

Abschlussprüfung Winter 2019

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Museums App

Programmieren von einer “Museum App” für iPhone und iPad

Abgabetermin: 05.12.2019

Prüfungsbewerber:

Laurent Brusa
Tegnerstr 2
10435 Berlin

Ausbildungsbetrieb:

Studio Tim Deussen
Kreuzigerstraße 10
10247 Berlin

Inhaltsverzeichnis

1 - Einleitung	1
1.1 Projektbeschreibung	1
1.2 Projektziele	1
1.3 Projektumfeld	2
1.4 Projektabgrenzung	2
2 - Projektplanung	2
2.1 Projektphasen	2
2.3 Entwicklungsprozess	3
2.4 Kostenplanung	3
2.2 Software und Hardware	4
3 - Entwurfsphase	4
3.1 UX/UI Design	4
3.2 Erster Entwurf - Prototyping	4
3.3 Entwurf mit Sketch	5
3.4 MVC Design Pattern	6
3.5 Die Modellschicht	7
3.5.1 JSON	7
3.5.2 Data Struktur in Model.swift	8
3.6 Der View Controller	9
3.7 Die View	9
3.7.1 Autolayout	9
3.7.2 LaunchScreen.storyboard	10
3.8 Delegation Design Pattern	10
3.9 Beispiel von MVC und Delegation anhand den UISearchController	10
4 - Implementierung	11
4.1 Xcode	11
4.2 Zugriffskontrolle	11
4.3 GIT	12
4.4 Der Assets.xcassets Folder	12
4.5 Suchen	12
4.6 ViewController.swift	12
5 - Unit Testing	13
5.1 Best Practices für Tests - FIRST	13
5.2 Mit XCTAssert Modelle testen.	14
5.3 Deployment und Übergabe	15
6 - Fazit	15

Museums App
Programmieren von einer "Museum App" für iPhone und iPad

Literaturverzeichnis	15
Anhang	I
A1.1 Detaillierte Zeitplanung: Tortendiagramm	I
A1.2 Detaillierte Zeitplanung	II
A1.3 Detaillierte Auflistung von Ressourcen	III
A2 - Wireframes Entwürfe	IV
A2.1 List View:	IV
A2.2 Collection View:	V
A2.3 Detailed View:	VI
A3 - Sketch Entwurf	VII
A3.1 Table View	VII
A3.2 Entwurf in Sketch - Collection View	VIII
A3.3 Entwurf in Sketch - Logo	IX
A3.4 Entwurf in Sketch - Detail View	X
A4 - Xcode Screenshots	XI
A4.1 AppIcon in Xcode	XI
A4.2 models.JSON	XII
A4.3 main.storyboard	XIII
A4.4 Constraints Beispiel- Cell View Prototype	XIV
A4.5 LaunchScreen.storyboard	XV
A5 - Quellcode	XVI
A5.1 Quellcode - ViewController.swift	XVI
A5.2 Quellcode - DetailViewController.swift	XX
A5.3 Quellcode - ModelCell.swift	XXI
A5.4 Quellcode - ModelCell.swift	XXII
A5.5 Git - .gitignore Datei in Terminal erstellen	XXII

1 - Einleitung

Ausbildungsbetrieb ist das Studio Tim Deussen¹ in Berlin.

Das Studio realisiert vielfältige Produktionen im Bereich der digitalen Videoproduktion, 360 Grad Aufnahmen und 3D Animationen, sowie VR und AR Erlebnisse, die Online und auf Messen gezeigt wurden.

Die Erfahrungen mit diesem breiten Bogen der digitalen Medienproduktion spannen eine Brücke zu digitalen Inhalten und Apps für Android und iOS, die für Konzerne wie Wuerth Elektronik, Steute oder Otis produziert wurden.

Die folgende Projektdokumentation schildert den Ablauf des IHK-Abschlussprojektes, das im Rahmen der Ausbildung zur Fachinformatiker Fachrichtung Anwendungsentwicklung durchgeführt wurde.

Der Auftraggeber ist 3DPC², eine internationale Plattform für Pioniere im 3D-Druck und ein internationaler Wettbewerb für alle Enthusiasten, die sich mit neuen Technologien befassen und sich dabei dem 3D-Druck bedienen um ihre Projekte voranzutreiben. Jedes Jahr hält die 3DPC einen Internationalen Design-Wettbewerb für additive Fertigungsverfahren: "Die 3D Pioneers Challenge". Diese wird in enger Kooperation von der Rapid.Tech + FabCon 3.D und d.sign21 im Rahmen der Fachmesse Erfurt seit 2015 ausgeführt.

1.1 Projektbeschreibung

Es wird eine "Museum App" für iPhone und iPad programmiert.

Durch multimedialen Mitteln soll diese App Einblicke und ein tieferes Verständnis in die 3D-Druck Welt ermöglichen. Die 3D Pioneers Challenge wird eine "Pushing Boundaries" Ausstellung und Road-Show veranlassen um aus etwa 32 3D Modellen, die Finalisten von diesem Jahr, zu zeigen.

Die App soll eine suchbare Auflistung der Objekte in einer 3D-Druck Ausstellung darstellen.

1.2 Projektziele

Ziel des Projektes ist eine Begleit-App zur Ausstellung zu entwickeln. Die App soll durch eine grafische Benutzeroberfläche und eigene Datenbank die Exponate von dem Wettbewerb veranschaulichen. Zu jedem Exponat sollte ein Foto, eine kurze Beschreibung und auch ein Video gezeigt werden. Die Besucher werden mehr Informationen über die Exponate und Videos über die Druckprozesse und Verarbeitungstechniken mit denen die Exponate erstellt wurden.

¹ <https://tim-deussen.de>

² <https://www.3dpc.io/de>

1.3 Projektumfeld

Es gibt 32 Exponate (3D Modelle). Zu jedem einzelnen Modell werden die Assets (Beschreibung, Fotos und Videos) von den einzelnen Firmen und Wettbewerber geliefert. Die App muss für iOS mit Swift in der Xcode IDE programmiert werden und wird nicht im App-Store veröffentlicht, sondern lokal für die Ausstellung zur Verfügung gestellt. Dafür werden iPads für die Ausstellung gemietet. Die Sprache der App wird Englisch sein. Auf 12 gemieteten iPads wird die "Pushing Boundaries" Ausstellungs-App vorinstalliert und an die Ausstellung geliefert. Eine suchbare Auflistung der Exponate der 3D-Ausstellung wird angeboten.

1.4 Projektabgrenzung

Apple hat einen relativ aufwendigen Genehmigungsprozess für Apps auf dem Apple Store. Man kann in diesem Fall darauf verzichten. Unser Studio hat eine Apple Developer Lizenz, und damit darf man die App auf eigene Geräte installieren. Das heißt, es wird auf die Genehmigung von Apple verzichtet. Wir werden 12 iPads mieten und die Software nur lokal auf die Geräte installieren. Die App soll selbst contained und ohne Internet Zugang auch problemlos funktionieren. Die Anbindung der Datenbank an einen externen Server ist nicht nötig. Es wird die aktuelle Version von iOS benutzt.

2 - Projektplanung

2.1 Projektphasen

Für die Umsetzung des Projektes wurden 70 Stunden eingeplant. Diese wurden vor Projektbeginn auf verschiedene Phasen verteilt, die während der Softwareentwicklung durchlaufen werden. Eine Übersicht dieser Phasen befindet sich im Anhang ([A1.1 Detaillierte Zeitplanung: Tortendiagramm](#)) und ([A1.2 Detaillierte Zeitplanung](#)).

Projektphase	Geplante Zeit
Vorbesprechung - Analysephase	6 h
Entwurfsphase	8 h
Programmierung und Implementierungsphase	31 h
Testen	8 h
Deployment - Abnahme und Einführung	7 h
Erstellen der Dokumentation	10 h
Gesamt	70 h

Tabelle 1: Zeitplanung

2.3 Entwicklungsprozess

Es gab drei Möglichkeiten um die Daten in der App zu speichern:

1. Die Fotos und Videos können in das lokale Netzwerk geladen werden. Dafür könnte ein Mac mini oder ein kleinerer Server gemietet werden. Er wäre in einem separaten Netzwerk und nicht im Internet. Die Assets werden dann über LAN gestreamt.
2. Die Assets können über das Internet von einem Cloud-Service wie Amazon AWS oder Google über Internet gestreamt werden. Dies wäre preiswert und relativ einfach einzurichten.
3. Alle Assets werden auf dem Gerät installiert und keine externe Streaming-Option verwendet.

Man hat sich für die Option 3 entschieden, weil die Anzahl der in der Ausstellung gezeigten Stücke begrenzt ist und diese sich nicht ändern werden. Es ist also kein dynamisches Update erforderlich. Die Videos sind alle etwa eine Minute lang und können auf jeweils 20 MB komprimiert werden. Es wird 32 Videos und Fotos geben, die insgesamt zusammen bei etwa 650 MB liegen. Die Größe der App bleibt überschaubar. Die App wird nicht im App Store verteilt. Unser Studio hat die Entwickler Lizenz und damit können wir App-IDs generieren und die App auf jedem Gerät installieren. Die ID jedes Geräts wird gleichzeitig von Apple registriert. Die App-ID ist eindeutig und wird mit der Geräte-ID verknüpft.

2.4 Kostenplanung

Der Preis für die Miete des iPads wurde angefragt und beträgt 980 Euro für eine Woche. Die Projektkosten sollen im Folgenden kalkuliert werden. Dafür müssen neben den Personalkosten, auch noch die Aufwendungen für die Ressourcen (Hard- und Software, Büroarbeitsplatz etc.) berücksichtigt werden. Die Kalkulation wird anhand von Stundensätzen durchgeführt. Der Stundensatz eines Auszubildenden beträgt demzufolge 9,50 €. Für die Ressourcennutzung wurde in Absprache mit meinem Vorgesetzten ein Gemeinkostenzuschlagsatz auf den Lohn von 105 % angenommen.

	Mitarbeiter	Zeit	Kosten	Ressourcen	Gesamt
Entwicklungs-kosten	Praktikant	60 h	570 EUR	600 EUR	1170 EUR
Mieten von iPads	Externe Firma		980 EUR		980 EUR
Gesamt					2150 EUR

Tabelle 2: Kostenaufstellung

2.2 Software und Hardware

Die iOS App wird mit der Programmierungssprache Swift in der Xcode IDE programmiert. Diese Software ist kostenfrei, allerdings man braucht eine Developer Lizenz. Sie kostet 100 EUR pro Jahr und wird direkt von Apple erworben. Unser Studio hat bereits eine Lizenz. Somit werden keine weiteren Kosten anfallen.
Eine Auflistung der benutzten Software und Hardware Ressourcen sind im Anhang. ([A1.3 Detaillierte Auflistung von Ressourcen](#))

3 - Entwurfsphase

3.1 UX/UI Design

UX Design bezieht sich auf den Begriff User Experience Design. Darunter versteht man sämtliche Verhaltensweisen, Emotionen und Ansichten einer Person über ein bestimmtes Produkt, System oder Anwendung. Damit erreicht man Benutzerfreundlichkeit, Zugänglichkeit und Attraktivität, die in der Interaktion mit einem Produkt bereitgestellt werden. Es erweitert alle Aspekte eines Produkts oder einer Dienstleistung, wie sie von den Nutzern wahrgenommen werden und hat den Fokus auf die Qualität der User Experience.

Das User Interface ist die Oberfläche, auf der die Interaktion zwischen Mensch und Maschine stattfindet. Das Interface soll im besten Falle intuitiv, effizient und angenehm zu beherrschen sein. Beide Elemente sind für ein Produkt entscheidend und arbeiten eng zusammen.

Die App wurde möglichst einfach, ohne unnötige Popup's oder komplizierte Menüs, gestaltet. In diesem Fall soll das User Interface der App leicht zu bedienen und übersichtlich und einfach zu verstehen sein.

3.2 Erster Entwurf - Prototyping³ ⁴

Um eine gute Gestaltung zu gewährleisten und darauf Feedback zu erhalten, erstellen wir Prototypen. Dafür werden Tools verwendet, die man bereits kennt. Es kann Apple Keynote oder die bevorzugte Textverarbeitung sein oder auch einfach ein Blatt Papier und ein Stift. Paper-Wireframes⁵ in UX Design ist eine Technik um Prototypen zu gestalten, die oft in frühen Phasen des Designprozesses verwendet wird. Dadurch werden Probleme früher erkannt. Das spart viel Zeit, Geld und Energie. Außerdem ist die Kommunikation im Team viel einfacher. Man hat einen visuellen Eindruck und auch nichttechnische Teammitglieder fühlen sich eingebunden und können von Anfang an am kreativen Prozess mitwirken. Wenn ein Feature früh von Worten zu Papier kommt, wird es für viele Interessengruppen "realer".

³ <https://developer.apple.com/videos/play/wwdc2017/818/>

⁴ <https://developer.apple.com/videos/play/wwdc2016/805/>

⁵ <https://blog.marvelapp.com/paper-wireframing-will-make-better-designer/>

Auch für den Auftraggeber können Logos, mit etwas Präzision gezeichnet ein schönes Detail sein. Text wird z.B. durch einigen Morsezeichen-Linien ersetzt. Für das Zeichnen von Fotos sind leere Rechtecke mit diagonalen Linien, die durch sie verlaufen, etwas langweilig. Doch etwas zeichnen ist erlaubt. Auf den Entwurf kann man Anmerkungen schreiben, die erklären:

- wie ein deaktiviertes Element wieder aktiv wird
- was durch das Anklicken bestimmter Elemente ausgelöst wird
- welche unterschiedene unterstützte Formate für ein Eingabefeld sind
- Variationen von Inhaltselementen
- wahrscheinliche Interface-Animationen

Es gab am Anfang zwei Möglichkeiten um diesen Content mit Photos, Videos und Text darzustellen. Im iOS spricht man von "Table Views"⁶ (Tabellenansicht) und "Collection Views"⁷ (Sammlungsansicht).

