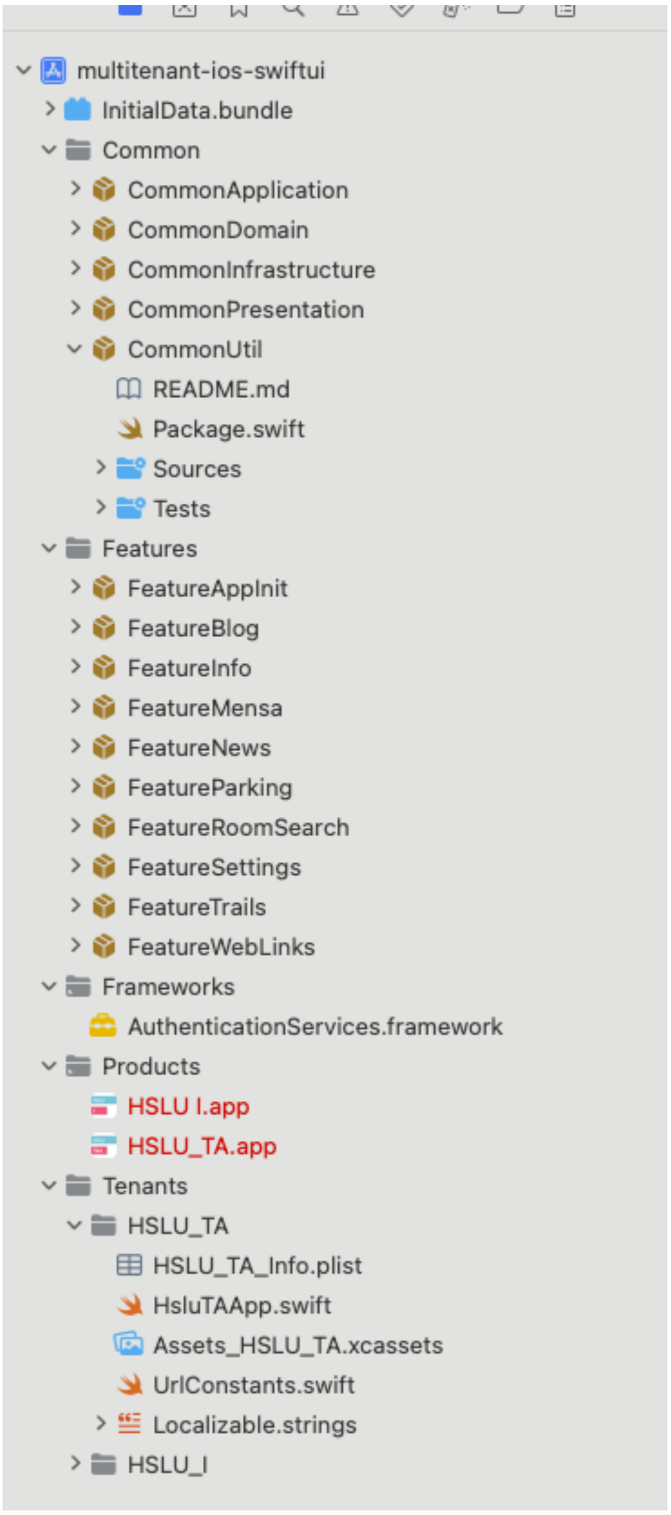
Aufgabenstellung

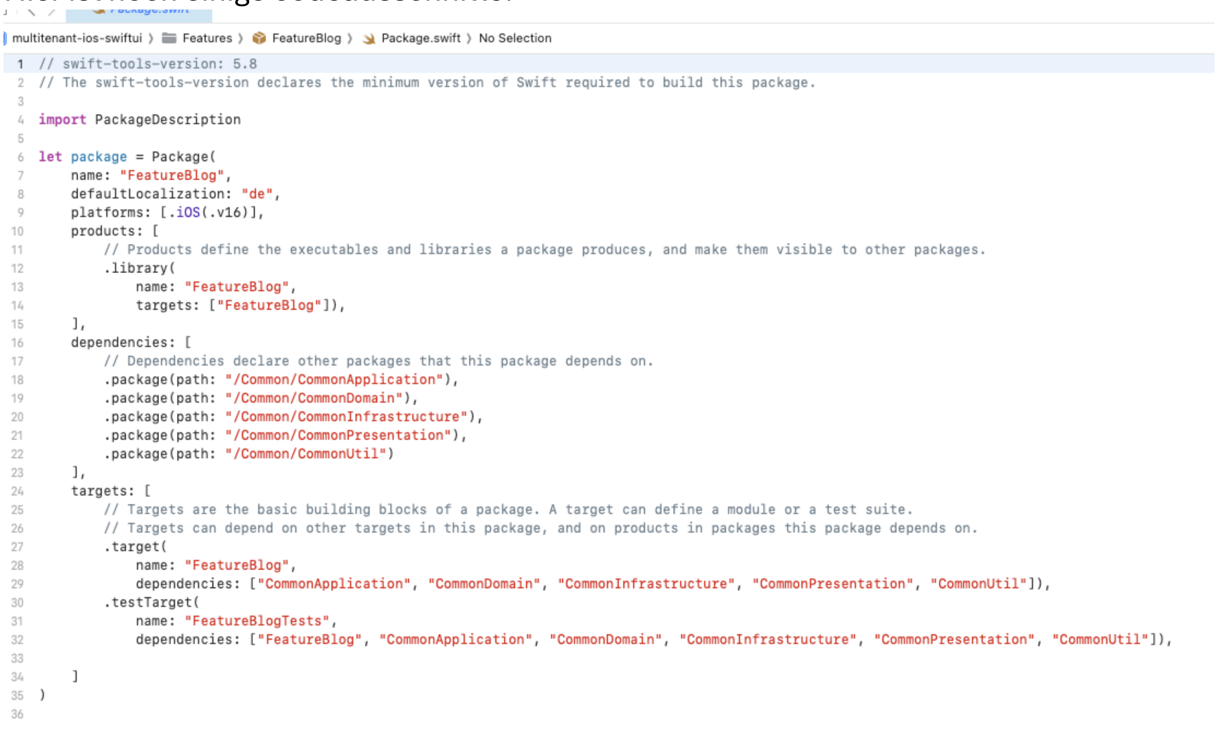
Wir müssen eine bestehende iOS APP, welche in Swift geschrieben ist, nach Android Jetpackcompose erstellen. Dazu soll falls sinnvoll Domain Driven Design verwendet werden.

Hier ist der Aufbau der iOS APP:



```
multitenant-ios-swiftui
  InitialData.bundle
  Common
    CommonApplication
    CommonDomain
    CommonInfrastructure
    CommonPresentation
    CommonUtil
      README.md
      Package.swift
      Sources
      Tests
  Features
    FeatureAppInit
    FeatureBlog
    FeatureInfo
    FeatureMensa
    FeatureNews
    FeatureParking
    FeatureRoomSearch
    FeatureSettings
    FeatureTrails
    FeatureWebLinks
  Frameworks
    AuthenticationServices.framework
  Products
    HSLU I.app
    HSLU_TA.app
  Tenants
    HSLU_TA
      HSLU_TA_Info.plist
      HsluTAApp.swift
      Assets_HSLU_TA.xcassets
      UrlConstants.swift
      Localizable.strings
    HSLU_I
```

Hier ist noch einige codeausschnitte:

