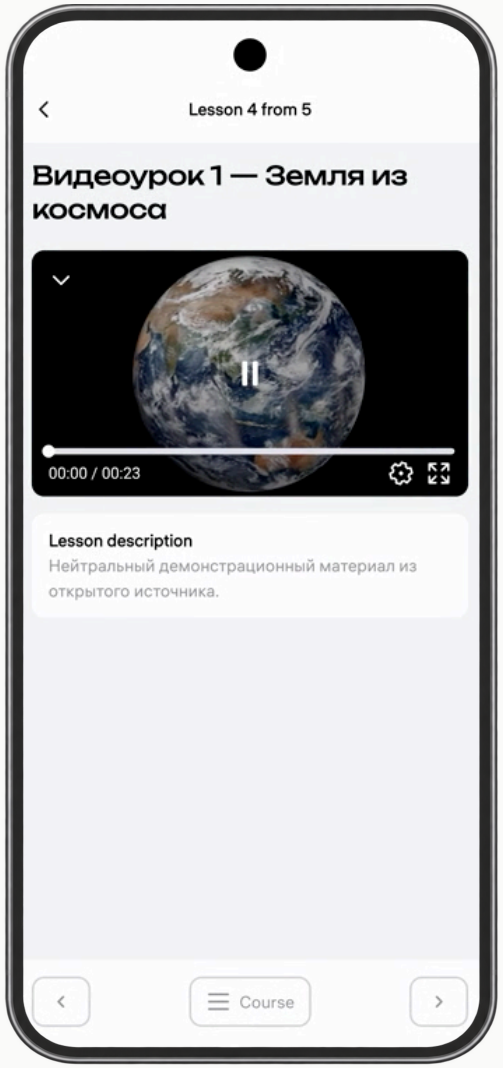
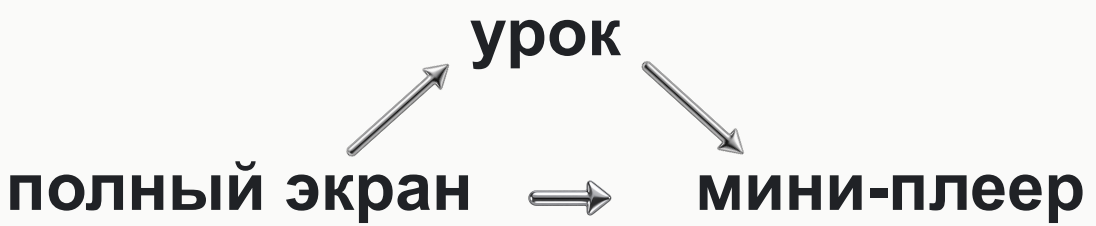
KMP Player:
нативный опыт без дублирования логики



