Организационные моменты для проекта

Шаблоны нашего проекта :

<https://rave.io/>

https://gowo.su/  
<https://timeweb.com/ru/community/articles/sovmestnyy-prosmotr-filmov-v-discord>

В первую очередь сделаем верстку/экраны

сайты с открытым API от куда будем брать сами фильмы и смотреть

направляем на прямую на API через JSON, мы как посредники будем транслировать

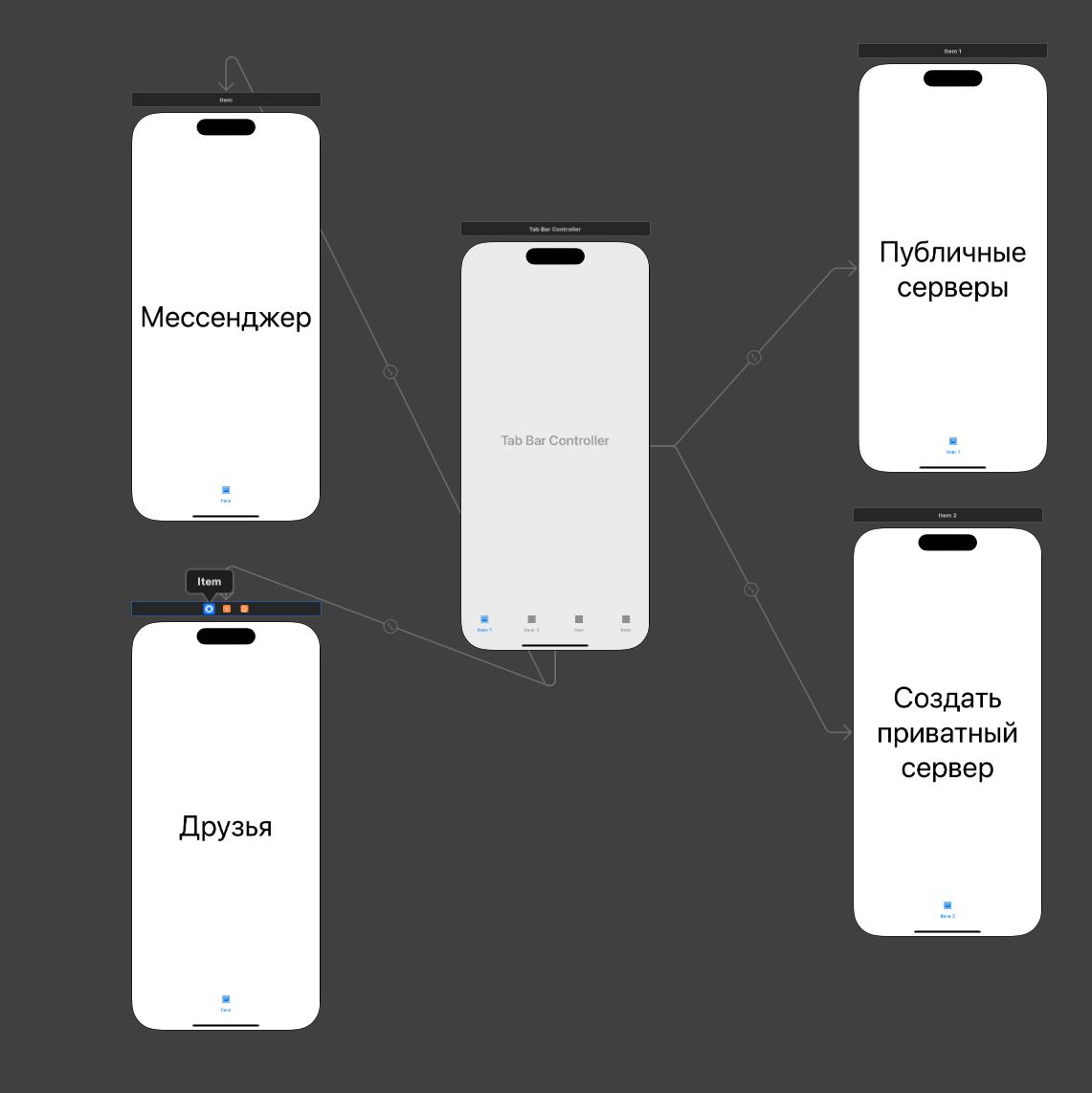
Сначала открывается экран с публичными серверами по умолчанию