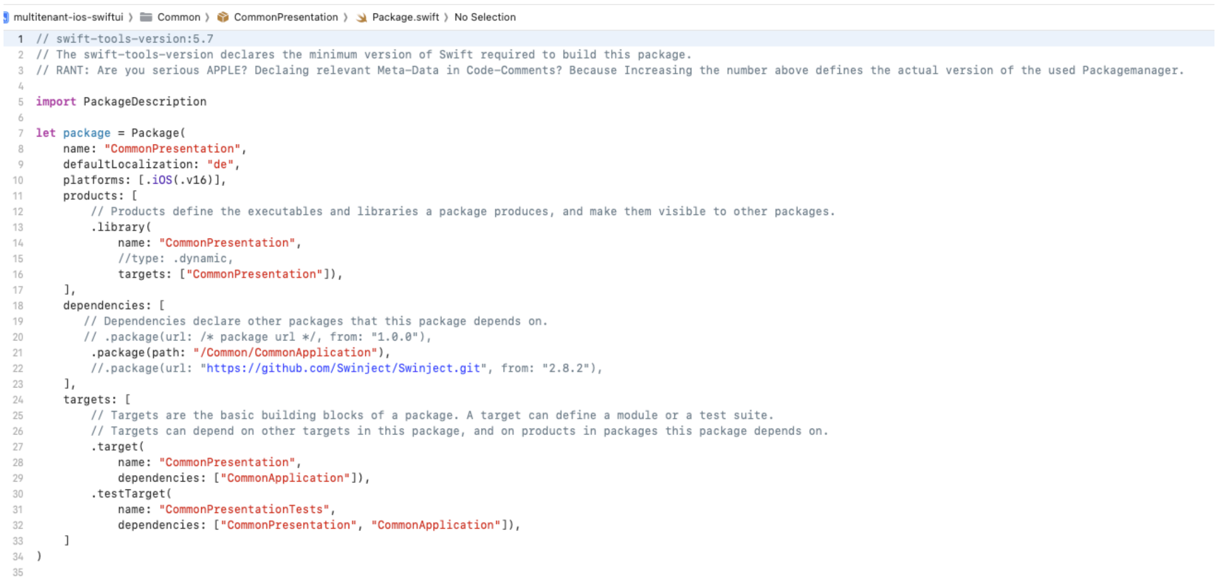

multitenant-ios-swiftui 〉 Features 〉 FeatureBlog 〉 Package.swift 〉 No Selection

```swift
// swift-tools-version: 5.8
// The swift-tools-version declares the minimum version of Swift required to build this package.

import PackageDescription

let package = Package(
    name: "FeatureBlog",
    defaultLocalization: "de",
    platforms: [.iOS(.v16)],
    products: [
        // Products define the executables and libraries a package produces, and make them visible to other packages.
        .library(
            name: "FeatureBlog",
            targets: ["FeatureBlog"]),
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(path: "/Common/CommonApplication"),
        .package(path: "/Common/CommonDomain"),
        .package(path: "/Common/CommonInfrastructure"),
        .package(path: "/Common/CommonPresentation"),
        .package(path: "/Common/CommonUtil")
    ],
    targets: [
        // Targets are the basic building blocks of a package. A target can define a module or a test suite.
        // Targets can depend on other targets in this package, and on products in packages this package depends on.
        .target(
            name: "FeatureBlog",
            dependencies: ["CommonApplication", "CommonDomain", "CommonInfrastructure", "CommonPresentation", "CommonUtil"]),
        .testTarget(
            name: "FeatureBlogTests",
            dependencies: ["FeatureBlog", "CommonApplication", "CommonDomain", "CommonInfrastructure", "CommonPresentation", "CommonUtil"]),

    ]
)
```