Eine Table View zeigt eine einzelne Spalte mit vertikal scrollenden Inhalten, unterteilt in Zeilen. Jede Zeile einer Tabelle zeigt eine einzelne Information an. Eine Collection View ist dagegen organisiert in Zellen, die eine schachbrettartige geordnete Sammlung von Datenelementen verwaltet und diese in benutzerdefinierbaren Layouts präsentiert. Es wird oft in der Photo App, im AppStore, iTunes etc. benutzt.

Mit Beiden sind komplexere Layouts möglich.

Jetzt stellt sich die Frage, welches Design am besten für die App geeignet ist.

Durch das Zeichnen von Wireframes bekommt man ein Gefühl für die App.

Beide Möglichkeiten wurden skizziert und verglichen. Die Skizzen mit der Table View Ansicht, und die Collection View Ansicht sind in Anhang zu sehen. ([A2. Wireframes Entwürfe](#))

Diese Skizzen zu zeichnen dauerte rund 40 Minuten.

3.3 Entwurf mit Sketch⁸

Sketch ist eine beliebte App für UX- und UI-Design, ein Vektorgrafik-Editor, der von der niederländischen Firma Bohemian Coding entwickelt wurde. Es ist in erster Linie ein Werkzeug für Website- und Mobile-App-Design, aber in letzter Zeit wurde der Fokus verstärkt auf Prototyping und Zusammenarbeit gelegt, um es zu einer umfassenden Plattform für digitales Design zu machen. Im Unterschied zu Photoshop, arbeitet Sketch vektorbasiert und es ist damit einfacher, Logos zu entwerfen und zu erstellen.

Als alleiniger Entwickler für dieses Projekt hielt ich es für wichtig, eine Realitätstreue Skizze und ein Wireframe zu erstellen, welche ich den Kollegen zeigen kann.

Ich habe mit dem Logo angefangen. ([A3.3 Entwurf in Sketch - Logo](#))

⁶ https://developer.apple.com/documentation/uikit/views_and_controls/table_views

⁷ <https://developer.apple.com/documentation/uikit/uicollectionview>

⁸ <https://www.sketch.com>

Das Logo in Sketch als Vektor zu haben ist wichtig, weil man damit die Icons für die Apps leicht erstellen und in unterschiedliche Größen in Xcode exportieren kann. ([A4.1 AppIcon in Xcode](#))

Um Optionen zu vergleichen, wurde ein Prototyp des ersten Bildschirms der App erstellt, einmal mit einer Tabellenansicht und einmal mit der Sammlungsansicht. Beide sind im Anhang zu finden. ([A3.1 Entwurf in Sketch - Table View](#)) und ([A3.2 Entwurf in Sketch - Collection View](#)) Beide sind für die App geeignet, aber angesichts der Menge an Informationen über die Ausstellungsstücke, die wir dem Benutzer zur Verfügung stellen wollen, "fühlt" sich die Tabellenansicht mit Titel und Untertitel besser an, als die Bildansicht. Die Detailansicht wird ein Popup-Fenster mit Titel, Untertitel, einem größeren Bild, sowie eine Beschreibung des Artikels mit zugehörigem Link zum Video sein. ([A3.4 Entwurf in Sketch - Detail View](#))

3.4 MVC Design Pattern

MVC ist eines der drei grundlegenden Designmuster von iOS.

Die anderen beiden sind: Delegieren (delegation), ein Objekt dazu bringen, etwas im Namen eines anderen zu tun; und Ziel-Aktion (target-action), indem es Ereignisse wie Tastenanschläge mit Aktionsmethoden verbindet.

Das Model-View-Controller-Muster besagt, dass die Objekte in folgende Bereiche aufgeteilt werden können:

- **Model-Objekte:** Diese Objekte enthalten die Daten und alle Operationen mit den Daten. Wenn man beispielsweise eine Kochbuch-App schreiben würde, würde das Modell aus den Rezepten bestehen. In einem Spiel wäre es das Design der Levels, der Spielerwert und die Positionen der Monster. Das Model der App ist die `models.JSON`-Datei mit der Beschreibung der Exponate. ([A4.2 models.JSON](#)).
- **View-Objekte:** Diese bilden den visuellen Teil der App (Bilder, Schaltflächen, Beschriftungen, Textfelder, Zellen der Tabellenansicht usw.). Eine View kann sich selbst zeichnen und auf Benutzereingaben reagieren, aber sie verarbeitet typischerweise keine Anwendungslogik. Viele Views, wie z.B. `UITableView`, können in vielen verschiedenen Anwendungen wiederverwendet werden, da sie nicht an ein bestimmtes Datenmodell gebunden sind.
- **Controller-Objekte:** Der Controller ist das Objekt, das die Model-Objekte mit den Views verbindet. Es "hört" beim Tippen auf die Views, lässt die Model-Objekte einige Berechnungen durchführen und aktualisiert die Views, um den neuen Zustand des Models widerzuspiegeln. Dafür ist der Controller verantwortlich. Unter iOS wird der Controller als "View Controller" bezeichnet.

Konzeptionell passen diese drei Bausteine so zusammen:

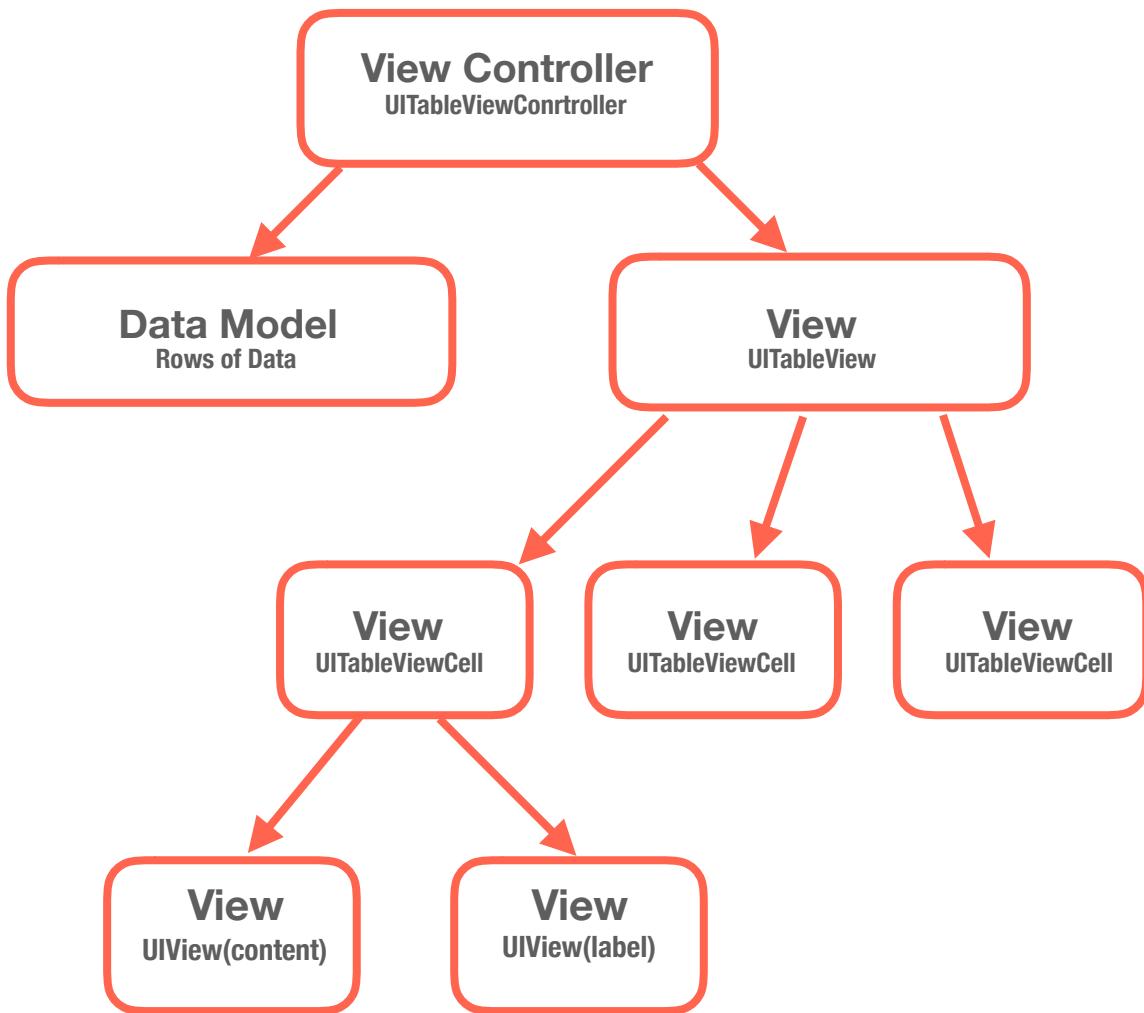


Abbildung 1: MVC Design Pattern

3.5 Die Modellschicht

Man hat sich entschieden, die Assets lokal und nicht auf einem externen Server zu speichern. Es gibt mehrere Möglichkeiten Daten in die App zu integrieren. In diesem Fall wird eine einfache JSON-Datei verwendet, die die Zellen mit den notwendigen Informationen zur Darstellung versorgen wird. Die Videos werden lokal im Ordner "Videos" gespeichert. Die Bilder sind jedoch aus Optimierungsgründen in einem speziellen Ordner "Assets.xcassets" abgelegt.

3.5.1 JSON

Das JSON-Format (JavaScript Object Notation) eignet sich aufgrund seiner einfach lesbaren Textform sehr gut für den Austausch strukturierter Daten zwischen Anwendungen. Dabei werden Daten über assoziative Arrays {...} und Listen [...] repräsentiert, und es eignet sich sehr gut um Daten in Swift zu verarbeiten.

Die UIKit Foundation Klassen `JSONDecoder` und `JSONEncoder` erlauben eine Konvertierung von JSON-Daten zu Swift-Typen. Dabei empfiehlt es sich, Datentypen als *struct* zu definieren, die 1:1 der Struktur aus dem JSON entsprechen. Diese müssen mit dem Protokoll `Codable` versehen werden.

Die `models.json` Datei⁹ sieht so aus:

```
{  
    "title": "Amie1.0",  
    "subtitle": "Full-scale building enclosure prototypes.",  
    "image": "Amie1.0",  
    "contentText": "AMIE 1.0 is a 3D-printed full-scale building enclosure prototype. The structure was printed using BAAM technology and incorporates next-generation modified atmosphere insulation panels. \rThe structure is a mobile living space with the ability to produce,[...]"  
}
```

JSON-Dateien werden mit einer Instanz von `JSONDecoder()` dekodiert. `JSONModels` ist ein Array vom Typ `Model` und an der Eigenschaft `models` zugewiesen:

```
var models = [Model]()  
  
func parse(JSON: Data) {  
  
    let decoder = JSONDecoder()  
  
    if let JSONModels = try? decoder.decode(Models.self, from: JSON) {  
        models = JSONModels.models  
    }  
}
```

Die Klasse `Models` und `Model` wird in einer getrennten Datei erstellt und hier unten in 3.5.2 nochmal erklärt.

3.5.2 Data Struktur in `Model.swift`

Es wird eine `ModelCell.swift` Datei erstellt.

In dieser Datei definiert man zwei typen:

- ein *struct Model* mit 4 Eigenschaften, die die Felder der JSON-Datei widerspiegeln werden.

⁹ Für die Erstellung der JSON-Datei hat man das Code Beautifier Plugin verwendet. Dies hat sehr geholfen, eine große JSON-Datei zu debuggen. <https://codebeautify.org/JSONviewer>

```
struct Model: Codable {  
    var title: String  
    var subtitle: String  
    var image: String  
    var contentText: String  
}
```

- ein *struct Models*, der ein Array vom Typ *Model* sein wird:

```
struct Models: Codable {  
    let models: [Model]  
}
```

Wichtig ist hier, dass der *struct* mit dem *Codable* Protokoll konform ist. Das ermöglicht Swift, die Daten aus der JSON-Datei in die Data Struktur umzuwandeln und diese werden vom Controller als Prototyp-Zellen für die Tabellenansicht verwendet.

3.6 Der View Controller

Im Allgemeinen behandelt ein View-Controller einen Bildschirm der App. Wenn die Anwendung mehr als einen Bildschirm hat, wird jeder von einem separaten View-Controller behandelt und hat seine eigenen Views. Die App fließt von einem View-Controller zum anderen.

Eine View und ein View-Controller sind zwei verschiedene Dinge. Eine View ist ein Objekt, das etwas auf den Bildschirm zeichnet, wie z.B. eine Schaltfläche oder ein Label. Der View-Controller ist das, was hinter den Kulissen funktioniert. Es ist die Brücke, die zwischen dem Datenmodell und den Views besteht.

Diese App wird zwei View Controller haben: *ViewController.swift* (der Table View Controller) und *DetailViewController.swift* um die Popup View zu steuern. Beide Files sind im Anhang dargestellt. ([A5.1 Quellcode - ViewController.swift](#)) ([A5.2 Quellcode - DetailViewController.swift](#))
Der Navigations-Controller wird vom UIKit Framework zur Verfügung gestellt.

3.7 Die View

Das Anzeige-Layout des View-Controllers wird im Storyboard gestaltet.

