



장진호 - 신뢰할 수 있는 동료님을 응원해보세요!

지금 만족하고 있어요

장진호 - 신뢰할 수 있는 동료

iOS 개발자

기술 스택

iOS, Swift, iOS Autolayout, SwiftUI, 객체지향, tca

경력

주식회사 위대한상상

Senior iOS Developer / Staff Engineer | M&F

2022.05. ~ 2024.09. (2년 5개월)

- 검색기능 개발
- Server-driven UI 구조 설계 및 개발 리드
 - 단방향 데이터 흐름 구조 설계
 - UI 렌더링 성능 30% 향상
- Tuist를 활용한 프로젝트 구조 및 성능 향상
 - Code conflict 20% 감소
 - CI/CD 빌드 시간 50분에서 30분으로 감소
 - 앱 런치 타임 5초에서 3초로 감소 (아이폰 6 기준)
- 디자인 시스템 개발
- 로그 데이터 전송 효율성 개선
- Clean Architecture, N

추천해요 🍷

제안하기 ✉

댓글 💬

관련 스몰 워크샵 진행

WemeetPlace

iOS Developer

2021.03. ~ 2022.05. (1년 3개월)

- 메세징 기능 개발
- Layer 별 Class, Method 분리로 유지 보수성, Testability 향상
- 모듈화를 통한 협업 생산성 증가 및 build time 개선
- Profiling을 통한 성능 향상 (Memory usage 감소, Launch time 감소 등)
- Crash free rate 85%에서 99.95%로 향상

RoomFriends

iOS Developer

2019.09. ~ 2020.11. (1년 3개월)

- 룸프렌즈 iOS application a-z개발 및 런칭
- RxSwift, ReactorKit을 활용한 구조 개선
- Carthage를 활용한 Build time 개선

프로젝트

앱 리뉴얼 (서버드리븐 UI 구조 구축)

위대한상상

2023.08. ~ 2024.01.

서버드리븐 UI는 서버에서 내려준 데이터만으로 UI를 그려내는 구조를 이야기합니다. 이는 데이터가 약속된 형태(혹은 스키마) 범위 안에서 내려오기만 한다면 클라이언트는 모든 경우의 수를 커버하여 정확하게 UI를 그려내고 동작을 수행해 낼 수 있어야 한다는 것을 의미합니다. UI 데이터를 컴포넌트단위로 나누기는 했지만, 여전히 너무나 가변적인 데이터들을 매주 정적인 언어인 Swift로 대응해야 했습니다.

당시 가장 먼저 했던 일은 스펙을 제한시키고 합의하는 것이었습니다. (많은 과정이 있었으나 기술적인 내용은 아니므로 생략) 그리고 그와 동시에 서버드리븐 UI 기능이 올라갈 구조를 설계하게 되었습니다. 먼저는 UI를 컴포넌트 단위로 나누기로 했고, 이 컴포넌트는 화면의 넓이를 가득 채우는 직사각형 단위로 나눌 수 있었으므로 화면을 CollectionView, 각 컴포넌트를 Cell(혹은 Secion)에 대응시켰습니다. 이때, 우려되었던 점은 UI가 Controller혹은 Doamin layer와 직접 소통하는 부분이 많아지면 UI가 데이터를 받아야 할 수 있어야 하지만 너무나 많은 뷰 타입들이 ()가 매우 빠르게 올라갈

것이 자명해 보였습니다. 이러한 이유들로 인해 UI Layer에는 모델만 전달 할 수 있고, UI Layer가 직접적인 메시지를 보내진 못하도록 Layer간의 분리를 아주 명확하게 하는 단방향 구조를 설계하였습니다.

문제는 UI뿐만이 아니었습니다. 서버드리븐 UI 구조에서 클라이언트는 데이터를 UI로 그려내기만 할 뿐 비즈니스로직을 갖고있기 어려웠기 때문에 유저 액션에 따른 동작 수행을 어떻게 해야할지 방법을 찾아야 했습니다. 이는 필요한 기능을 딥링크 형태로 구현해 두어 서버에서 조절(혹은 호출) 가능하도록 하여 문제를 해결 하였는데, 만약 이 딥링크의 동작이 특정 상태값에 의존되기 시작하면 이부분도 유지보수가 매우 어려워질 것이 자명해보였습니다. 언제 어떤 상황에서 실행 가능해야하는 기능이 경우에 따라 서로 다른 상태값을 필요로 하게 되면 분명 꼬이는 지점이 발생할 것이라는 생각이었습니다. 이 부분은 딥링크 동작이 오직 path와 query parameter로만 결정되도록 순수 함수의 형태를 유지시키는 것으로 해결하였습니다.

이렇게 만든 극단적인 단방향 구조와 순수 함수형태의 기능 정의로 인해 안정적인 프로젝트를 만들 수 있었습니다.

이 과정중 추가로 Custom한 Diffable DataSource를 구현해 UI성능 향상도 이끌어 내기도 하였습니다.

Tuist 도입

위대한상상

입사 당시 요기요 iOS 프로젝트는 CocoaPod만을 사용하고 있었습니다. CocoaPod 사용 자체가 문제가 될 부분은 없지만, 매우 애플은 빌드머신(Github instance)의 성능으로 인해 CI/CD 빌드가 1시간 까지 걸리기도 하고, 프로젝트파일 conflict로 인한 혼선이 종종 발생하던 상황이었던지라 이부분의 개선 필요성을 느끼게 되었습니다. 마침 당시 iOS 조직 내에서는 모듈화에 관심이 많았던 터라 샘플 프로젝트를 제작해 팀원들을 설득해 나갈 수 있었고 Tuist도입을 진행하게 되었습니다.

Tuist는 Cocoapod을 지원하지 않기에 모든 외부 의존성에 Swift Package 지원을 추가하였고, 기능별로 모듈을 분리해 pre build를 활용하는 것으로 빌드 시간을 대폭 감소시킬 수 있었습니다. 또한 의존하는 모듈의 Mach-O type을 static으로 변경시키고 앱 시동시 수행하던 로직을 lazy하게 돌림으로써 cold launch time도 감소시킬 수 있었습니다.

로그 전송 효율화

위대한상상

요기요에서는 유저 행동에 대한 수많은 데이터를 실시간으로 수집하고 있었습니다. 문제는 모든 탭, 스크롤, 뷰의 노출이 200ms 이상 걸리는 구조였기 때문에 로그 전송이 지연되어 발생하는


댓글

것이었습니다. 이를 효율화 하고자 데이터를 묶어서 전송하는 프로젝트를 진행했으며, 상당한 효율화를 이끌어 낼 수 있었습니다.

다만 여러개의 로그를 한꺼번에 보내다보니 로그 누락의 이슈가 있었고, 이를 스토리지 캐싱으로 해결하였습니다.

포트폴리오

URL

<https://www.figma.com/board/MB6ig0TpxaVyq0d2ipCsu3/My-Problem-Solving-Hist...> 

교육

광운대학교

대학교(학사) | 전자재료공학
2013.03. ~ 2019.08. | 졸업

Powered by Rallit.