multitenant-ios-swiftui 〉 Common 〉 CommonPresentation 〉 Package.swift 〉 No Selection

```swift
// swift-tools-version:5.7
// The swift-tools-version declares the minimum version of Swift required to build this package.
// RANT: Are you serious APPLE? Declaing relevant Meta-Data in Code-Comments? Because Increasing the number above defines the actual version of the used Packagemanager.

import PackageDescription

let package = Package(
    name: "CommonPresentation",
    defaultLocalization: "de",
    platforms: [.iOS(.v16)],
    products: [
        // Products define the executables and libraries a package produces, and make them visible to other packages.
        .library(
            name: "CommonPresentation",
            //type: .dynamic,
            targets: ["CommonPresentation"]),
    ],
    dependencies: [
       // Dependencies declare other packages that this package depends on.
       // .package(url: /* package url */, from: "1.0.0"),
        .package(path: "/Common/CommonApplication"),
        //.package(url: "https://github.com/Swinject/Swinject.git", from: "2.8.2"),
    ],
    targets: [
        // Targets are the basic building blocks of a package. A target can define a module or a test suite.
        // Targets can depend on other targets in this package, and on products in packages this package depends on.
        .target(
            name: "CommonPresentation",
            dependencies: ["CommonApplication"]),
        .testTarget(
            name: "CommonPresentationTests",
            dependencies: ["CommonPresentation", "CommonApplication"]),
    ]
)
```

```swift
// Ressources: FileManger: https://www.appypie.com/filemanager-
files-swift-how-to/
// Async-Stuff:
https://stackoverflow.com/questions/71556293/how-can-i-avoid-
that-my-swift-async-method-runs-on-the-main-thread-in-
swiftui/73015435#73015435

import Foundation
import Network
import SwiftUI

import CommonDomain
import CommonInfrastructure
import CommonUtil
import OSLog


open class
CommonApplicationBaseModuleLoader<I:CommonAppBaseModuleItemAPI
DTO, T:CommonAppBaseModuleAPIDTO<I>,
C:CommonModuleBaseLoaderConfig> : ObservableObject
{
    @Published public var loadingStatus :
CommonModuleLoaderStatus<I> = .Not_Initialized()

    public var tenantConfig     : AppTenantConfig
= AppTenantConfig()
    public var appConfig        : AppModuleLoaderConfigProtocol
= CommonModuleBaseLoaderConfig()
    public var networkService    : CommonNetworkService
= CommonNetworkService()
    public var storageService    : CommonStorageService
= CommonStorageService()

    var logger = Logger(subsystem: "ch.hslu.mobileapp",
category: "hslui " + String(describing:
CommonApplicationBaseModuleLoader.self))

    public init() { }

    open func setup(tenantConfig : AppTenantConfig, appConfig: C,
storageService: CommonStorageService, networkService :
CommonNetworkService)
    {
        self.tenantConfig = tenantConfig
        self.appConfig = appConfig
        self.storageService = storageService
        self.networkService = networkService

        self.loadingStatus = .Initialized()
    }
    // 1) Does sync of remote / local data
    // 2) Does sync of fresh data / with local data, that was
ordered by the user
    // The last step may differ in another scenario ->
consider using it via dependency injection?
    open func process(implicit:Bool) async -> (Bool, String,
String, [I])? // AppNewsModuleItemAPIDTO
    {
        DispatchQueue.main.async { self.loadingStatus =
.Processing(["Syncing ... "]) }

        switch(self.appConfig.moduleLoaderType)
        {
            case "Hybrid":

                let result = await self.syncData()

                if (result.0)
                {
                    if !result.3.isEmpty
                    {
                        DispatchQueue.main.async {
self.loadingStatus = .Success(result.3) }
                        if(!implicit) { return (true,
result.1, result.2, result.3) }
                    }
                    else
                    {
                        DispatchQueue.main.async {
self.loadingStatus = .ErrorOn(["Data", "No Data"]) }
                        if(!implicit) { return (false, "Data",
"No Data", []) }
                    }
                }
                else
                {
                    DispatchQueue.main.async {
self.loadingStatus = .ErrorOn([result.1, result.2]) }
                    if(!implicit) { return (false, result.1,
result.2, []) }
                }

            case "Local":

                DispatchQueue.main.async { self.loadingStatus
= .ErrorOn(["", "Not implemented"]) }
                if(!implicit) { return (false, "", "Not
Implemented", []) }

            case "Remote":
                DispatchQueue.main.async { self.loadingStatus
= .ErrorOn(["", "Not implemented"]) }
                if(!implicit) { return (false, "", "Not
Implemented", []) }

            default :

                DispatchQueue.main.async { self.loadingStatus
= .ErrorOn(["", "Not implemented"]) }
                if(!implicit) { return (false, "", "Not
Implemented", []) }
        }

        return nil
    }

    // Does a reset of the module - data-wise
    // Deletes everything and re-triggers a process / sync
data of the module
    open func reset(implicit : Bool) async -> (Bool, String,
String, [I])? // AppNewsModuleItemAPIDTO
    {
        // do such things only when we guarantee network-
access...
        // in order to prevent cumbersome app-behaviour
        let hasConnection = await
self.networkService.hasConnection()
        let remoteHasData = await
