

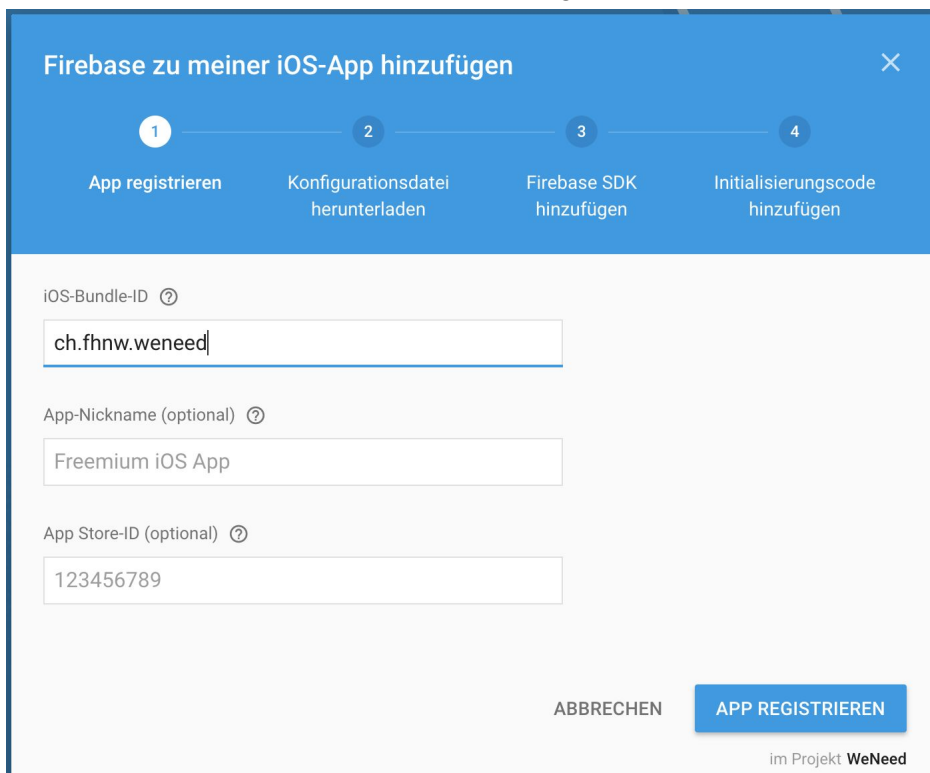
WeNeed App iOS

1. Neues iOS Projekt erstellen

- a. Single View Application
- b. Eindeutigen BundleIdentifier vergeben ("ch.fhnw.weneed")

2. Auf Firebase ein neues iOS Projekt erstellen

- a. Projekt auf <https://console.firebase.google.com/> erstellen (ggf. schon aus Ionic Aufgabe vorhanden)
- b. iOS hinzufügen "Firebase zu meiner iOS-App hinzufügen"
- c. Bundle Identifier wie in Schritt 1 vergeben setzen



The screenshot shows the 'Add Firebase to my iOS app' dialog in the Firebase console. It features a progress bar at the top with four steps: 1. App registrieren (selected), 2. Konfigurationsdatei herunterladen, 3. Firebase SDK hinzufügen, and 4. Initialisierungscode hinzufügen. Below the progress bar, there are three input fields: 'iOS-Bundle-ID' with the value 'ch.fhnw.weneed', 'App-Nickname (optional)' with the value 'Freemium iOS App', and 'App Store-ID (optional)' with the value '123456789'. At the bottom right, there are two buttons: 'ABBRECHEN' and 'APP REGISTRIEREN'. The text 'im Projekt WeNeed' is visible at the bottom right corner of the dialog.