Снизу таб бар с моими комнатами публичными серверами и чат   
сверху слева настройки  
справа сверху друзья

Публичные серверы и таб бар

Мои серверы

Настройки потом решим





Основные технологии  
  
Для реализации вашего проекта онлайн кинотеатра на SwiftUI с использованием MVP архитектуры, вам потребуются как клиентская, так и серверная части. Давайте рассмотрим подробнее, какие технологии могут пригодиться для каждой из них.

Клиентская часть (iOS приложение):

SwiftUI:

SwiftUI - это современный декларативный фреймворк для создания пользовательских интерфейсов на платформе iOS.

Он предоставляет множество инструментов для создания динамичных и красивых пользовательских интерфейсов.

Combine:

Combine - это фреймворк для обработки и управления потоками данных, включая обмен данными между вашим приложением и сервером.

Он предоставляет множество операторов для обработки потоков данных и обеспечения асинхронной связи.

SwiftUI Views:

Вы можете использовать различные виды SwiftUI Views для построения интерфейса вашего кинотеатра, такие как Text, Image, VStack, HStack, ZStack и т. д.

MVP архитектура:

MVP (Model-View-Presenter) - это шаблон проектирования, который разделяет ваше приложение на три основных компонента: Model, View и Presenter.

Model - это ваша бизнес-логика и данные.

View - это отображение пользовательского интерфейса.

Presenter - это прослойка между Model и View, которая обрабатывает взаимодействие между ними.

Серверная часть:

Node.js:

Node.js - это среда выполнения JavaScript на сервере, которая позволяет вам создавать быстрые и масштабируемые сервера.

Вы можете использовать Node.js для создания WebSocket сервера для обмена сообщениями в реальном времени между вашими клиентами.

WebSocket:

WebSocket - это протокол для обмена сообщениями между клиентом и сервером в реальном времени.

Он идеально подходит для сценариев, требующих постоянного обновления данных, таких как совместный просмотр фильмов.

Express.js:

Express.js - это минималистичный и гибкий веб-фреймворк для Node.js.

Он может использоваться для создания API для вашего приложения, обрабатывая запросы от клиентов и отправляя обновления состояния воспроизведения фильма через WebSocket.

База данных (например, MongoDB):

Вы можете использовать базу данных для хранения информации о комнатах, пользователях, текущем состоянии воспроизведения фильмов и т. д.

Общий механизм экрана:

Real-time Sync:

С использованием WebSocket вы можете обеспечить реально временную синхронизацию состояния воспроизведения фильма между всеми участниками.

Когда один пользователь выполняет действие (например, пауза, перемотка), это действие отправляется на сервер и ретранслируется всем остальным пользователям в комнате.

Общий интерфейс:

Создайте общий интерфейс, который будет обновляться на основе обновлений состояния воспроизведения, полученных от сервера.

Используйте модель прослушивания изменений для обновления пользовательского интерфейса в реальном времени в соответствии с действиями других пользователей.

Групповое управление:

Реализуйте механизм группового управления, который позволит всем пользователям в комнате управлять воспроизведением фильма (например, один пользователь нажимает "пауза", и фильм останавливается для всех).

Обработка конфликтов:

Обеспечьте механизм обработки конфликтов, если два или более пользователей пытаются выполнить действие одновременно (например, один пользователь нажимает "пауза", а другой нажимает "воспроизведение").

Это основные технологии и логика, которые могут быть использованы для реализации вашего проекта. Убедитесь, что вы также проведете дополнительное исследование и тестирование, чтобы обеспечить безопасность и производительность вашего приложения.