Мальцева Ангелина · независимый консультант

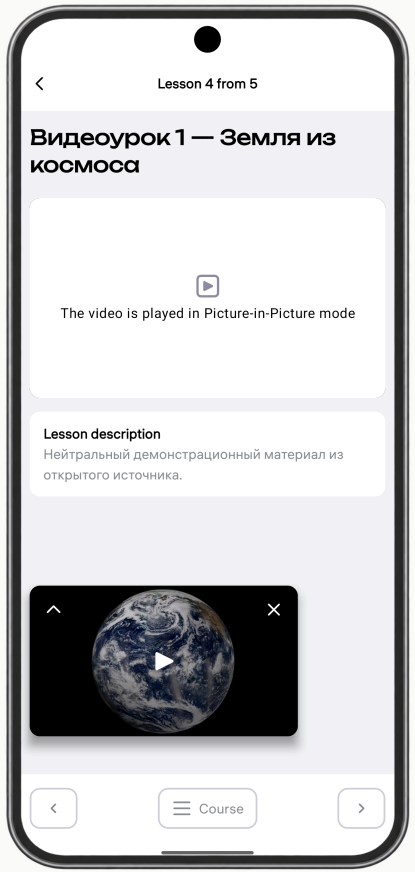
Пользовательский маршрут





При смене режима сохраняем материал и позицию

Виды мини-плеера



Мини-плеер приложения

Android



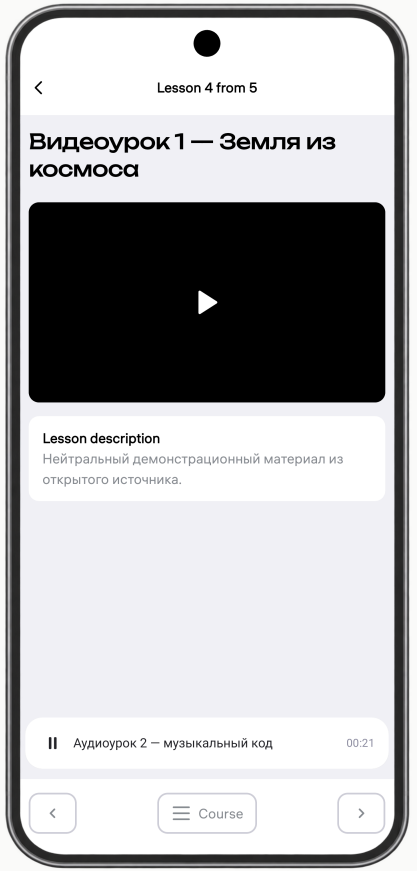
Системный PiP

Android



Системный PiP

iOS



Аудио мини-плеер

iOS и Android

Что разберем

01 управление и состояние

02 взаимодействие с системой

03 режимы видео и возврат к уроку

04 общая логика в KMP

Что разберем

01 управление и состояние

02 взаимодействие с системой

03 режимы видео и возврат к уроку

04 общая логика в KMP

Результат

Как собрать плеер с нативными возможностями и общей логикой интерфейса



Требования к плееру

Один активный материал

Несколько режимов отображения

Поддержка аудио и видео

Возврат к текущему уроку

Требования к плееру

Один активный материал

Несколько режимов отображения

Поддержка аудио и видео

Возврат к текущему уроку

ОСНОВА РЕШЕНИЯ

ANDROID

Media3 • ExoPlayer

IOS

AVPlayer • AVKit

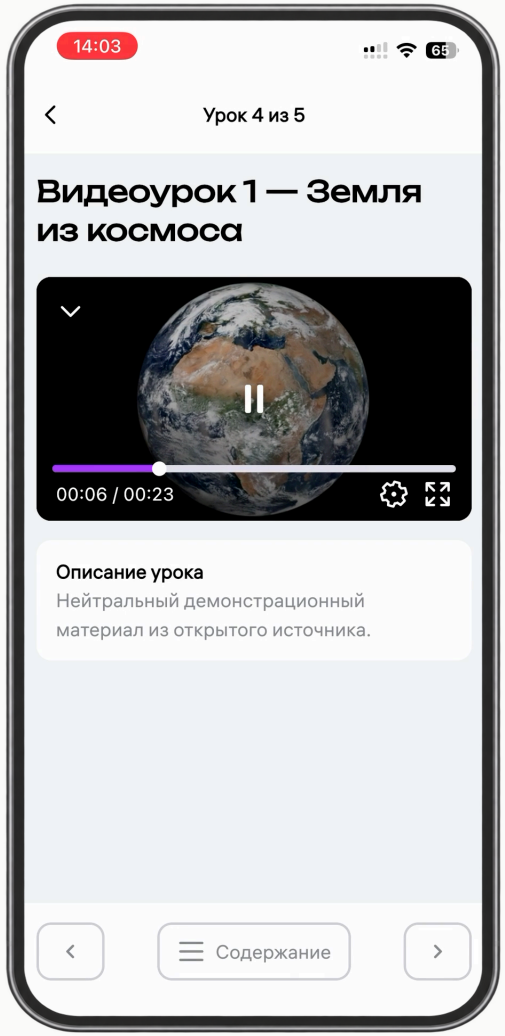
COMMON

Kotlin • Compose Multiplatform



Управление и состояние

Скорость



- Можно изменить скорость воспроизведения
- Если материал играл, воспроизведение **продолжается**
- При паузе после смены скорости **пауза сохраняется**

Android: меняем скорость

СМЕНА СКОРОСТИ

KOTLIN · Android

Задаём скорость воспроизведения

```kotlin
fun setSpeed(speed: PlayerSpeed) {
    mediaController?.setPlaybackSpeed(speed.value)
}
```

iOS: меняем скорость, сохраняя паузу

01 / СМЕНА СКОРОСТИ

iOS / Kotlin

Пауза сохраняется

```
fun setSpeed(speed: PlayerSpeed) {
    if (isPlaying()) {
        avPlayer?.rate = speed.value
    }
    updateSpeedAndPlayingStatus(speed)
}
```

rate меняем только внутри условия isPlaying()