Firestore zu meiner iOS-App hinzufügen

1 App registrieren 2 Konfigurationsdatei herunterladen 3 Firebase SDK hinzufügen 4 Initialisierungscode hinzufügen

iOS-Bundle-ID ⓘ
ch.fhnw.weneed

App-Nickname (optional) ⓘ
Freemium iOS App

App Store-ID (optional) ⓘ
123456789

ABBRECHEN APP REGISTRIEREN

im Projekt WeNeed

- d. Datei GoogleService-Info.plist herunterladen
- e. Realtime-Database in Firebase hinzufügen (öffentlich)

3.) CocoaPods installieren

- Xcode Projekt schliessen und im Terminal in Projektverzeichnis wechseln
- Dependency Manager CocoaPods installieren: `$sudo gem install cocoapods`
- `$pod init` im Ordner der ProjektDatei aufrufen
- Alternativ die Datei Podfile anlegen und diesen Inhalt hinein kopieren

```
platform :ios, '11.0'
use_frameworks!

target 'weneed' do
  pod 'Firebase/Core'
  pod 'Firebase/Database'
end
```

- `$pod install` aufrufen -> Es wird eine *.xcworkspace Datei erstellt.
Künftig nur noch diese Datei öffnen, nicht mehr die *.xcodeproj Datei

4.) iOS Projekt mit Firebase verknüpfen

- Workspace Datei öffnen
- Heruntergeladene Datei GoogleService-Info.plist in das Projekt importieren


Destination: ☒ Copy items if needed

Added folders: ☒ Create groups

☐ Create folder references

Add to targets: ☒  weneed

☐  weneedTests

☐  weneedUITests

- Datei AppDelegate.swift anpassen

```
import UIKit
import Firebase

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?


    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        FirebaseApp.configure()

        return true
    }
}
```

- App im Simulator aufrufen → Soll fehlerfrei starten

5.) Model erstellen

a. Klasse Item.swift erstellen

```
import Foundation
import Firebase

class Item{

    var id: String?
    var name: String = ""
    var isDone = false

    init(name: String){
        self.name = name
    }

    func asDictionary()->[String:AnyObject]{
        return ["name": name as AnyObject,
                "isDone": isDone ? 1 as AnyObject: 0 as AnyObject]
    }

    static func parse(_ dataItem: DataSnapshot)->Item?{
        if let dic = dataItem.value as? Dictionary<String, AnyObject>{
            let item = Item(name: dic["name"] as! String)
            item.isDone = dic["isDone"] as! Bool
            item.id = dataItem.key
            return item
        }
        return nil
    }
}
```

b. Klasse ItemModel.swift erstellen

```
import Foundation
import Firebase

class ItemModel{

    var ref: DatabaseReference!
    var items = [Item]()

    init() {
        configureDatabase()
        addListeners()
    }

    deinit {
        ref.removeAllObservers()
    }

    private func configureDatabase() {
        FirebaseApp.configure()
        ref = Database.database().reference()
    }
}
```

```

private func addListeners(){
    ref.observe(.value) { (snapshot) in
        self.updateItems(snapshot: snapshot)
    }
}

private func updateItems(snapshot: DataSnapshot){
    items.removeAll()
    if let dataSnapshots = snapshot.children.allObjects as? [DataSnapshot]{
        for dataSnapshot in dataSnapshots{
            if let snaps = dataSnapshot.children.allObjects as?
                [DataSnapshot]{
                for snap in snaps {
                    if let item = Item.parse(snap){
                        self.items.append(item)
                    }
                }
            }
        }
    }
}

func addItem(name: String){
    let item = Item.init(name: name)
    self.ref.child("items").childByAutoId().setValue(item.asDictionary())
}

func deleteItem(id: String){
    self.ref.child("items").child(id).removeValue()
}

func markItemAsDone(item:Item){
    item.isDone = false
}

self.ref.child("items").child(item.id!).updateChildValues(item.asDictionary())
}

func updateItem(item: Item, name: String, isDone: Bool){
    item.name = name
    item.isDone = isDone
}

self.ref.child("items").child(item.id!).updateChildValues(item.asDictionary())
}
}