Просмотр фильма (варианты как транслировать фильм)  
  
Использование внешних источников:

Вместо загрузки видео на сервер вы можете использовать API внешних видеохостингов, таких как YouTube, Vimeo или других, которые предоставляют возможность встраивания видео.

Ваш сервер будет хранить только ссылку на видео или его идентификатор, а не само видео.

Прямая трансляция:

Вы можете настроить ваш сервер для прямой трансляции видео, когда оно запрашивается клиентом.

Это означает, что сервер будет динамически транслировать видео напрямую с источника, когда пользователь запрашивает его, вместо того чтобы хранить его на сервере.

Peer-to-Peer (P2P) технологии:

Вы можете рассмотреть возможность использования P2P технологий для совместного просмотра фильмов, где видео передается напрямую между участниками, минуя сервер.

Это может быть реализовано с использованием WebRTC или других P2P протоколов.

Использование стриминговых сервисов:

Вы можете интегрировать ваше приложение с существующими стриминговыми сервисами, такими как Netflix, Disney+, Hulu и т. д., позволяя пользователям смотреть фильмы, которые они уже имеют доступные через эти сервисы.

Синхронизация воспроизведения без видео на сервере:

Вместо трансляции видео на сервере вы можете сосредоточиться на синхронизации состояния воспроизведения между пользователями.

Каждый клиент будет запускать видео с собственного источника (например, локально на устройстве или с онлайн-платформы) и сервер будет отслеживать и синхронизировать состояние воспроизведения между ними.

Трудности

При разработке проекта онлайн кинотеатра совместного просмотра фильмов в реальном времени могут возникнуть различные трудности. Вот несколько из них и технологии, которые помогут их решить:

Синхронизация воспроизведения:

Трудность: Одной из основных задач будет обеспечение точной синхронизации воспроизведения фильма между всеми пользователями.

Решение: Использование WebSocket для обмена сообщениями в реальном времени и подходящего алгоритма синхронизации, такого как определение мастера в комнате или использование общего времени.

Масштабируемость:

Трудность: При увеличении числа активных комнат и пользователей может возникнуть необходимость масштабирования серверной части.

Решение: Использование облачных решений, таких как AWS, Google Cloud или Microsoft Azure, для автоматического масштабирования серверных ресурсов в зависимости от нагрузки.

Безопасность:

Трудность: Необходимо обеспечить безопасность передачи данных и защиту от атак, таких как перехват данных или взлом сервера.

Решение: Использование HTTPS для шифрования данных, аутентификация и авторизация пользователей, защита от инъекций и другие практики безопасности, а также регулярное обновление и мониторинг безопасности сервера.

Оптимизация производительности:

Трудность: Обеспечение плавного и быстрого просмотра фильмов для пользователей при загрузке видео и передаче данных.

Решение: Оптимизация кода клиентского и серверного приложения, использование CDN для ускорения загрузки видео и ресурсов, кэширование данных и оптимизация сетевого взаимодействия.

Поддержка разных устройств:

Трудность: Обеспечение работы приложения на различных устройствах и платформах (iOS, Android, веб и т. д.).

Решение: Использование кросс-платформенных фреймворков, таких как React Native или Flutter, или разработка отдельных версий приложения для разных платформ с общим API для обмена данными с сервером.

Управление комнатами и пользователями:

Трудность: Необходимо эффективно управлять созданием, присоединением и завершением комнат, а также управлением пользователями внутри них.

Решение: Реализация соответствующего API на сервере для управления комнатами и пользователями, использование средств баз данных для хранения информации о комнатах и пользователях, а также управление жизненным циклом комнат и пользователей.

Кроме того, важно учитывать правовые аспекты, такие как авторские права на контент и соблюдение законодательства о защите данных пользователей. Необходимо также провести тестирование приложения на различных устройствах и сценариях использования, чтобы обнаружить и исправить возможные проблемы до выпуска в продакшн.