Storyboards sind ein erstmals in iOS 5 eingeführtes Feature, das die Zeit verkürzt, Benutzeroberflächen für iOS Anwendungen zu erstellen. Storyboards ermöglichen, mehrere View-Controller-Ansichten innerhalb einer Datei zu gestalten und Übergänge zwischen View-Controllern zu erstellen. Storyboards sind in der Main.storyboard File gespeichert und ein Screenshot davon ist im Anhang zu sehen. ([A4.3 main.storyboard](#))

3.7.1 Autolayout

Um die Views zu gestalten muss man "Constraints" festlegen. Constraints sind Einschränkungen, die die Größe und Abstände der Elemente und Subviews auf dem Bildschirm bestimmen. Dies wurde mit Auto Layout gemacht.
Auto Layout ist eine wichtige UIKit-Technologie, die es einfach macht, viele verschiedene

Bildschirmgrößen in der App zu unterstützen, indem die richtigen Constraints gesetzt werden. Ein Beispiel anhand der Cell-View-prototype ist im Anhang. ([A4.4 Constraints Beispiel- Cell View Prototype](#)). Hier kann man die 8 Constraints von der Table View Cell rechts unten, als auch in dem Storyboard sehen.

3.7.2 LaunchScreen.storyboard

Der Startbildschirm wird angezeigt, sobald der Benutzer die App-Icon antippt und er bleibt auf dem Bildschirm, bis die Haupt-Oberfläche angezeigt wird. Für den Startbildschirm wird die LaunchScreen.storyboard Datei in Xcode ausgewählt und im Interface Builder eine neue Bildansicht hinzugefügt. Es wird sichergestellt, dass in den Einstellungen die Startbild-Datei des ausgewählten LaunchScreen.storyboards angezeigt wird. Als Bild verwendet man die Datei mit dem Logo, welche zuvor mit Sketch vorbereitet wurde. Anhang ([A4.5 LaunchScreen.storyboard](#))

3.8 Delegation Design Pattern

Delegation ist ein Entwurfsmuster, das es einer Klasse oder Struktur ermöglicht, einen Teil ihrer Verantwortlichkeiten an eine Instanz eines anderen Typs weiterzugeben (oder zu delegieren). Mit dem Delegationssystem kann die Table View einfach eine Meldung senden, dass ein Tap aufgetreten ist, und den Delegierten die Angelegenheit erledigen lassen.

In der Regel haben die Komponenten nur einen Delegierten. Die Table View teilt es jedoch in zwei separate Helfer auf: die *UITableViewDataSource* zum Einfügen von Zeilen in die Tabelle und das *UITableViewDelegate* zum Behandeln von Tippen auf die Zeilen und mehrere andere Aufgaben. Im Grunde bedeutet dies, dass die Delegierteneigenschaft ein Objekt sein muss, das *somethingHappened(_)* implementiert hat. Dieses Objekt ist verantwortlich für die Behandlung dessen, was in seiner View passieren muss, nachdem der User die *somethingHappened(_)* Methode verwendet hat. Es ist ein sehr interessantes Muster und wird in iOS häufig verwendet.

3.9 Beispiel von MVC und Delegation anhand den *UISearchController*¹⁰

MVC und Delegation sind im Apple iOS Framework unerlässlich. Der Controller erstellt die View, aber die View spricht nicht direkt mit dem Controller und ist logisch getrennt. Durch die Delegation wird das Suchfeld entweder präsentiert oder ausgeblendet und durch die Implementierung von dem *UISearchResultsUpdating* Protokoll wird die Table View ständig mit den Suchergebnissen aktualisiert. Die Suchfeld-View wird durch den SearchController gesteuert und gehört als Eigenschaft zum Navigationselement des View Controllers, der die automatisch oben in die Navigationsleiste platziert, wenn die Table View angezeigt wird. In der Klasse *ViewController* wird die *UISearchControllerDelegate* Klasse implementiert und eine Instanz von *UISearchController* erstellt. ([A5.1 - Quellcode](#))

¹⁰ <https://developer.apple.com/documentation/uikit/uisearchcontroller>

Das Model kommt unten in der `updateSearchResults(for searchController:)` Methode vor.

```
//In dem ViewController wird UISearchControllerDelegate implementiert
class ViewController: UITableViewController, UISearchControllerDelegate{

    // es wird einen neuen UISearchController erstellt
    let searchController = UISearchController(searchResultsController: nil)
    // searchController ist Navigationselement des View Controllers
    self.navigationItem.searchController = searchController

    [...]
}

// Implementierung des UISearchResultsUpdating protokolls.
extension ViewController: UISearchResultsUpdating {
    //Diese Methode wird vom UISearchResultsUpdating Protokoll benötigt
    func updateSearchResults(for searchController: UISearchController) {
        if let text = searchController.searchBar.text, !text.isEmpty {
            //Hier wird auf den Model zugegriffen
            filteredModels = models.filter { model in
                model.title.localizedCaseInsensitiveContains(text)
            }
        } else {
            filteredModels = models
        }
        tableView.reloadData()
    }
}
```

4 - Implementierung

4.1 Xcode

Die App wird für iOS 12.0 und neuere mit Xcode¹¹ entwickelt. Xcode ist eine integrierte Entwicklungsumgebung für macOS mit einer Reihe von Werkzeugen, die von Apple für die Entwicklung von Software für macOS, iOS, iPadOS, watchOS und tvOS entwickelt wurden.

4.2 Zugriffskontrolle

Ein grundlegendes Konzept von objektorientierter Programmierung ist die Datenkapselung, die mit den Schlüsselwörtern `private`, `protected` und `public` gesteuert wird. Swift reduziert die Notwendigkeit, explizit Zugriffsschutz festzulegen, indem es Standard-Zugriffsebenen für typische Szenarien bereitstellt. Wenn, wie in diesem Fall, eine Single-Target-App geschrieben wird, muss man möglicherweise überhaupt keine expliziten Sichtbarkeiten angeben.¹² Alle Entitäten in dem Code haben eine Default Sichtbarkeit von `internal`.

¹¹ <https://developer.apple.com/xcode/>

¹² <https://docs.swift.org/swift-book/LanguageGuide/AccessControl.html>

4.3 GIT

Git ist ein Versionskontrollsysteem (version control system). Es ermöglicht ein Schnappschuss des Systems zu machen, so dass man später immer wieder zurückkehren und eine Historie der Änderungen am Projekt einsehen kann. Das Projekt wird unter Versionskontrolle gehalten. Dies ist in Xcode einfach zu schaffen. Es werden lokale Commits gemacht und bei jedem größeren Update und jeder größeren Funktionalität wird der Code auf GitHub gepusht. Man kann es über die App oder im Terminal tun.

Es ist allerdings sinnvoll eine .gitignore Datei zu schaffen. Eine .gitignore Datei sagt git, welche Dateien (oder Muster) es ignorieren soll. Es wird normalerweise verwendet, um zu vermeiden, dass vorübergehende Dateien aus dem Arbeitsverzeichnis übertragen werden, die für andere Mitarbeiter nicht nützlich sind, wie z.B. temporäre Dateien die erstellt werden, usw.

Diese Datei kann einfach in Terminal erstellt werden. [\(A5.5 Git\)](#)

4.4 Der Assets.xcassets Folder

Etwas, das man bei Fotodateien beachten sollte, ist, dass selbst wenn wir ein Bild in einem jpeg-Format von nur wenigen MB öffnen, iOS es dekomprimieren muss, um es zu verwenden, und das kann sehr viel Speicherplatz in Anspruch nehmen. Es ist besser, den Asset-Ordner für Bilder gemäß den Apple-Richtlinien zu verwenden.

Dieser Ordner ermöglicht es iOS, die Speicherung und den schnellen Zugriff auf die Bilder zu optimieren. Xcode kompiliert die Assets zusammen mit dem Code für maximale Leistung.

Auch das Logo für die App muss in unterschiedlicher Auflösung dort gespeichert werden. Das Gute daran ist, dass alle Bilder, die für das App-Icon erstellt werden, für alle Geräte und Größen optimiert sind. Andere Bilder werden hier abgelegt und bei der Kompilierung speicheroptimiert.

4.5 Suchen

Die App zeigt beim Start das Logo des 3DPC (3D Pioneer Challenge) und öffnet sich dann im Hochformat in einer Table View. Wenn der Benutzer die Liste der Modelle vorsichtig herunterzieht, sieht er die Sucheingabe, um nach einem bestimmten Modell zu suchen. Das Suchfeld wird ansonsten standardmäßig nicht angezeigt. Der Benutzer kann nach unten scrollen und eine Detailansicht sehen, in der er das Element in der Liste auswählt.

Zusammenfassend lässt sich sagen, dass die JSON-Daten ein Dictionary darstellen und in diesem Dictionary befinden sich eine Reihe von weiteren Dictionaries. Jedes der Dictionaries aus dem Array repräsentiert ein Suchergebnis.

4.6 ViewController.swift

Die Klasse *UITableViewController* ist eine Unterklasse von *ViewController*. Diese Klassen werden von dem *UIKit* Apple Framework zur Verfügung gestellt.

In Swift for iOS hat jede Ansicht das MVC (Model View Controller) Muster. Es gibt tatsächlich viele MVC's in einer App, jeder Bildschirm hat einen. In diesem Fall hat die View also einen View Controller in der Datei `ViewController.swift`. Eine Tabellenansicht lädt die Daten aus dem Modell und zeigt sie in benutzerdefinierten Zellen an. Es werden zwei benutzerdefinierte Zellen für den Inhalt und für den Fall, dass die Suche nichts zurückgibt, erstellt.
In `viewDidLoad()` wird die View eingerichtet, dem Navigationscontroller einen Titel gegeben und einen `SearchController` erstellt.

5 - Unit Testing

Modultests zählen zu den White-Box-Tests. Das heißt, dass der zu testende Quellcode bekannt ist. Unit Testing wird möglichst automatisiert ausgeführt. Im Allgemeinen sollten die Tests Folgendes abdecken:

- Kernfunktionalität von Klassen und Methoden .
- die gängigsten UI-Workflows, Randbedingungen.

5.1 Best Practices für Tests - FIRST

Das Akronym FIRST beschreibt einen kurzen Satz von Kriterien für effektive Unit-Tests.

Diese Kriterien sind:

- **Fast:** Die Tests sollten schnell laufen.
- **Independent/Isolated:** Tests sollten den Status nicht miteinander teilen.
- **Repeatable:** Sie sollten bei jeder Durchführung eines Tests die gleichen Ergebnisse erzielen.
- **Self-validating:** Die Tests sollten vollständig automatisiert und die Ausgabe sollte entweder "pass" oder "fail" sein.
- **Timely:** Im Idealfall sollten Tests geschrieben werden, bevor der getestete Production Code geschrieben wird (Test-Driven Development).

Die Einhaltung der FIRST-Grundsätze hält die Tests klar und hilfreich, anstatt sich in Blockaden für die App zu verwandeln. Im Mittelpunkt der testgesteuerte Programmierung (eine Methode, die häufig bei der agilen Entwicklung von Computerprogrammen eingesetzt wird) steht das Konzept, dass der Test zuerst geschrieben wird. Danach beginnt man mit der Programmierung, die man solange fortführt, bis der Test erfolgreich durchlaufen wird.

Der Testnavigator in Xcode bietet die einfachste Möglichkeit mit Tests zu arbeiten. Es können damit Testziele erstellt und Tests durchgeführt werden.

Mit "New Unit Test Target" und "New UI Unit Test Target" werden Testklassen erstellt.
(Abbildung 2)

Museums App

Programmieren von einer "Museum App" für iPhone und iPad

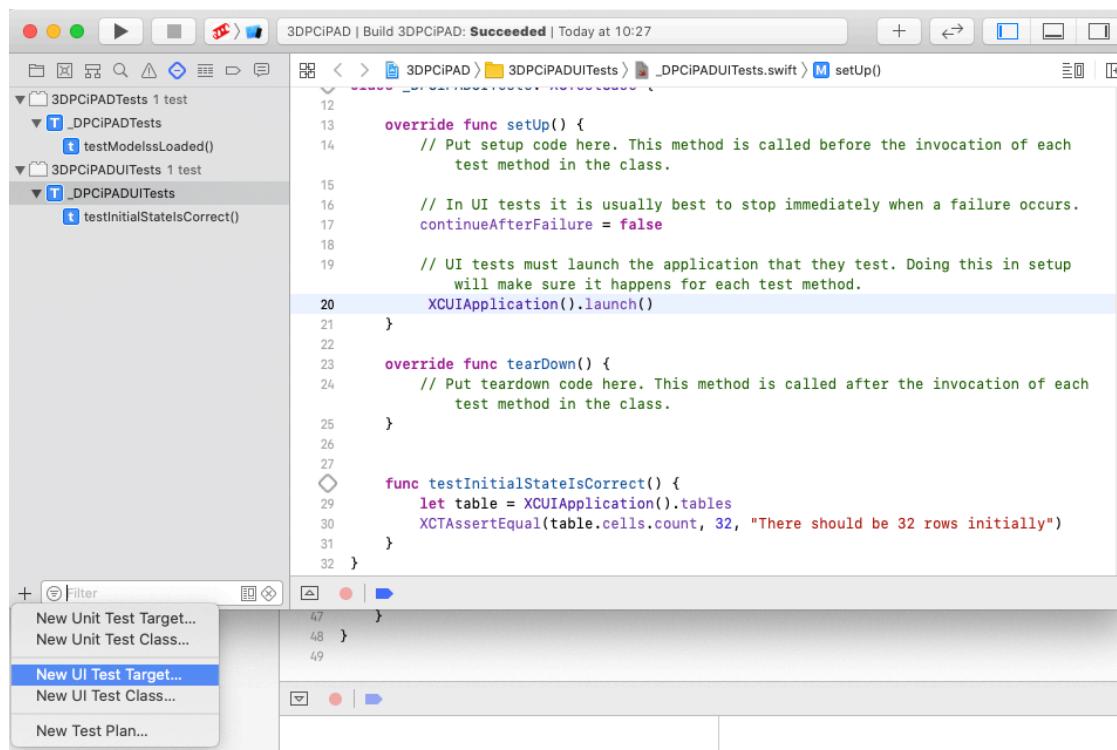


Abbildung 2: Testklassen werden erstellt

5.2 Mit XCTAssert Modelle testen.

Die XCTAssert-Funktionen werden verwendet, um eine Kernfunktion der App zu testen. Der Inhalt von `setup()` wird durch diesen ergänzt:

`XCUIApplication().launch()`

UI-Tests müssen die Anwendung starten, die sie testen. Wenn dies im Setup passiert, wird sichergestellt, dass es für jede Prüfmethode geschieht.