```

- c. AppDelegate.swift von Firebase-Referenzen befreien, da dies nun im Model passiert
- d. ViewController.swift anpassen

```

class ViewController: UIViewController {

    let model = ItemModel()

    override func viewDidLoad() {
        super.viewDidLoad()
        model.addItem(name: "Foo")
    }

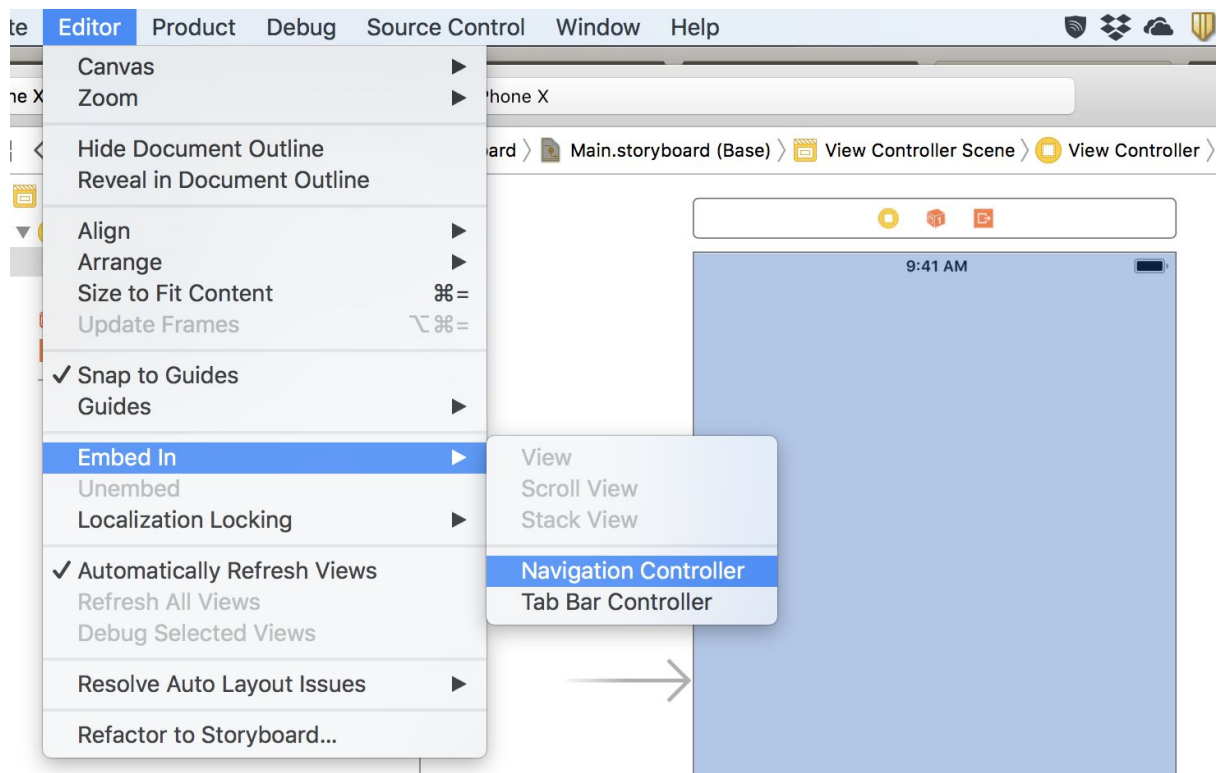
}

```

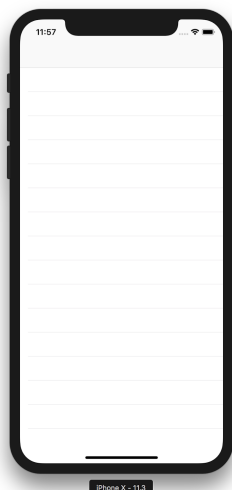
- e. App ausführen und prüfen, dass beim Start der App Daten in die DB gespeichert werden. Hierzu im Browser den DB-Inhalt kontrollieren und per Debugger / print() Inhalte von Objekten prüfen.

6.) UI ergänzen

- a. Main.storyboard öffnen und ViewController selektieren. Im Menü Editor > Embed in > Navigation Controller selektieren



- b. Table View aus Object library auf den ViewController ziehen und mit Constraints ergänzen, sodass es keinen Abstand zum ParentView hat.
- c. App ausführen. Eine leere Tabelle sollte zu sehen sein



- d. Bar Button Item aus der Object Library auf die Navigationbar ziehen
- e. Bar Button Item das System Item Add in Attribute Inspector zuweisen
- f. Table View konfigurieren:
 - a. Style: Grouped
 - b. Prototype Cells: 1
- g. UITableViewCell konfigurieren:
 - a. Identifier: "ItemCell"
 - b. Style: Basic