self.networkService.isRemoteReachable(url:
self.tenantConfig.monitoringEndpoint)

        if(hasConnection && remoteHasData)
        {
            let syncFileDeleted = await
self.storageService.deleteFile(fileName:
self.appConfig.moduleLoaderSyncLocalFileName)
            let dataFileDeleted = await
self.storageService.deleteFile(fileName:
self.appConfig.moduleLoaderDataLocalFileName)

            if (syncFileDeleted && dataFileDeleted)
            {
                return await self.process(implicit: implicit)
            }
            else
            {
                return (false, "Data", "Unable to delete
Module-File", [])
            }
        }
        else
        {
            if(!remoteHasData)
            {
                return (false, "Network", "Remote Server is
not reachable!", [])
            }
            else
            {
                return (false, "Network", "No Network
connection available!", [])
            }
        }
    }

    // 1. Try to Get Last Update Timestamp
    //      If there is response -> see whether cached file is
available with same timestamp
    //      If there is no response -> see
    //      If there is no Connection -> try to grab local
version
    open func syncData() async -> (Bool, String, String, [I])
// AppNewsModuleItemAPIDTO
    {
        var remoteSyncData : (CommonAppBaseModuleSyncAPIDTO?,
Data)
        var remoteUpdateTS : String = ""
        var localUpdateTS : String = ""

        var canUpdate = false;
        var updateNeeded = false;
        var storeSyncInfo = false;

        if(await self.networkService.hasConnection())
        {
            // Get Update-Timestamp
            remoteSyncData = await
self.networkService.getAsJsonObject(url:
self.appConfig.moduleLoaderSyncURL) as
(CommonAppBaseModuleSyncAPIDTO?, Data)
            if(remoteSyncData.0 != nil)
            {
                remoteUpdateTS =
remoteSyncData.0!.ModuleLastUpdated
            }
            else
            {
                canUpdate = false; // Remote not available
            }

            // Get the File From Storage ->
            let localSyncData : CommonAppBaseModuleSyncAPIDTO?
= await self.storageService.getFile(fileName:
self.appConfig.moduleLoaderSyncLocalFileName)
            if(localSyncData != nil)
            {
                localUpdateTS =
localSyncData!.ModuleLastUpdated
            }

            if(localUpdateTS == "" && remoteUpdateTS == "")
            {
                canUpdate = false
                updateNeeded = true
            }
            if(localUpdateTS == "" && remoteUpdateTS != "")
            {
                canUpdate = true
                updateNeeded = true
                storeSyncInfo = true;
            }

            if(localUpdateTS != "" && remoteUpdateTS == "")
            {
                canUpdate = false
                updateNeeded = false // is not decideable
because remote info is missing, that can be compared to local
version
            }

            if(localUpdateTS != "" && remoteUpdateTS != "")
            {
                if(localUpdateTS == remoteUpdateTS)
                {
                    canUpdate = true
                    updateNeeded = false
                }
                else
                {
                    canUpdate = true
                    updateNeeded = true
                    storeSyncInfo = true
                }
            }

            if storeSyncInfo
            {
                let _ = await
self.storageService.storeFile(fileName:
self.appConfig.moduleLoaderSyncLocalFileName, rawData:
remoteSyncData.1)
            }
```

```
        }

        if canUpdate && updateNeeded
        {
            // else sync data from remote ...
            let (localTmpPath, _, _) = await 
self.networkService.getAsDownload(url: 
self.appConfig.moduleLoaderUrl) as (URL?, String, String)
            if localTmpPath != nil
            {
                let _ = await 
self.storageService.moveFile(fromFileName: localTmpPath!, 
toFileName: self.appConfig.moduleLoaderDataLocalFileName)
                let remoteData : T? = await 
self.storageService.getFile(fileName: 
self.appConfig.moduleLoaderDataLocalFileName)

                // self.loadingStatus = .Success
                return (true, "Data", "Getting Remote Data", 
remoteData!.List!)
            }
            else
            {
                return (false, "Data", "Unable to fetch remote 
data", [])
            }
        }
        else
        {
            if !canUpdate && updateNeeded
            {
                return (false, "Network", "Unable to fetch 
remote data", [])
            }
            else
            {
                // canUpdate && !updateNeeded
                // !canUpdate && !updateNeeded

                let localData : T? = await 
self.storageService.getFile(fileName: 
self.appConfig.moduleLoaderDataLocalFileName)
                if localData != nil
                {
                    // self.loadingStatus = .Success_Cached
                    // DispatchQueue.main.async { 
self.loadingStatus = .Success_Cached }
                    return (true, "Data", "Getting Cached 
Data", localData!.List!)
                }
                else
                {
                    return (false, "Data", "Unable to fetch local data", [])
                }
            }
        }
    }
}
```

Frage

Wie würdest du nun vorgehen und was ist wichitg zu beachten?

Kannst du die Aufgabe analysieren und in einem Word Dokument zusammenfassen wie wir Vorgehen sollten?