Dies ist ein Beispiel von einer Testmethode mit XCTestAssertions:

```
func testInitialStateIsCorrect() {
    let table = XCUIApplication().tables
    XCTAssertEqual(table.cells.count, 32, "There should be 32 rows")
}
```

Diese Methode prüft, dass wenn die App startet, eine bestimmte Anzahl von Zeilen (32) in die Table View erzeugt wird.

Eine vollständige Liste der XCTestAssertions ist auf der Apple Developer Webseite ersichtlich.¹³

¹³ <https://developer.apple.com/documentation/xctest#2870839>

5.3 Deployment und Übergabe

Die gemieteten iPads sind zu uns geliefert worden. Wir haben die App installiert und einen Black-Box-Test ausgeführt indem wir die App gestartet und die verschiedenen Funktionalitäten getestet und protokolliert haben.

Folgende Schritte haben wir durchgeführt:

- App startet
- Logo wird beim Starten gezeigt
- die Suchfunktion funktioniert
- die Modelle werden angezeigt
- bei der Auswahl einer Zeile wird die Detail-View angezeigt mit Foto und Beschreibung
- Video kann abgespielt werden
- man kann zurück zur Start-View navigieren

Die iPads wurden dann zur Ausstellung geliefert.

6 - Fazit

Der zu Beginn des Projektes erstellte Projektplan konnte eingehalten werden. Es ist zu erkennen, dass nur sehr geringfügig von der Zeitplanung abgewichen wurde. Die sich daraus ergebenen Differenzen konnten untereinander kompensiert werden, sodass das Projekt in dem von der IHK festgelegten Zeitrahmen von 70 Stunden umgesetzt werden konnte.

Im Zuge des Projektes konnte ich wertvolle Erfahrungen bzgl. der Planung und Programmierung von iOS Apps sammeln. Außerdem konnten neue Erkenntnisse in Bezug auf Einbinden und Nutzen von Xcode iOS Frameworks gewonnen werden.

Abschließend kann man sagen, dass die Realisierung des Projektes eine große Bereicherung war. Im Rahmen "Augmented Reality" könnte die App in Zukunft noch erweitert werden.

Literaturverzeichnis

Paul Hudson (2019) - Hacking with iOS, Self Published.

Paul Hegarty (2017) Stanford University CS 193P - *Developing iOS 11 Apps with Swift - Lecture 2. Model View Controller Design Pattern*, <https://youtu.be/w7a79cx3UaY>.

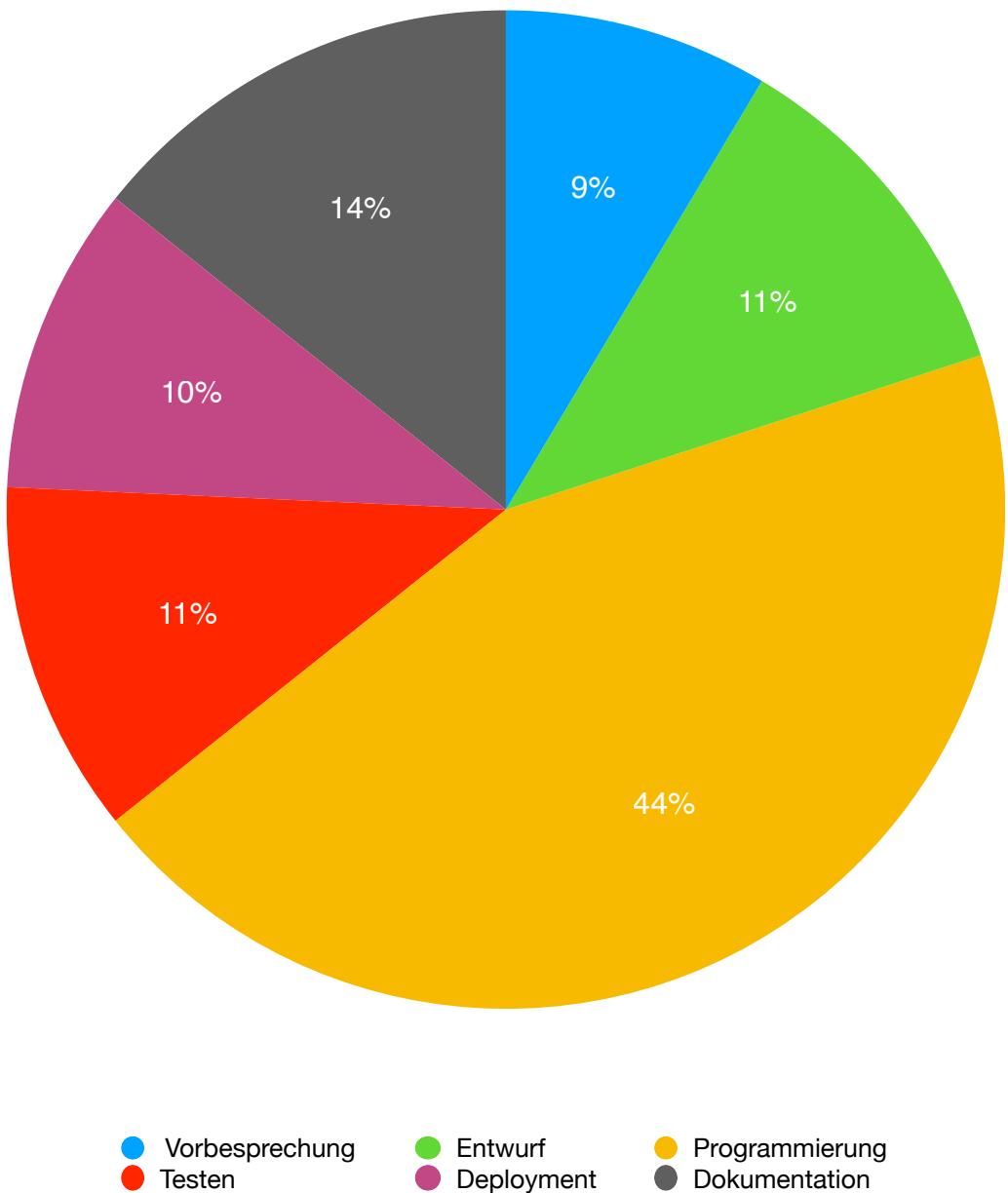
Ray Wenderlich Tutorial Team (2019) - *Swift Apprentice (Fifth Edition): Beginning Programming with Swift*, Razeware LLC (November 25, 2019),
ISBN-13: 978-1950325078.

Freeman u. a. (2014) - *Head First Design Patterns*. O'Reilly Media,
ISBN-13: 978-0596007126

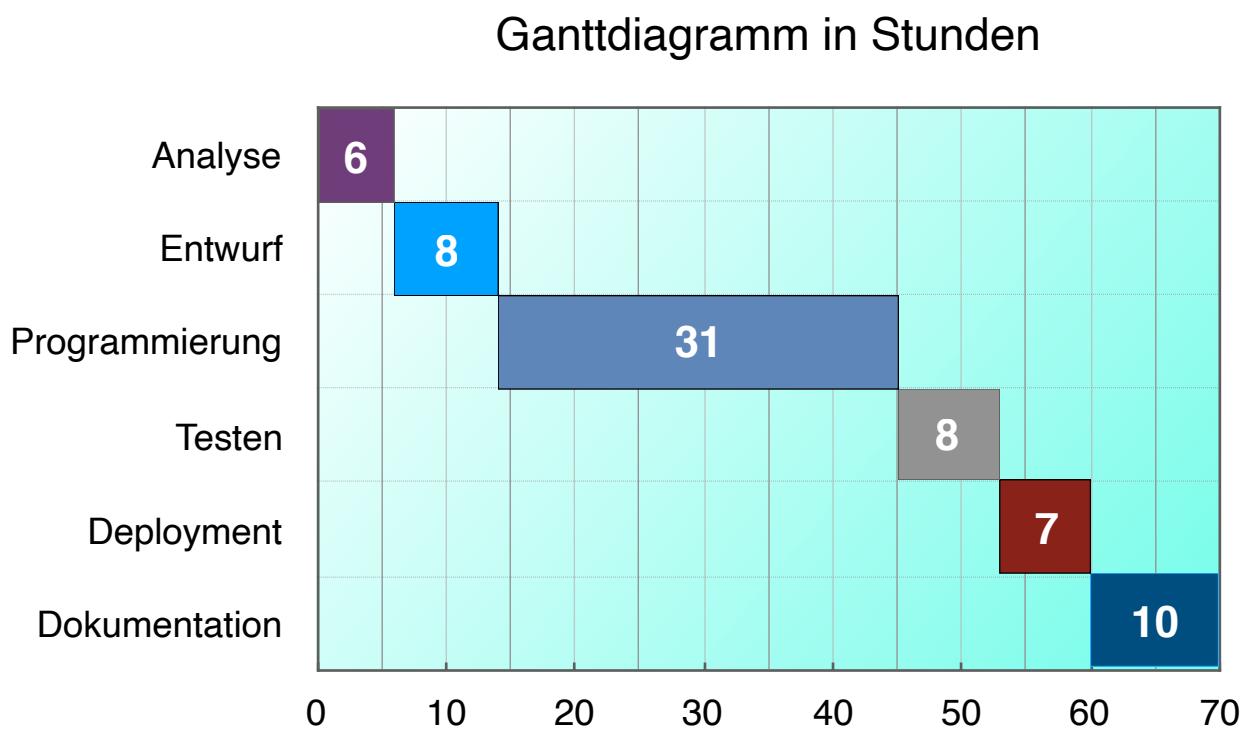
John Sundell (2019) *Unit Testing*,
<https://www.swiftbysundell.com/basics/unit-testing/>

Anhang

A1.1 Detaillierte Zeitplanung: Tortendiagramm



A1.2 Detaillierte Zeitplanung



A1.3 Detaillierte Auflistung von Ressourcen

Hardware

Für Entwickeln von dem Software und Testen:

- ★ Mac Mini 2018, 3.2GHz 6-core Intel Core i7
- ★ Apple iPad 9.7" (Wi-Fi/Cellular) early 2018, 32 GB - Modell A1954

Gemietet für die Ausstellung ¹⁴ :

- ★ Apple iPad Pro 9.7" (Wi-Fi/Cellular) 32 GB early 2018 - Modell A 1674

Software

- ★ macOS Mojave und später macOS Catalina
- ★ Xcode 11.0
- ★ Sketch v52
- ★ Apple Pages (für die Dokumentation)
- ★ iOS 12 und iOS13
- ★ Github

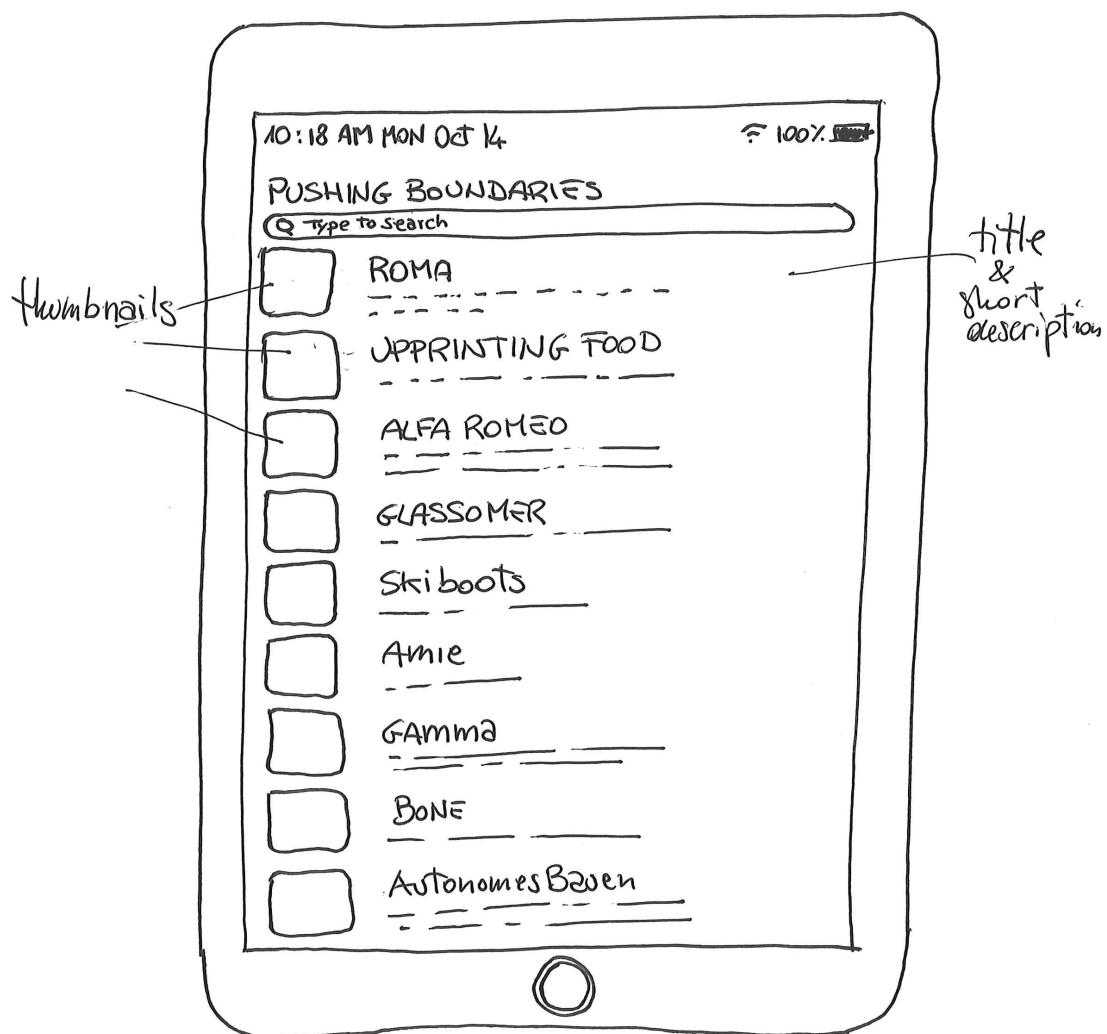
¹⁴ Apfelverleih.de

A2 - Wireframes Entwürfe

A2.1 List View:

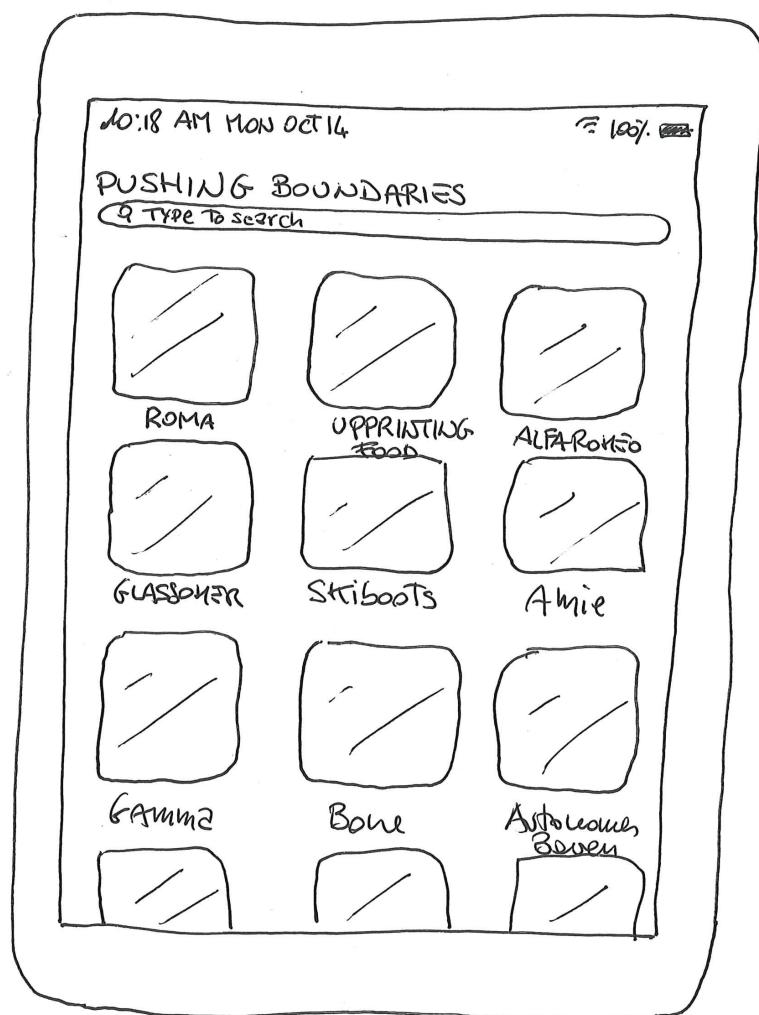
3DPC - Pushing Boundaries
Main screen iPad

(2)

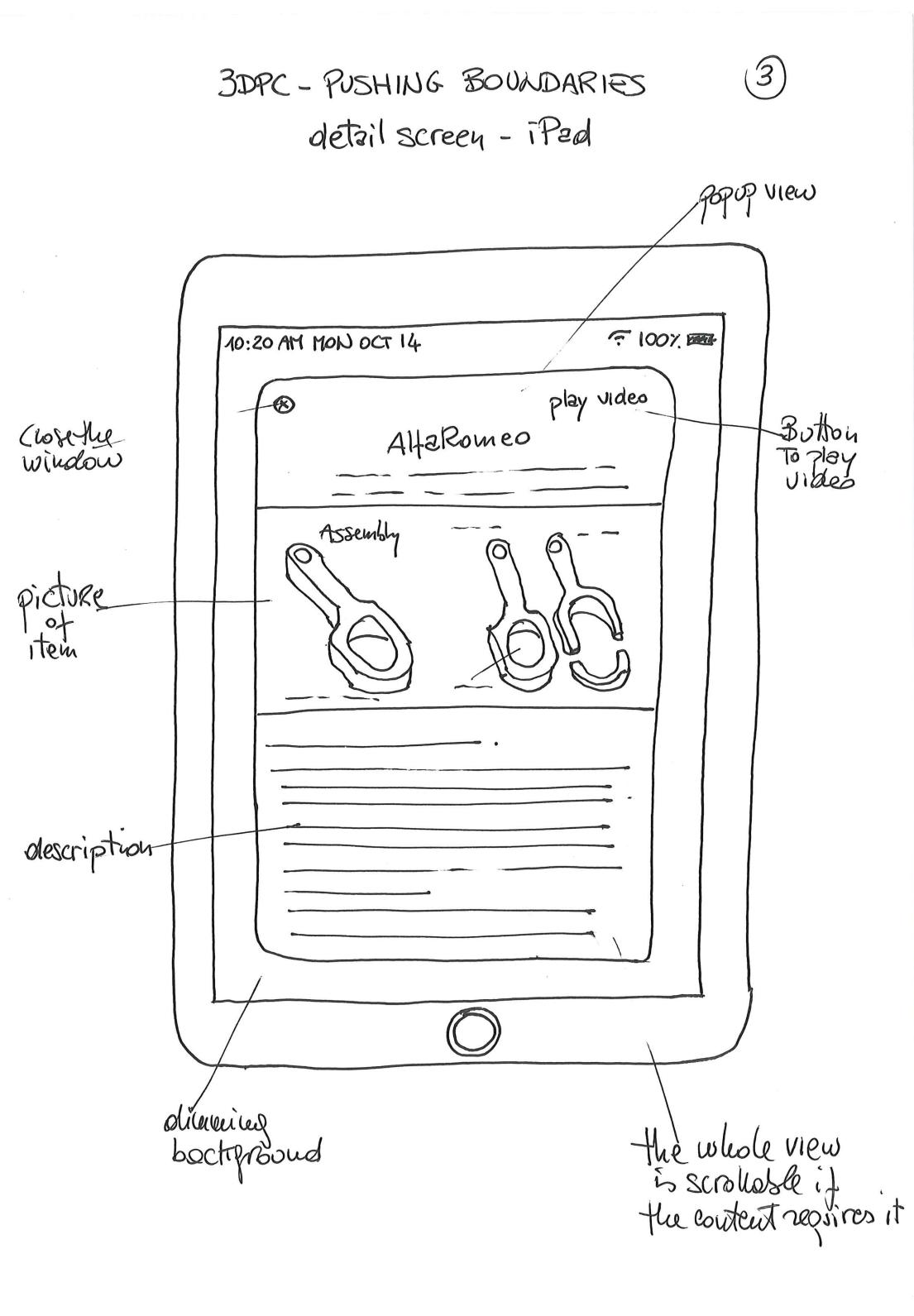


A2.2 Collection View:

3DPC - PUSHING BOUNDARIES
Main Screen iPad with
COLLECTION VIEW

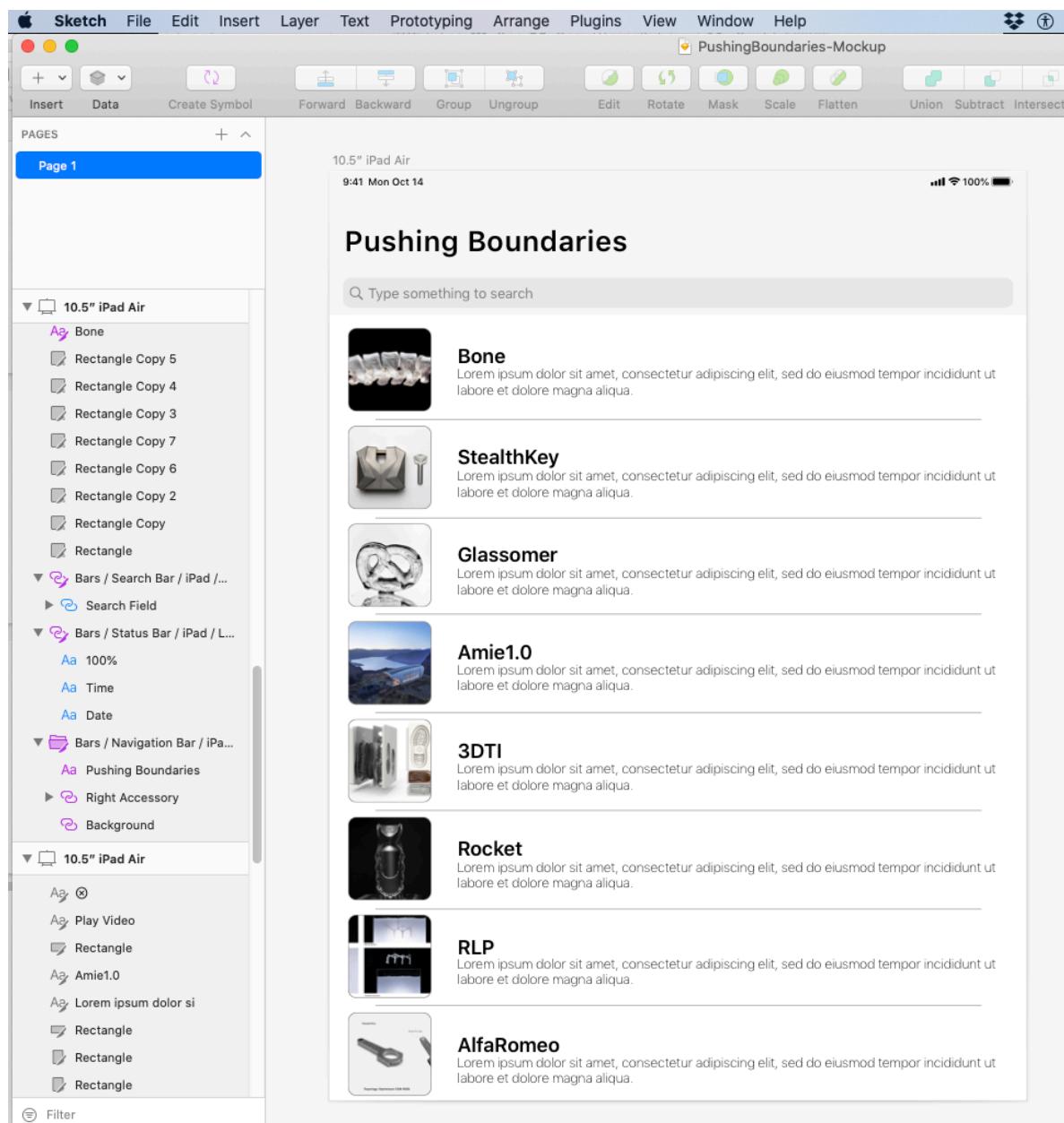


A2.3 Detailed View:

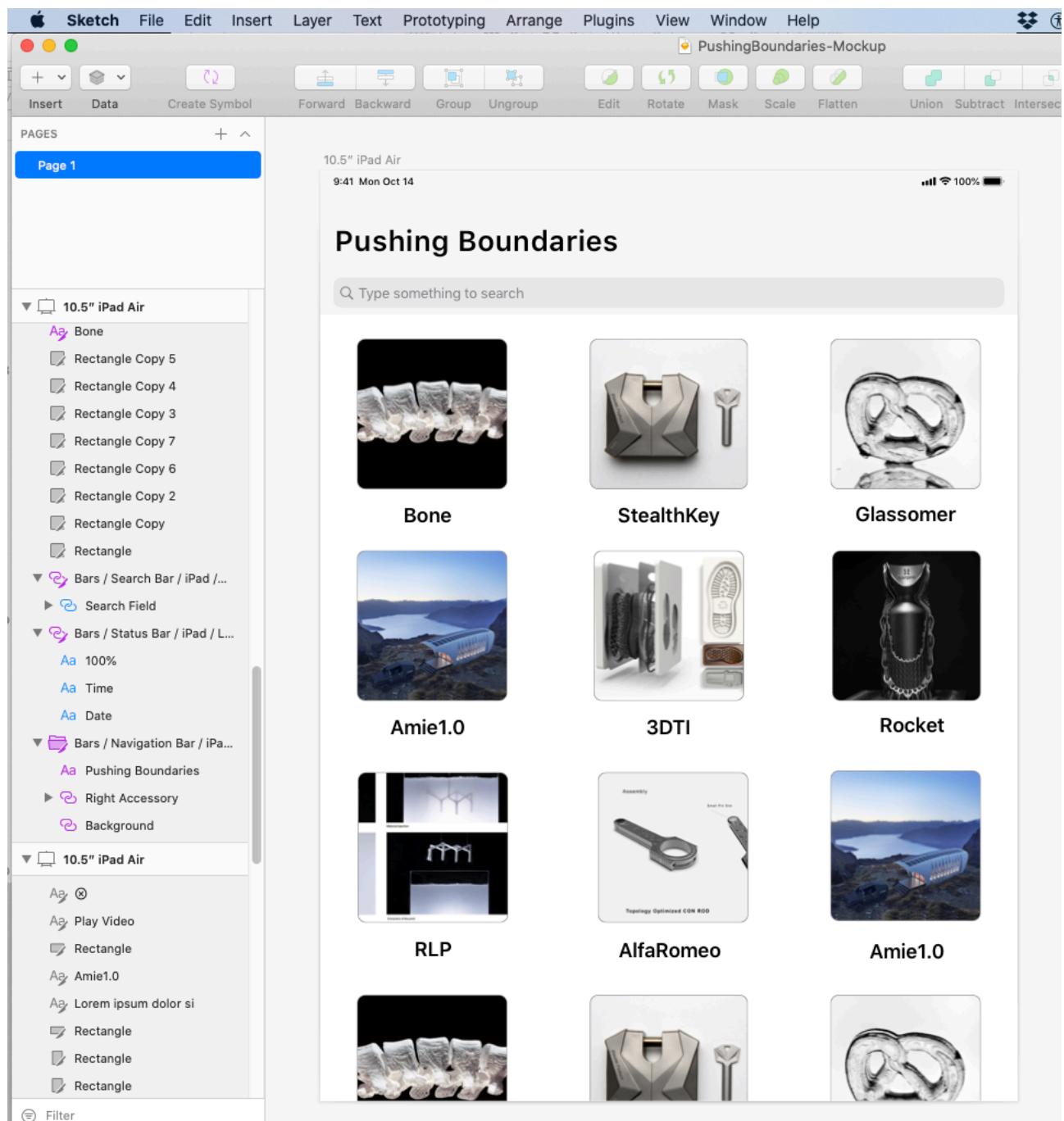


A3 - Sketch Entwurf

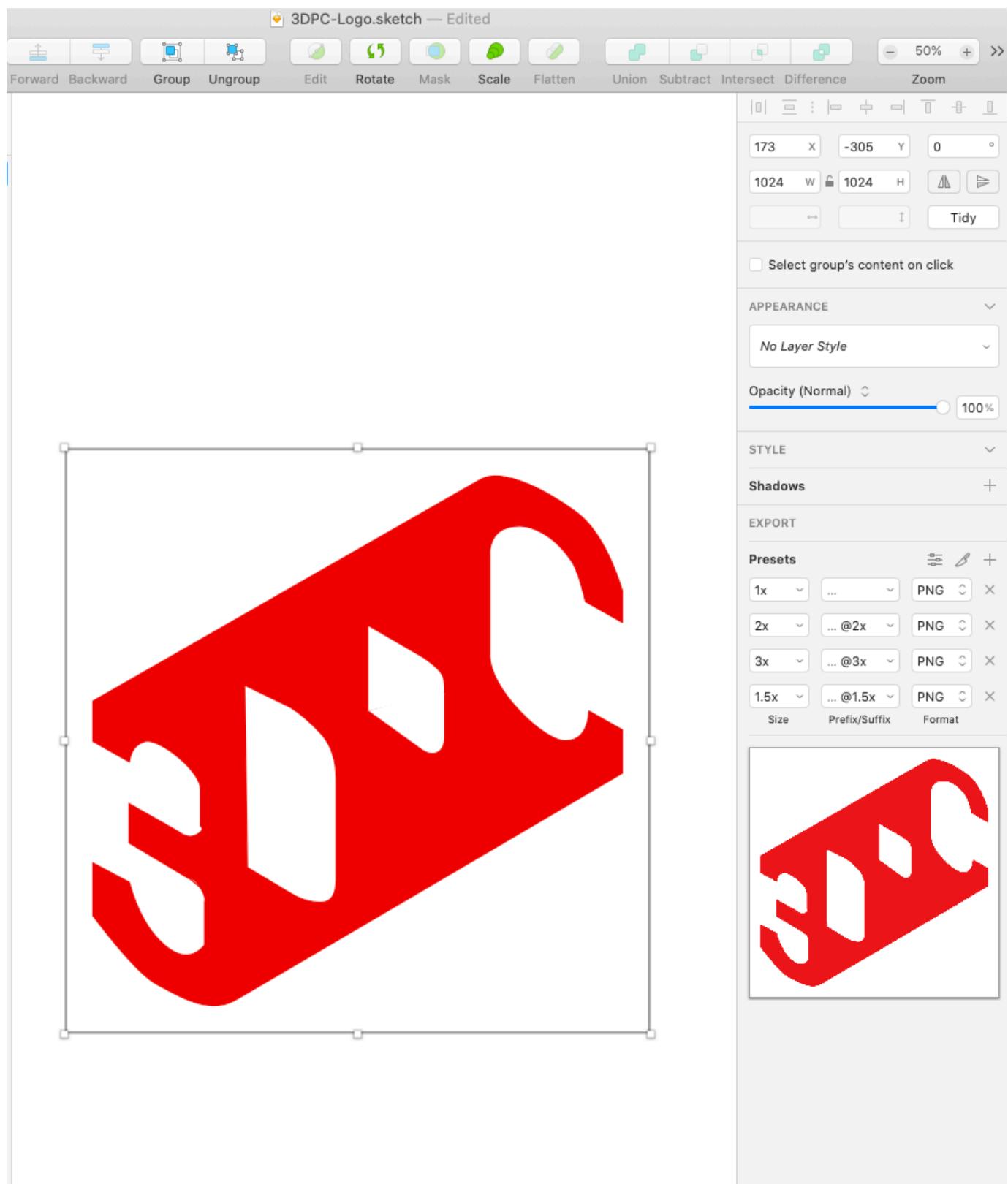
A3.1 Table View



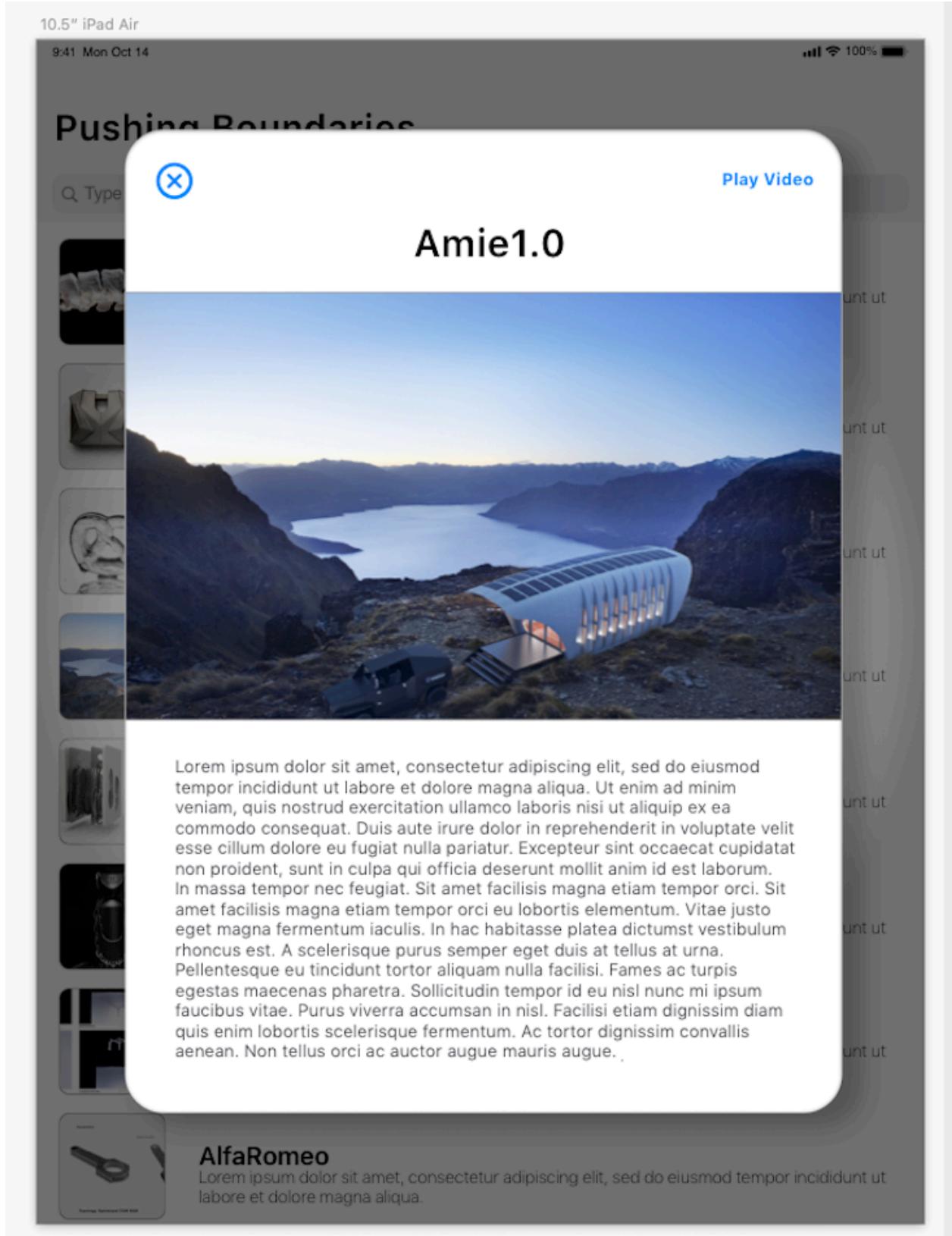
A3.2 Entwurf in Sketch - Collection View