7.) Outlets verknüpfen

- a. Assistent Editor öffnen und Verknüpfung für TableView erstellen
- b. Action für BarButton Item erstellen

```
@IBOutlet weak var tableView: UITableView!
@IBAction func addItemTapped(_ sender: Any) {
}
```

8.) UITableViewDatasource und UITableViewDelegate implementieren

```
extension ViewController: UITableViewDataSource{

    func tableView(_ tableView: UITableView, numberOfRowsInSectionSection section: Int) -> Int {
        return model.items.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "ItemCell", for: indexPath)
        cell.textLabel?.text = model.items[indexPath.row].name
        return cell
    }
}

extension ViewController: UITableViewDelegate{

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        print("row \(indexPath) selected")
    }
}
```

- a. Im Model eine Referenz var delegate: ViewController ergänzen und diese vom ViewController aus setzen
- b. Die Methode refreshUI() im ViewController ergänzen und die tableView mithilfe der Funktion reloadData() updaten
- c. Bei Updates im Model die Methode refreshUI() im ViewController aufrufen
- d. App starten. "Foo"-Objekte aus unserem Test oben sollten jetzt sichtbar werden.
- e. Nein? Hinweis: Weiss die TableView, wer Ihr Delegate und DataSource ist?

9.) AlertViewController ergänzen um Daten einzugeben

- a. Methode im ViewController ergänzen

```
func requestNewItem(){

    let alert = UIAlertController(title: "Add new item", message: nil, preferredStyle: .alert)
    alert.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: nil))
    alert.addTextField(configurationHandler: { textField in
        textField.placeholder = "Item name"
    })

    alert.addAction(UIAlertAction(title: "Add", style: .default, handler: { action in

        if let name = alert.textFields?.first?.text {
            if name.count > 0{
                self.model.addItem(name: name)
            }
        }
    })))

    self.present(alert, animated: true)
}
```

- b. Mit der zuvor erstellten IBAction vom + Button verbinden
c. Testen, dass das Hinzufügen funktioniert

10.) Daten als erledigt markieren

- a. Methode zum Abfüllen der Zelle anpassen und einen Haken zeichnen

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "ItemCell", for: indexPath)
    let item = model.items[indexPath.row]
    cell.textLabel?.text = item.name
    cell.accessoryType = item.isDone ? .checkmark : .none
    return cell
}
```

- b. Beim Antippen einer Zelle den Status invertieren

```
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    let item = self.model.items[indexPath.row]
    model.updateItem(item: item, name: item.name, isDone: !item.isDone)
}
```

10.) Daten editieren

a. Alert zum Editieren aufrufen

```
func editItem(item: Item){
    let alert = UIAlertController(title: "Edit item", message: nil, preferredStyle: .alert)
    alert.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: nil))
    alert.addTextField(configurationHandler: { textField in
        textField.placeholder = "Item name"
        textField.text = item.name
    })
    alert.addAction(UIAlertAction(title: "Save", style: .default, handler: { action in
        if let name = alert.textFields?.first?.text {
            if name.count > 0{
                self.model.updateItem(item: item, name: name, isDone: item.isDone)
            }
        }
    })))
    self.present(alert, animated: true)
}
```

b. Methode implementieren, die beim Swipe von Links nach Rechts aufgerufen wird