iOS: запускаем с выбранной скоростью

02 / НАЖАТИЕ PLAY

iOS / Kotlin

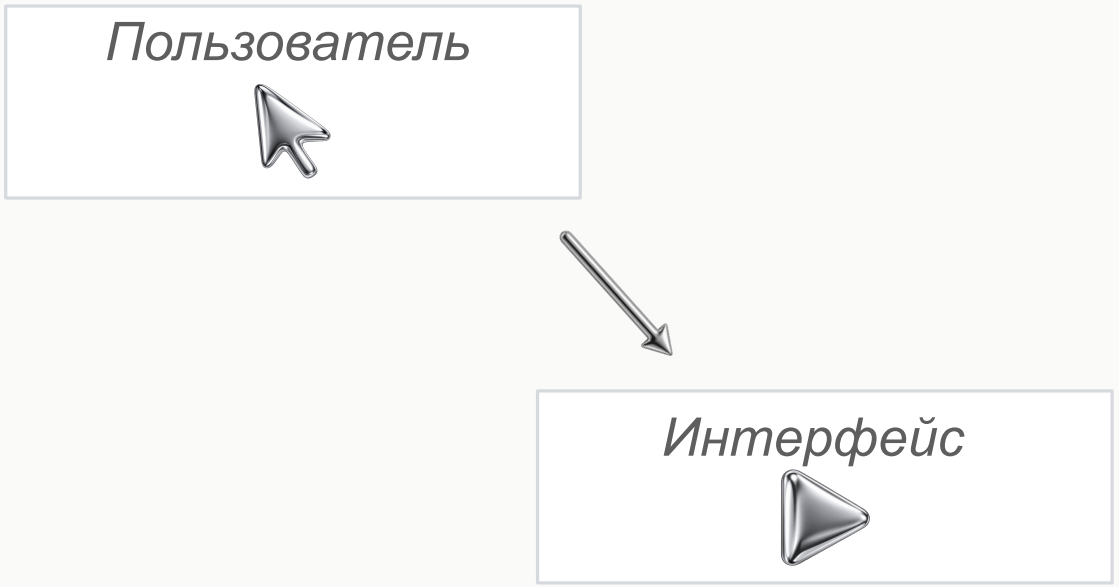
Применяем выбранную скорость

```
private fun playWithSpeed() {
    val currentSpeed =
        getCurrentState()?.speed ?: PlayerSpeed.X1

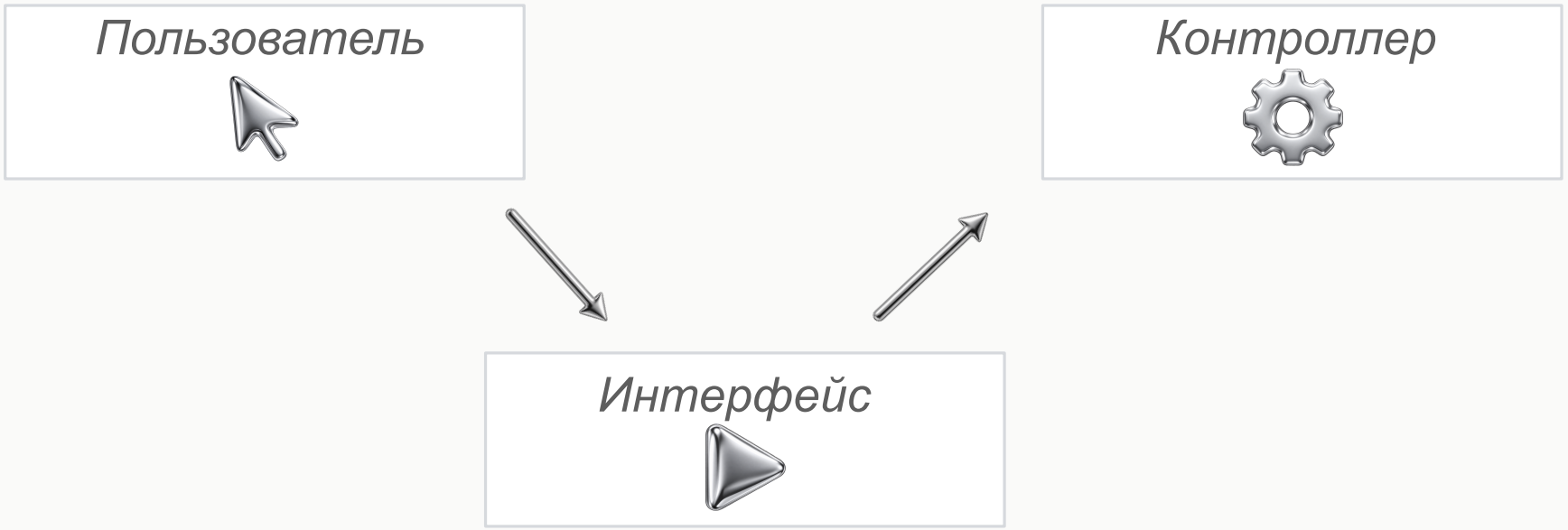
    avPlayer?.rate = currentSpeed.value
}
```

