multithreading

```swift
// request global queue wich is user initiated
DispatchQueue.global(qos: .userInitiated).async {
    // we are off the main thread now. this code block will be executed asynchronously

    // okay, iam done here. ready to go on main thread
    DispatchQueue.main.async {
        // we are back on main thread. lets continue by updating the UI e.g.
    }
}
```

Multithreading

```
print("1")
// request global queue wich is user initiated
DispatchQueue.global(qos: .userInitiated).async {
    // we are off the main thread now. this code block will be executed asynchronously
    print("2")
    // okay, iam done here. ready to go on main thread
    print("3")
    DispatchQueue.main.async {
        // we are back on main thread. lets continue by updating the UI e.g.
        print("4")
    }
}
print("5")


prints

“1”
“5”
this could take some time …
“2”
“3”
yep, this too
“4”
```

Multithreading

```
print(Thread.isMainThread)

DispatchQueue.global(qos: .userInitiated).async {
    print(Thread.isMainThread)

    DispatchQueue.main.async {
        print(Thread.isMainThread)
    }
}

print(Thread.isMainThread)
```

Multithreading

```
print(Thread.isMainThread) // “true”

DispatchQueue.global(qos: .userInitiated).async {
    print(Thread.isMainThread) // “false”

    DispatchQueue.main.async {
        print(Thread.isMainThread) // “true”
    }
}

print(Thread.isMainThread) // “true”
```

Storage

- UserDefaults ist ein Key-Value Store für simple Benutzer-Einstellungen, wobei alle Values sog. Property-List Typen sein müssen
- Mithilfe von NSCoding können beliebige NSObjects zu Property-List Typen werden
- Anschließend können beliebige NSObjects, die konform zu NSCoding sind, in Data serialisiert und auf Disk oder in den UserDefaults persistiert werden (und vice versa)
- Hat das Datenmodell sinnvolle Relationen, die auch verwendet werden, eignet sich die Verwendung von SQLite
- SQLite ist eine C-API, weshalb es viele 3rd-Party-Libraries gibt, die einen typisierten Wrapper um SQLite anbieten
- Die häufigste (und von Apple empfohlene) Datenbank ist CoreData
- CoreData ist ein Object-Relational-Mapper (ORM) und abstrahiert dadurch auch um die SQLite API
- Zudem ist CoreData sehr sehr mächtig und dadurch auch komplex, schwierig zu verstehen und zu debuggen. Des Weiteren passiert sehr viel unter der Haube, was man selbst nicht kontrollieren kann
- Allerdings gibt es viel Geschenk wenn man CoreData verwendet, wie beispielsweise dedizierte TableViewController, welche die Anfragen an CoreData (Model) mit der UI (TableView) automatisch (und animiert) in Synchronisation halten

Demo - LWM

UITableViewController, UITableViewDataSource, UITableViewDelegate, UITableViewCell
Detail-Segues, URLSession, NSCoding-Storage, escaping Closures, Result