A3.3 Entwurf in Sketch - Logo

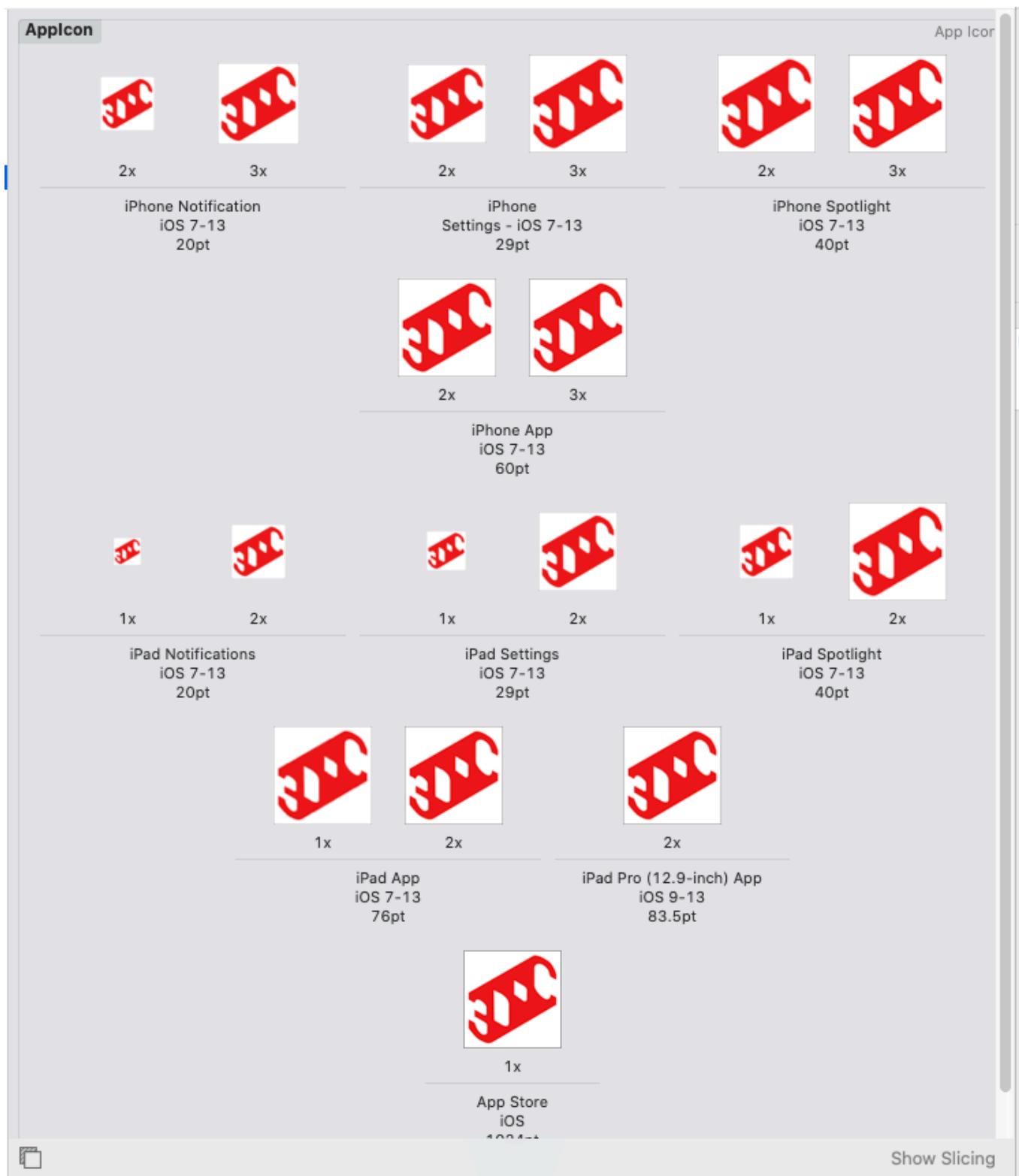


A3.4 Entwurf in Sketch - Detail View



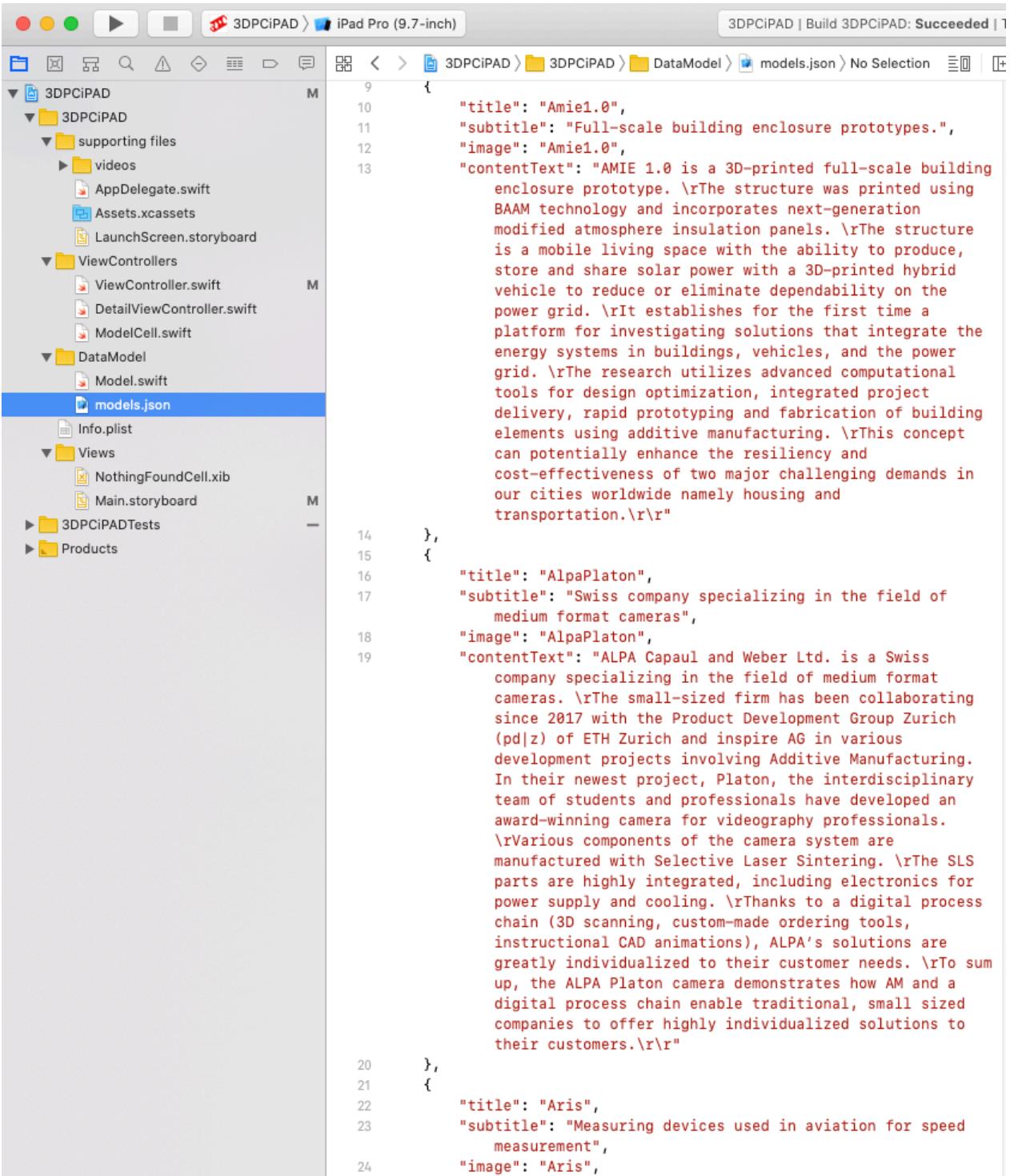
A4 - Xcode Screenshots

A4.1 AppIcon in Xcode



Museums App
Programmieren von einer "Museum App" für iPhone und iPad

A4.2 models.JSON

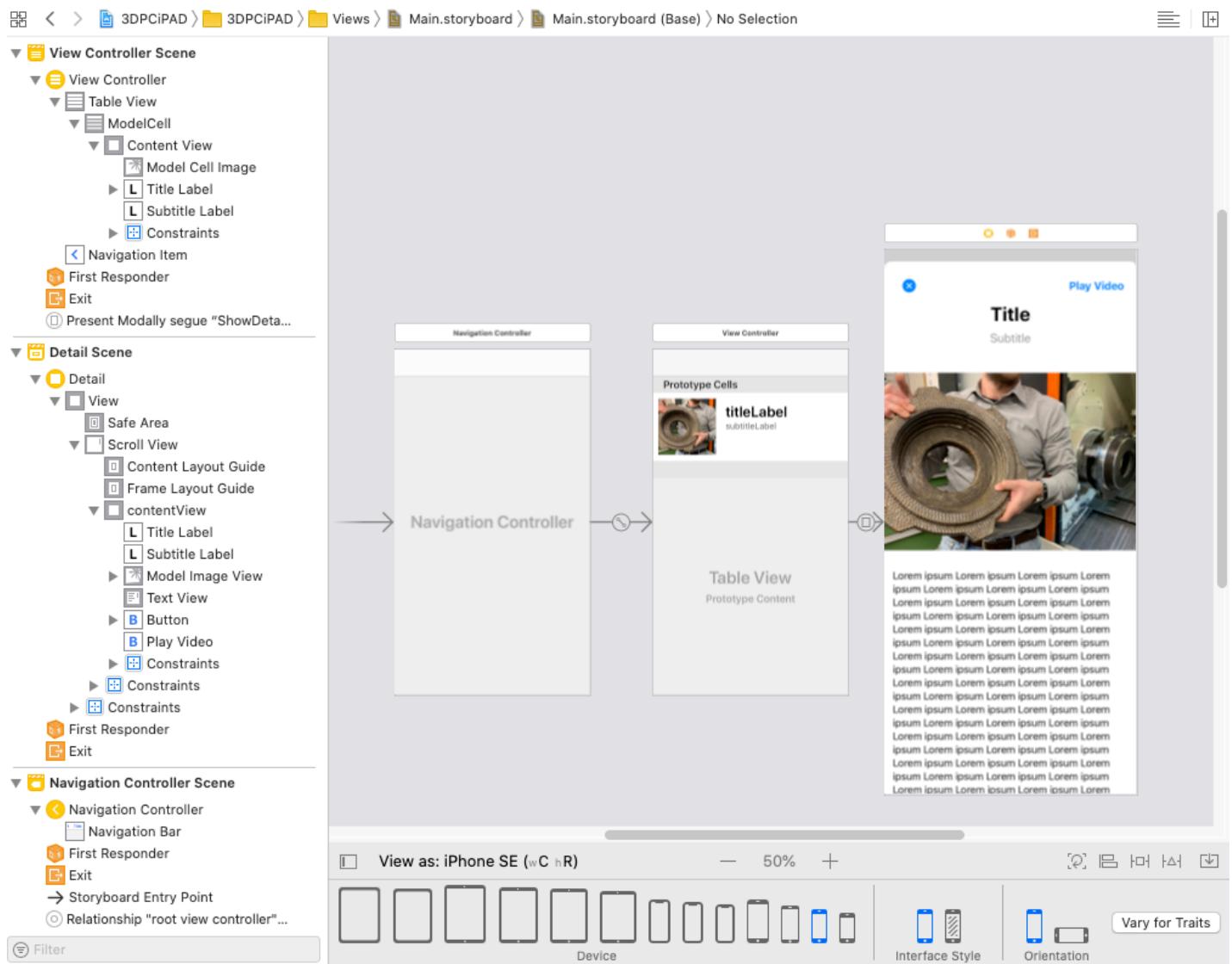


The screenshot shows the Xcode interface with the project '3DPCiPAD' open. The left sidebar displays the project structure, including '3DPCiPAD', '3DPCiPAD', 'supporting files', 'ViewControllers', 'DataModel', and 'models.json'. The 'models.json' file is selected and shown in the main editor area. The code in 'models.json' defines three objects, each representing a museum exhibit:

```
9  {
10     "title": "Amie1.0",
11     "subtitle": "Full-scale building enclosure prototypes.",
12     "image": "Amie1.0",
13     "contentText": "AMIE 1.0 is a 3D-printed full-scale building\nenclosure prototype. \\rThe structure was printed using\nBAAM technology and incorporates next-generation\nmodified atmosphere insulation panels. \\rThe structure\nis a mobile living space with the ability to produce,\nstore and share solar power with a 3D-printed hybrid\nvehicle to reduce or eliminate dependency on the\npower grid. \\rIt establishes for the first time a\nplatform for investigating solutions that integrate the\nenergy systems in buildings, vehicles, and the power\ngroup. \\rThe research utilizes advanced computational\ntools for design optimization, integrated project\ndelivery, rapid prototyping and fabrication of building\nelements using additive manufacturing. \\rThis concept\ncan potentially enhance the resiliency and\ncost-effectiveness of two major challenging demands in\nour cities worldwide namely housing and\ntransportation.\\r\\r"
14 },
15 {
16     "title": "AlpaPlaton",
17     "subtitle": "Swiss company specializing in the field of\nmedium format cameras",
18     "image": "AlpaPlaton",
19     "contentText": "ALPA Capaul and Weber Ltd. is a Swiss\ncompany specializing in the field of medium format\ncameras. \\rThe small-sized firm has been collaborating\nsince 2017 with the Product Development Group Zurich\n(pd|z) of ETH Zurich and inspire AG in various\ndevelopment projects involving Additive Manufacturing.\nIn their newest project, Platon, the interdisciplinary\nteam of students and professionals have developed an\naward-winning camera for videography professionals.\n\\rVarious components of the camera system are\nmanufactured with Selective Laser Sintering. \\rThe SLS\nparts are highly integrated, including electronics for\npower supply and cooling. \\rThanks to a digital process\nchain (3D scanning, custom-made ordering tools,\ninstructional CAD animations), ALPA's solutions are\ngreatly individualized to their customer needs. \\rTo sum\nup, the ALPA Platon camera demonstrates how AM and a\ndigital process chain enable traditional, small sized\ncompanies to offer highly individualized solutions to\ntheir customers.\\r\\r"
20 },
21 {
22     "title": "Aris",
23     "subtitle": "Measuring devices used in aviation for speed\nmeasurement",
24     "image": "Aris",
```

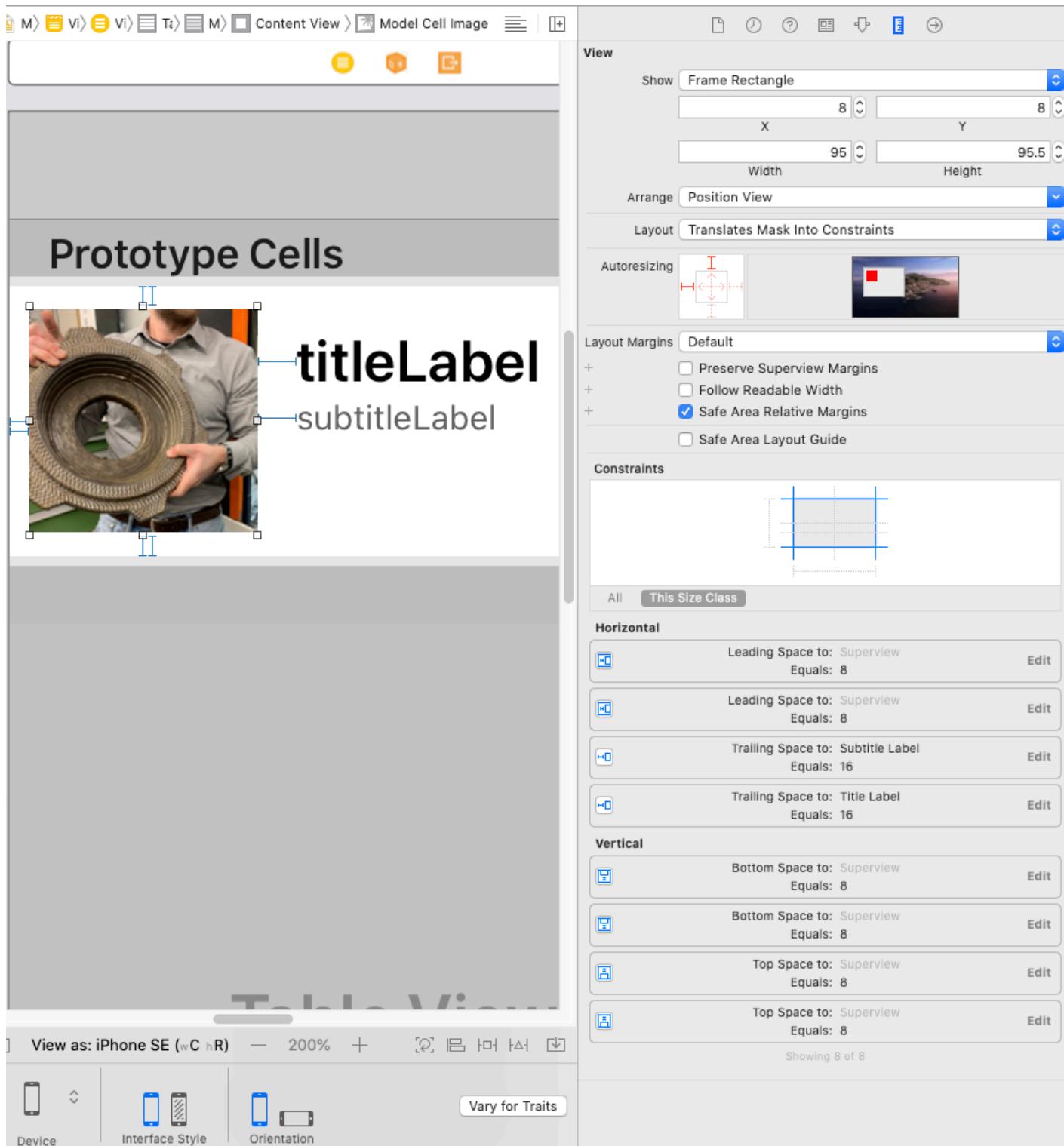
A4.3 main.storyboard

Man sieht drei View Controllers im dem Screenshot. Es sind der Navigation Controller, View Controller und Detail View Controller. Links im Navigator kann man dann sehen, dass jeweils viele Views untergeordnet sind. In dem View Controller in der Mitte hat man die Table View und ein Prototype für die Zelle. Rechts ist der Detail View Controller mit einer Pop Up View und Scroll View.