Берём сохранённую скорость и передаём её в AVPlayer

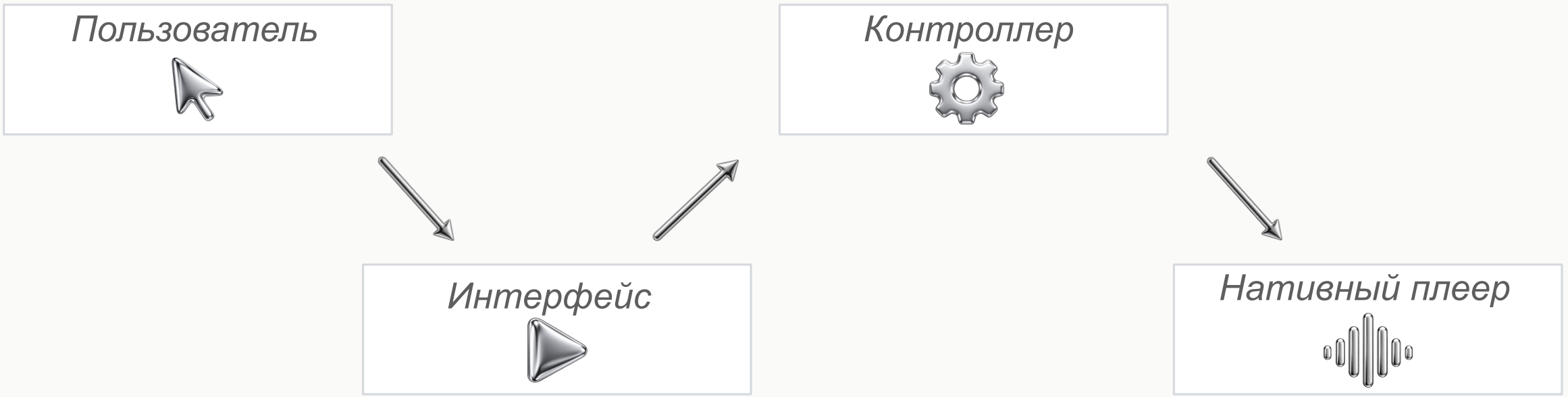
Управление плеером



Управление плеером



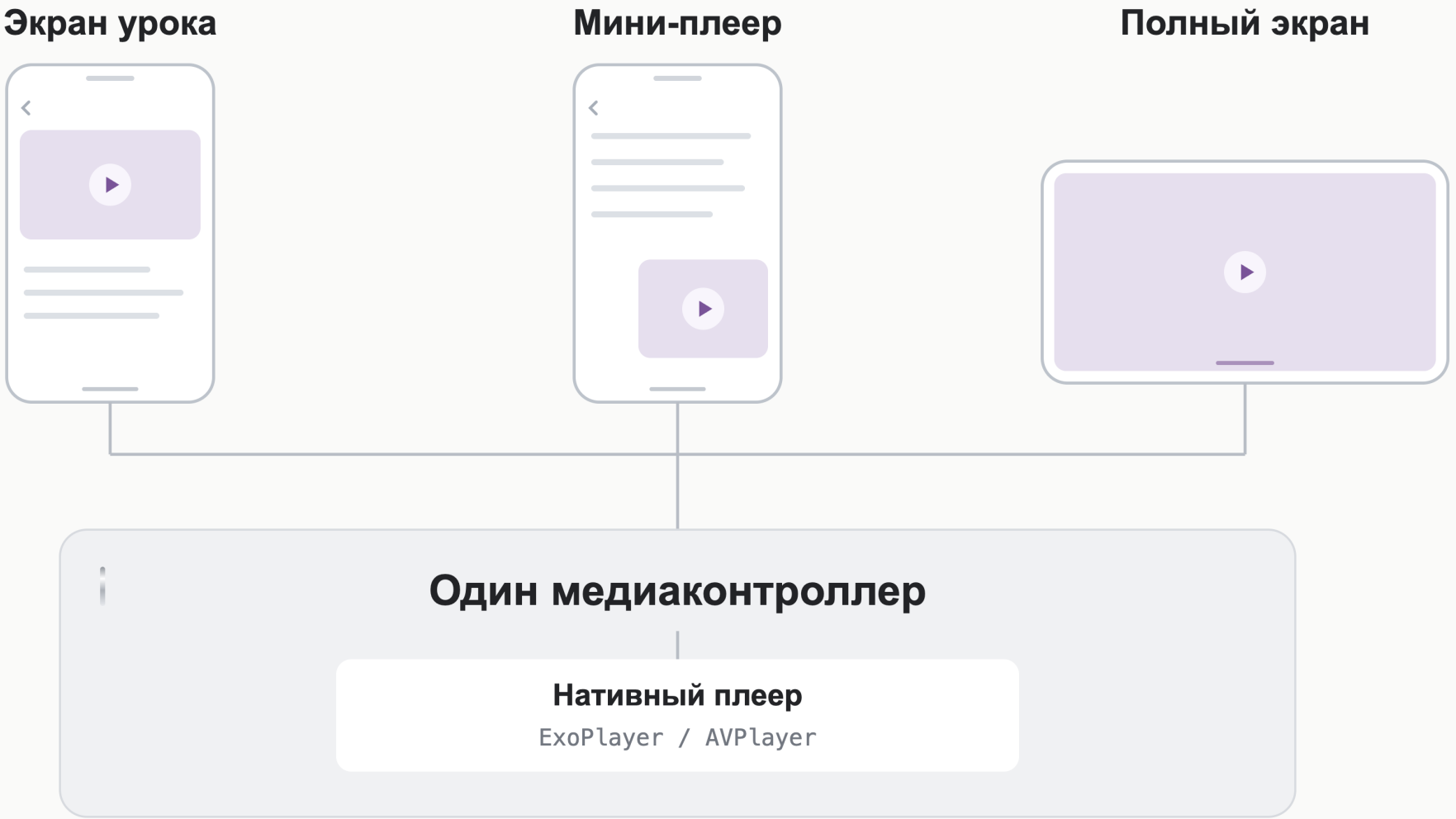
Управление плеером



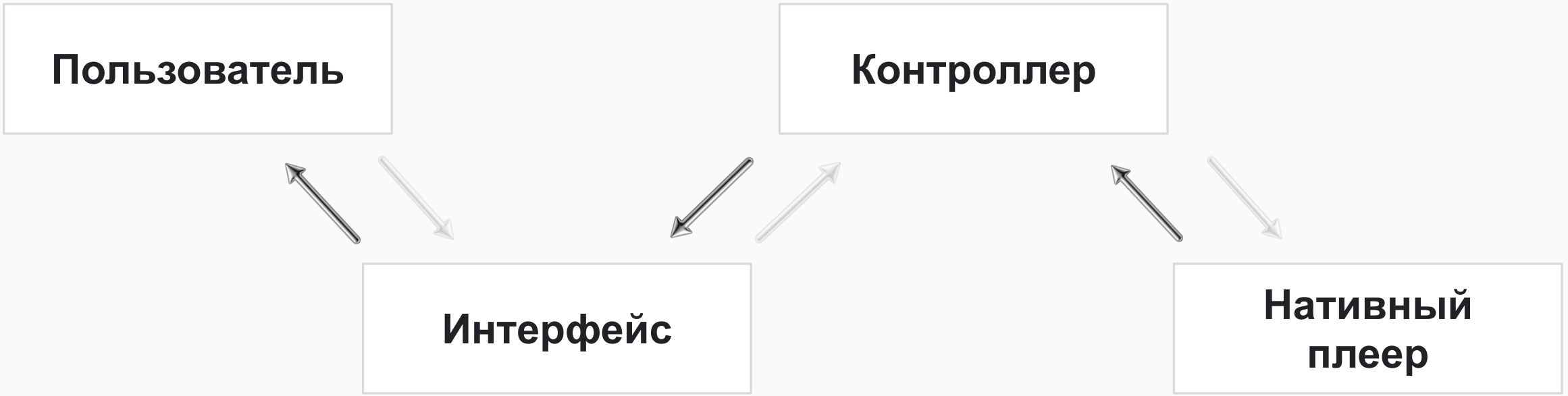
Управление плеером



Жизнь плеера не зависит от UI



Управление плеером



Android: подключаем Listener

СОСТОЯНИЕ ПЛЕЕРА

KOTLIN · Android

Готовый Player.Listener

```kotlin
val localListener = object : Player.Listener {
    override fun onIsPlayingChanged(isPlaying: Boolean) {
        updatePlayerState()
        ...
    }

    ...
}

mediaController?.addListener(localListener)
```

iOS: наблюдаем свойства

СОСТОЯНИЕ ПЛЕЕРА

SWIFT · поясняющий пример

Создаём наблюдателя

```
class PlayerObserver: NSObject {
    override func observeValue(
        forKeyPath keyPath: String?,
        of object: Any?,
        change: [NSKeyValueChangeKey: Any]?,
        context: UnsafeMutableRawPointer?
    ) {
        // Здесь обрабатываем изменения
    }
}
```

iOS: наблюдаем свойства

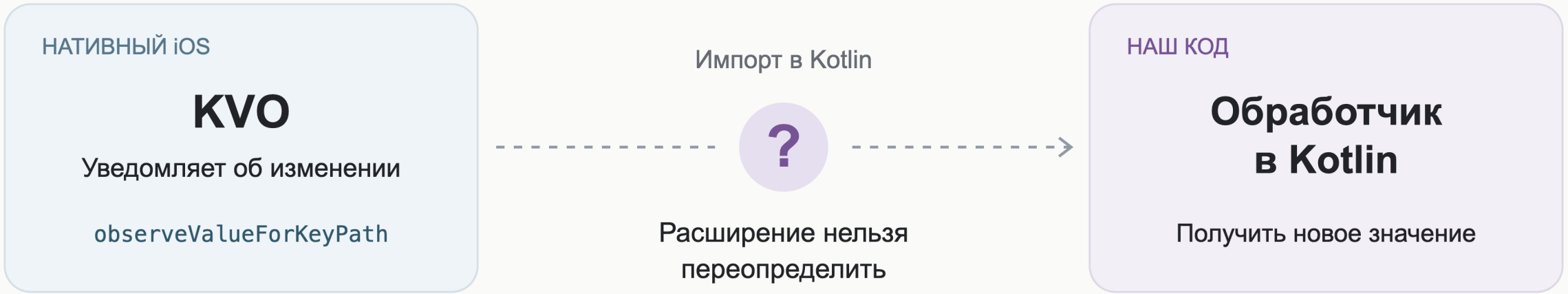
СОСТОЯНИЕ ПЛЕЕРА

KOTLIN · iOS

Пробуем реализовать callback

```
private val itemStatusObserver: NSObject =
    object : NSObject() {
        override fun observeValueForKeyPath(
            keyPath: String?,
            ofObject: Any?,
            change: Map<Any?, *>?,
            context: COpaquePointer?,
        ) {