```
func tableView(_ tableView: UITableView,
               leadingSwipeActionsConfigurationForRowAt indexPath: IndexPath) -> UISwipeActionsConfiguration?
{
    let editAction = UIContextualAction(style: .normal, title: "Edit", handler: { (ac: UIContextualAction,
        view: UIView, success: (Bool) -> Void) in
        let item = self.model.items[indexPath.row]
        self.editItem(item: item)
    })
    editAction.backgroundColor = .blue
    return UISwipeActionsConfiguration(actions: [editAction])
}
```

11.) Daten löschen

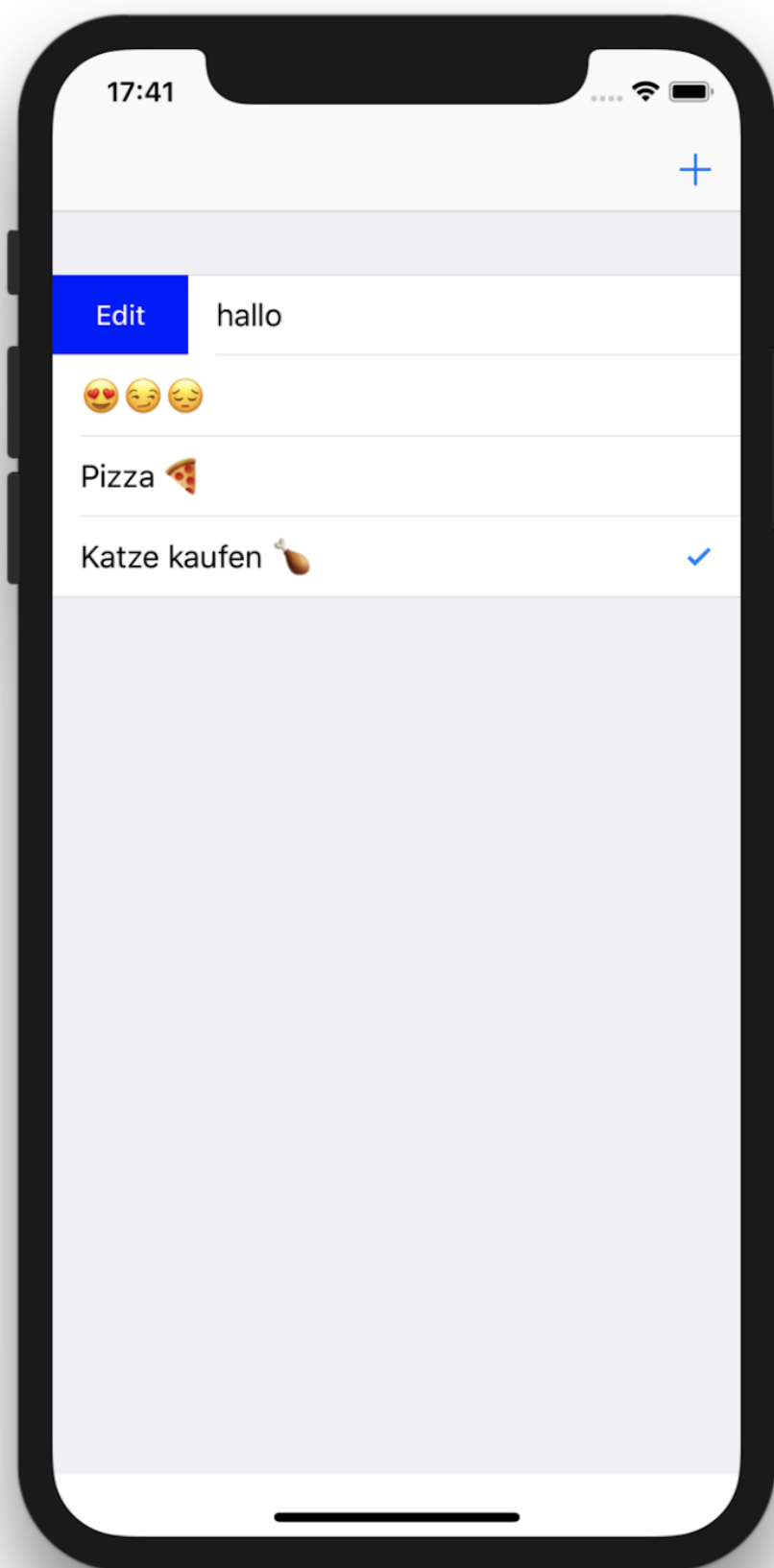
a. Methode implementieren, die beim Swipe von Rechts nach Links aufgerufen wird

```
func tableView(_ tableView: UITableView,
               trailingSwipeActionsConfigurationForRowAt indexPath: IndexPath) -> UISwipeActionsConfiguration?
{
    let deleteAction = UIContextualAction(style: .normal, title: "Delete", handler: { (ac: UIContextualAction,
        view: UIView, success: (Bool) -> Void) in
        if let itemID = self.model.items[indexPath.row].id{
            self.model.deleteItem(id: itemID)
        }
    })
    deleteAction.backgroundColor = .red
    return UISwipeActionsConfiguration(actions: [deleteAction])
}
```

12.) Make it beautiful

Ideen:

- App Icon
- Farben
- Fonts
- ...



iPhone X - 11.3