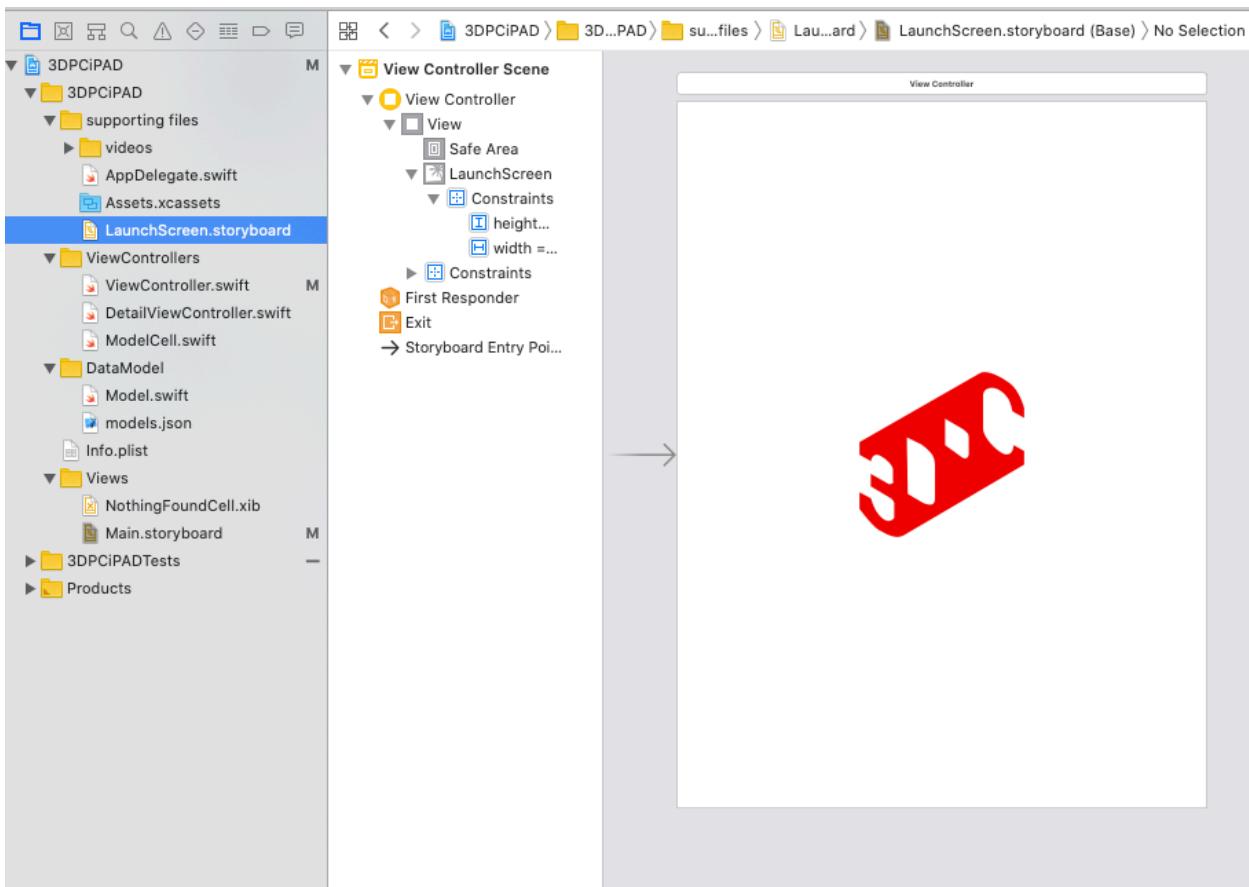


A4.4 Constraints Beispiel- Cell View Prototype

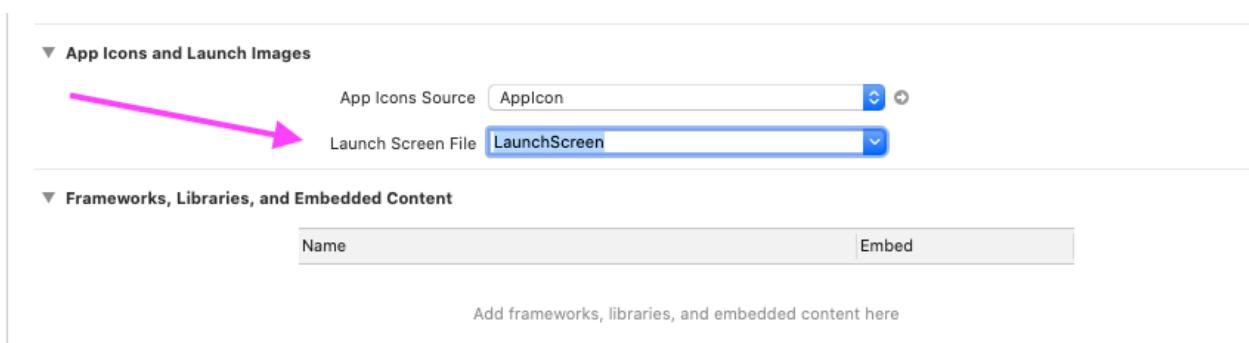
(Das Foto in dem "Prototype Cell" ist ein Placeholder)



A4.5 LaunchScreen.storyboard



Der LaunchScreen wird in die Settings gewählt



A5 - Quellcode

A5.1 Quellcode - ViewController.swift

```
import UIKit

class ViewController: UITableViewController, UISearchControllerDelegate{

    // initialize variables to contain my data
    private var models = [Model]()
    // this variable is for when I use the search
    private var filteredModels = [Model]()
    // I create a new UISearchController to add searching to my view
    // controller.
    private let searchController =
        UISearchController(searchResultsController: nil)

    override func viewDidLoad() {
        super.viewDidLoad()
        // assign a title to the property title of the navigation controller
        title = "Pushing Boundaries"
        // add a large title to the navigation bar
        navigationController?.navigationBar.prefersLargeTitles = true
        // add a search controller to the navigation bar
        createSearchController()
        // load the data
        getData()
        // register my custom nothing found cell!
        let cellNib = UINib(nibName: "NothingFoundCell", bundle: nil)
        tableView.register(cellNib,
                           forCellReuseIdentifier:"NothingFoundCell")
    }

    private func createSearchController() {
        // The search controller actually belongs as a property of the
        // navigation item of the view controller, which automatically places
        // it inside my navigation bar when the view controller is displayed.
        searchController.searchResultsUpdater = self
        searchController.obscuresBackgroundDuringPresentation = false
        searchController.searchBar.placeholder = "Type something here to
            search"
        searchController.definesPresentationContext = true

        // UISearchController needs iOS 11 min
        if #available(iOS 11.0, *) {
            self.navigationItem.searchController = searchController
        } else {
            self.tableView.tableHeaderView = self.searchController.searchBar
        }
    }
}
```

```
// MARK:- DATA
private func getData(){
    // I will first find my json file in my bundle
    if let path = Bundle.main.path(forResource: "models", ofType: "json")
    {
        // I get the json and transform in a data type
        do {
            let data = try Data(contentsOf: URL(fileURLWithPath: path))
            // if I can get the data from json I call my parse method
            parse(json: data)
        } catch {
            // if i cannot get the data an error is thrown so I catch it here
            print("error by getting data out of json")
        }
    }
}

private func parse(json: Data) {
    // I use a JSONDecoder object to convert the response data from the json file
    let decoder = JSONDecoder()

    if let jsonModels = try? decoder.decode(Models.self, from: json) {
        models = jsonModels.models
        // I use a custom sort as declared below
        models.sort(by: <)
    }
}

// MARK:- TableViews
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    // if nothing has been found
    if searchController.isActive && filteredModels.count == 0 &&
        searchController.searchBar.text != ""{
        return 1
    }
    // if i need to display the results of the search
    if searchController.isActive && searchController.searchBar.text != ""{
        return filteredModels.count
    } else { //all other cases display the whole array of models
        return models.count
    }
}
```

```
// Will return a cell to populate the table. The number of cells is
// defined above in numberOfRowsInSection
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

    // this will return my cell saying nothing found. if there are no
    // results, the method returns 1, for the row with the text "(Nothing
    // Found)". distinguish between "not searched yet" and "nothing
    // found".
    if searchController.isActive && filteredModels.count == 0 &&
        searchController.searchBar.text != "" {
        let cell = tableView.dequeueReusableCell(withIdentifier:
            "NothingFoundCell", for: indexPath)
        return cell
    }
    // this will return my result cells
    let cell = tableView.dequeueReusableCell(withIdentifier: "ModelCell",
        for: indexPath) as! ModelCell
    let model: Model
    if searchController.isActive && searchController.searchBar.text != "" {
        model = filteredModels[indexPath.row]
    } else {
        model = models[indexPath.row]
    }
    // configure is a cell method declared in ModelCell.swift
    cell.configure(for: model)
    return cell
}

// This will deselect the row after it has been selected and will perform
// the segue
override func tableView(_ tableView: UITableView, didSelectRowAt
    indexPath: IndexPath) {
    tableView.deselectRow(at: indexPath, animated: true)
    searchController.resignFirstResponder()
    performSegue(withIdentifier: "ShowDetail", sender: indexPath)
}

// This will disable the selection of a cell when the results are nil
override func tableView(_ tableView: UITableView, willSelectRowAt
    indexPath: IndexPath) -> IndexPath? {
    if searchController.isActive && filteredModels.count == 0 &&
        searchController.searchBar.text != "" {
        return nil
    } else {
        return indexPath
    }
}
```

```
// MARK:- Navigation
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "ShowDetail" {
        let detailViewController = segue.destination as!
            DetailViewController
        let indexPath = sender as! IndexPath
        if searchController.isActive && filteredModels.count != 0 &&
            searchController.searchBar.text != "" {
            let model = filteredModels[indexPath.row]
            detailViewController.model = model
        } else {
            let model = models[indexPath.row]
            detailViewController.model = model
        }
    }
}

// Add a conformance to UISearchResultsUpdating to my UIViewController
extension ViewController: UISearchResultsUpdating {
    // This method is required by the UISearchResultsUpdating protocol and
    // gets called every time the user types anything into the search bar,
    // so I can use the new text to filter my data however I want:
    func updateSearchResults(for searchController: UISearchController) {
        if let text = searchController.searchBar.text, !text.isEmpty {
            filteredModels = models.filter { model in
                model.title.localizedCaseInsensitiveContains(text)
            }
        } else {
            filteredModels = models
        }
        tableView.reloadData()
    }
}
```

A5.2 Quellcode - DetailViewController.swift

```
import UIKit
import AVKit

class DetailViewController: UIViewController {

    // This is the variable which gets allocated in the prepare for segue
    // method of the previous viewcontroller Visibility is internal by default
    // but not private!

    var model: Model!

    @IBOutlet private weak var titleLabel: UILabel!
    @IBOutlet private weak var subtitleLabel: UILabel!
    @IBOutlet private weak var modelImageView: UIImageView!
    @IBOutlet private weak var textView: UITextView!
    @IBAction private func playVideo(_ sender: UIButton) {
        let urlPath = Bundle.main.path(forResource: model.image, ofType:
    "mp4")!
        let videoURL = URL(fileURLWithPath: urlPath)
        let player = AVPlayer(url: videoURL)
        let vc = AVPlayerViewController()
        vc.player = player
        present(vc, animated: true) {
            vc.player?.play()
        }
    }

    // MARK:- Actions
    @IBAction private func close() {

        // this is the right way to dismiss the view controller. The
        // presenting vc is dismissing it not the presented, as explained by
        // Hegarty of Stanford

        presentingViewController?.dismiss(animated: true, completion:
            nil)
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        updateUI()
    }

    func updateUI() {
        titleLabel.text = model.title
        subtitleLabel.text = model.subtitle
        textView.text = model.contentText
        modelImageView.image = UIImage(named: "Placeholder")
    }
}
```

```
// the name of the large images have L appended to the name

modelImageView.image = UIImage(named: model.image + "L" + ".jpg")
// Dynamic Type for the content. iOS offers the option to enhance the
legibility of text by increasing font weight and setting the
preferred font size for apps. The user can open the Settings
app and navigate to General > Accessibility > Larger Text      to
access Dynamic Type text sizes and make te contents bigger
textView.font = .preferredFont(forTextStyle: .body)
//when the user returns from changing the text size settings, the app
should refresh the screen without needing an app restart. You can
do this by reloading the table view when the app receives a
UIContentSizeCategoryDidChange notification
textView.adjustsFontForContentSizeCategory = true
}

}
```

A5.3 Quellcode - ModelCell.swift

```
import Foundation

import UIKit

class ModelCell: UITableViewCell {

    @IBOutlet private weak var modelCellImage: UIImageView!
    @IBOutlet private weak var titleLabel: UILabel!
    @IBOutlet private weak var subtitleLabel: UILabel!

    // MARK:- Public Methods
    func configure(for model: Model) {
        titleLabel?.text = model.title
        subtitleLabel?.text = model.subtitle
        //This tells the UIImageView to load the image from the link and to
        //place it in the cell's image view
        modelCellImage?.image = UIImage(named: model.image)
        //make rounded corner
        modelCellImage?.layer.cornerRadius = 15

    }
}
```

A5.4 Quellcode - ModelCell.swift

```
struct Models: Codable {
    var models = [Model]()
}

struct Model: Codable {
    var title: String = ""
    var subtitle: String = ""
    var image: String = ""
    var contentText: String = ""
}

// sorting functions
func < (lhs: Model, rhs: Model) -> Bool {
    return lhs.title.localizedStandardCompare(rhs.title) == .orderedAscending
}

func > (lhs: Model, rhs: Model) -> Bool {
    return lhs.title.localizedStandardCompare(rhs.title)
        == .orderedDescending
}
```

A5.5 Git - .gitignore Datei in Terminal erstellen

```
1 git config --global alias.ignore '!gi()
2 { curl -L -s https://www.gitignore.io/api/$@ ;}; gi'
3
4 cd myProject
5 git ignore swift,macos >.gitignore
6 git add .gitignore
7 git commit -m "Add .gitignore file"
```