            // Здесь обрабатываем изменения
        }
    }
```

Обработчик KVO нельзя просто переопределить



Мост к KVO: объявляем протокол

СОСТОЯНИЕ ПЛЕЕРА

OBJECTIVE-C · cinterop

Объявляем контракт callback

```
#import <Foundation/Foundation.h>

@protocol NSKeyValueObserving
@required
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary<NSKeyValueChangeKey, id> *)change
                       context:(void *)context;
@end
```

nskeyvalueobserving.def · код проекта

Реализуем наблюдателя в Kotlin

СОСТОЯНИЕ ПЛЕЕРА

KOTLIN · iOS · шаг 1

Реализуем callback

```
private val itemStatusObserver: NSObject =
    object : NSObject(), NSKeyValueObservingProtocol {
        override fun observeValueForKeyPath(
            keyPath: String?,
            ofObject: Any?,
            change: Map<Any?, *>?,
            context: COpaquePointer?,
        ) {
            // Здесь обрабатываем изменения
        }
    }
```

Подписываемся на изменения

СОСТОЯНИЕ ПЛЕЕРА

KOTLIN · iOS · шаг 2

Подключаем наблюдателя

```
avPlayer?.addObserver(
    itemStatusObserver,
    forKeyPath = "rate",
    options = NSKeyValueObservingOptionNew,
    context = null,
)
```

Передаём изменения в обработчик

СОСТОЯНИЕ ПЛЕЕРА

KOTLIN · iOS · шаг 3

Разбираем изменения

```
val newValue = change?.get(NSKeyValueChangeNewKey)

when (keyPath) {
    "rate" -> handleRate(newValue as? Float)

    else -> Unit
}
```

Текущая позиция воспроизведения



Android: запускаем периодический опрос

ПОЗИЦИЯ ВОСПРОИЗВЕДЕНИЯ

KOTLIN · Android

Запускаем периодический опрос

```
updateTimeJob?.cancel()
updateTimeJob = coroutineScope.launch(Dispatchers.Main) {
    while (isActive) {
        val positionMs = mediaController?.currentPosition
        delay(DEBOUNCE_UPDATE_TIME_MS)
    }
}
```

iOS: подключаем time observer

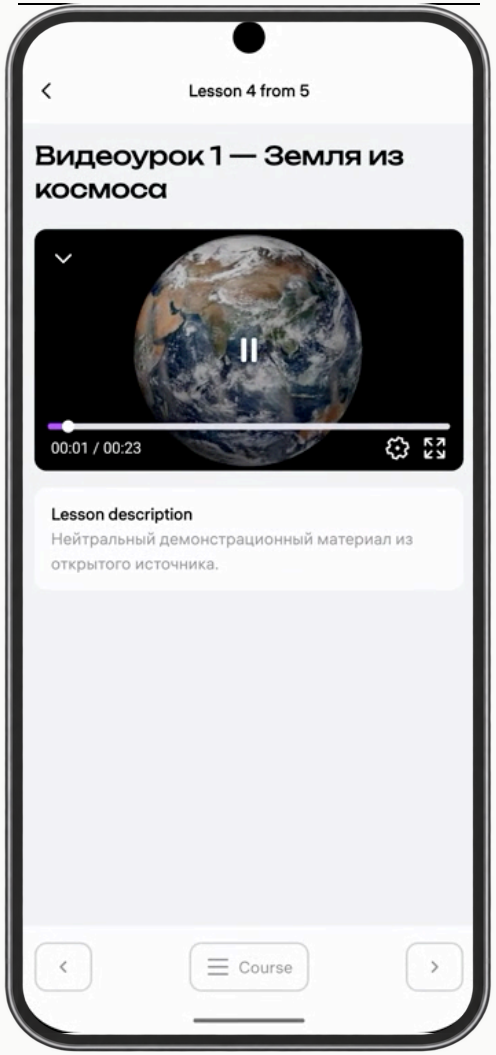
ПОЗИЦИЯ ВОСПРОИЗВЕДЕНИЯ

KOTLIN · iOS

```
private fun setupTimeObserver() {
    removeTimeObserver()

    timeObserver = avPlayer?.addPeriodicTimeObserverForInterval(
        interval = CMTimeMakeWithSeconds(1.0, 1),
        queue = null,
    ) { time ->
        updateCurrentTime()
    }
}
```

Во время перемотки позицию выбирает пользователь



ПОЗИЦИЯ ВОСПРОИЗВЕДЕНИЯ

KOTLIN · commonMain

Во время жеста сохраняем выбор

```
currentTime = if (uiState.isSliding) {
    uiState.currentTime
} else {
    controllerState.currentTime
}
```



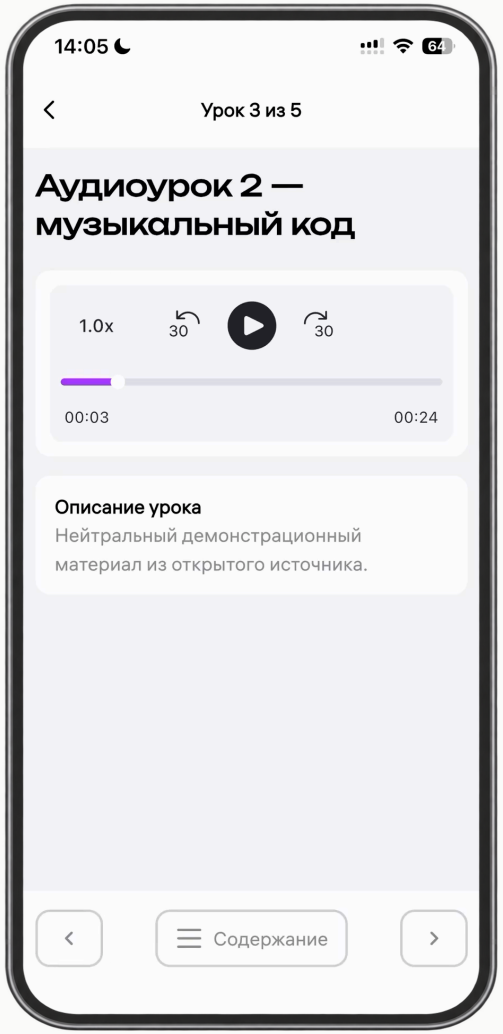
Общий интерфейс управления

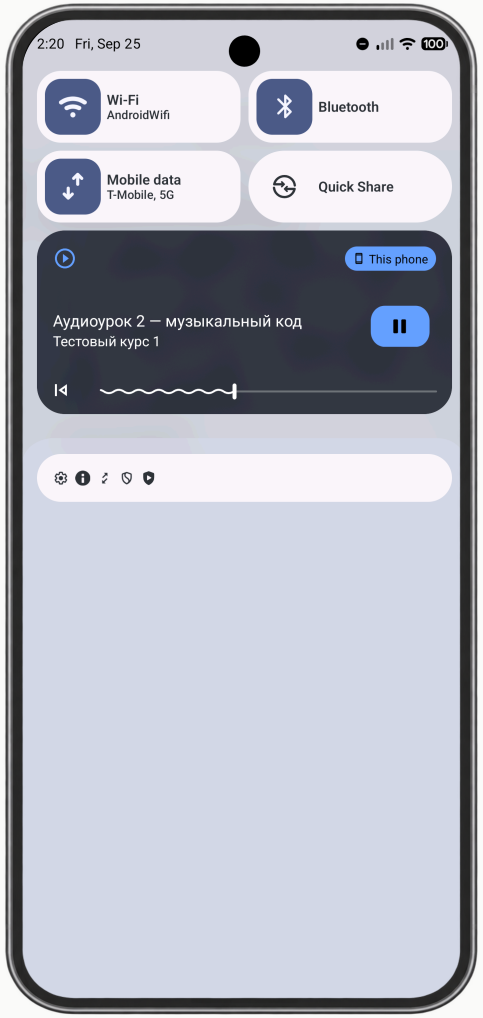
Интерфейс управляет плеером и получает его состояние через общий контроллер



Взаимодействие с системой

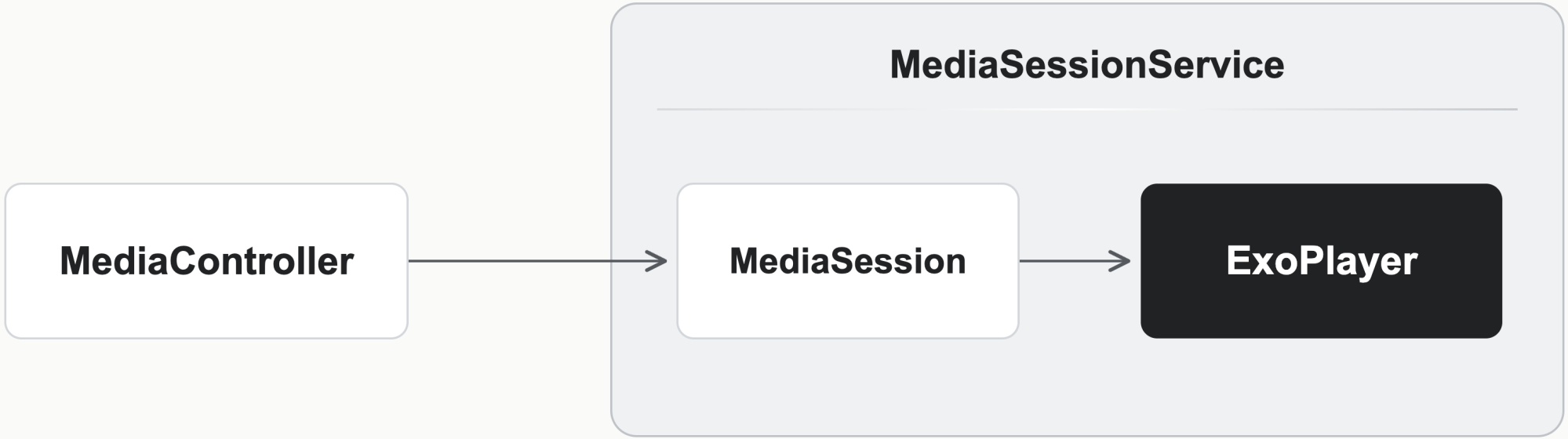
Управление вне приложения





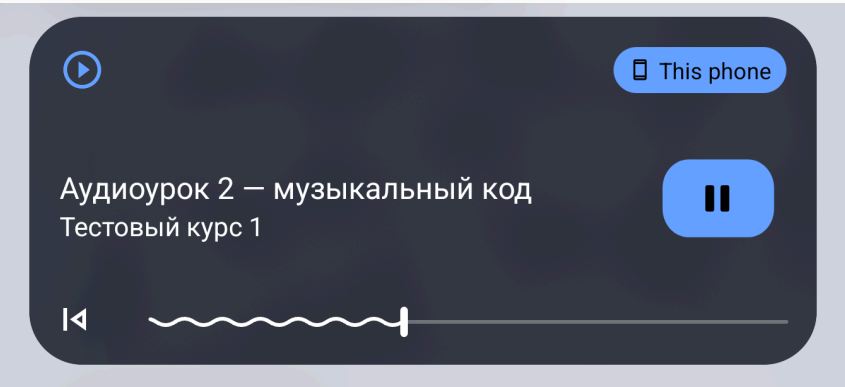
Управляем через систему, а состояние изменяется в приложении

Android: плеером владеет сервис



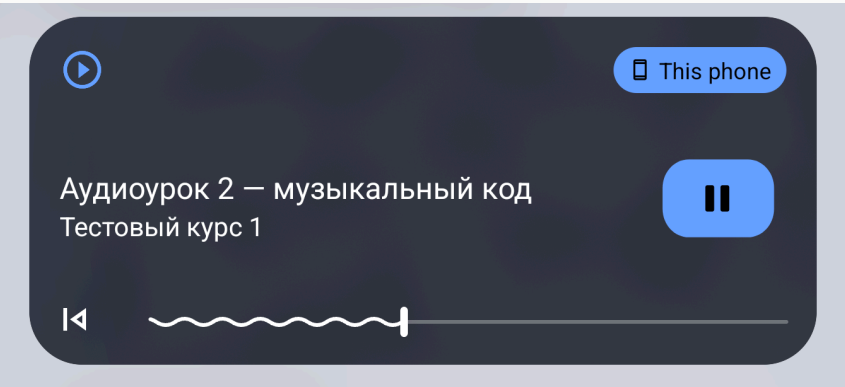

```
MediaSession.Builder(this, exoPlayer).build()
```

Android: системное уведомление




```
val mediaItem = MediaItem.fromUri(media.url)
mediaController.setMediaItem(mediaItem)
```

Android: системное уведомление




```
val mediaItem = MediaItem.fromUri(media.url)
    .buildUpon()
    .setMediaMetadata(
        MediaMetadata.Builder()
            .setTitle(media.title)
            .setArtist(media.subtitle)
            .build()
    )
    .build()

mediaController.setMediaItem(mediaItem)
```

iOS: фоновое воспроизведение

- **`Background Audio`** — включаем фоновое воспроизведение

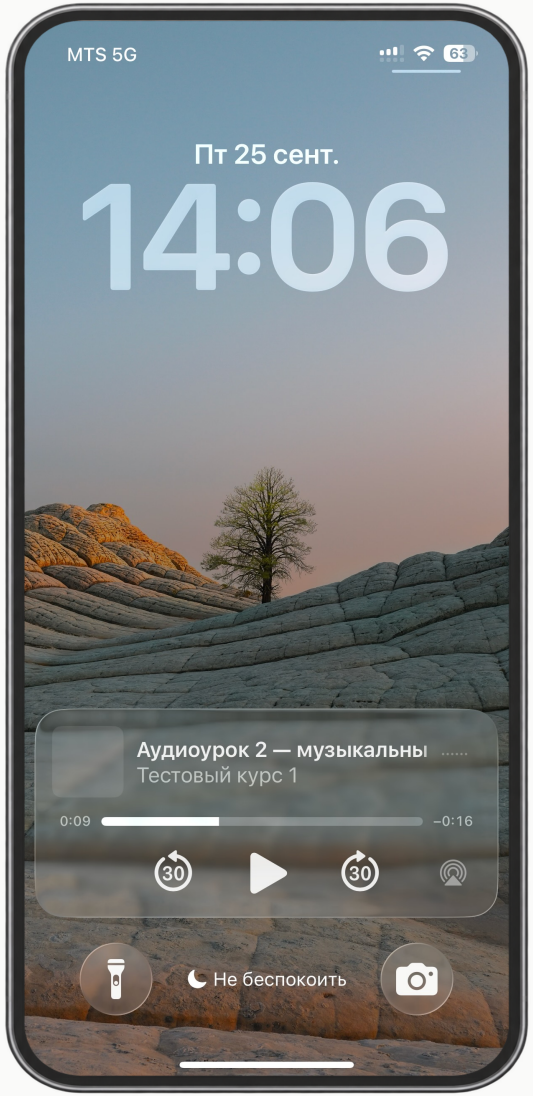
```
<key>UIBackgroundModes</key>
<array><string>audio</string></array>
```

iOS: фоновое воспроизведение

- `Background Audio` — включаем фоновое воспроизведение
- **`AVAudioSession`** — настраиваем звук

```
val session = AVAudioSession.sharedInstance()
session.setCategory(AVAudioSessionCategoryPlayback, error = null)
```

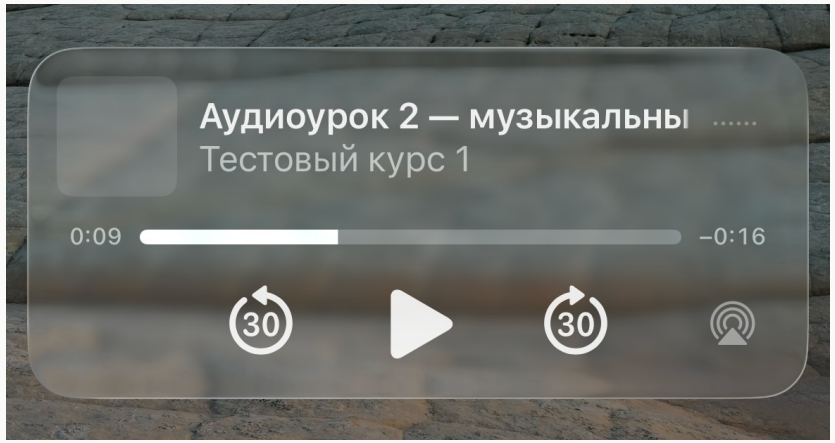
Управление из системной панели




```
val commandCenter = MPRemoteCommandCenter.sharedCommandCenter()

commandCenter.playCommand.addTargetWithHandler { _ ->
    this.play()
    MPRemoteCommandHandlerStatusSuccess
}
```

Управление из системной панели




```
// Один раз при создании плеера
nowPlayingSession = MPNowPlayingSession(
    players = listOf(avPlayer),
).apply {
    automaticallyPublishesNowPlayingInfo = true
}

// При смене материала
val playerItem = AVPlayerItem(uRL = mediaUrl).apply {
    nowPlayingInfo = mapOf(
        MPMediaItemPropertyTitle to media.title,
        MPMediaItemPropertyArtist to media.subtitle,
    )
}

avPlayer.replaceCurrentItemWithPlayerItem(playerItem)
```

Когда система прерывает воспроизведение

СИСТЕМНЫЕ ПРЕРЫВАНИЯ

KOTLIN · Android

Делегируем обработку audio focus

```kotlin
val exoPlayer = ExoPlayer.Builder(this)
    // …
    .setAudioAttributes(AudioAttributes.DEFAULT, true)
    // …
    .build()
```

Когда система прерывает воспроизведение

СИСТЕМНЫЕ ПРЕРЫВАНИЯ

KOTLIN · iOS

AVAudioSession сообщает о начале и завершении прерывания

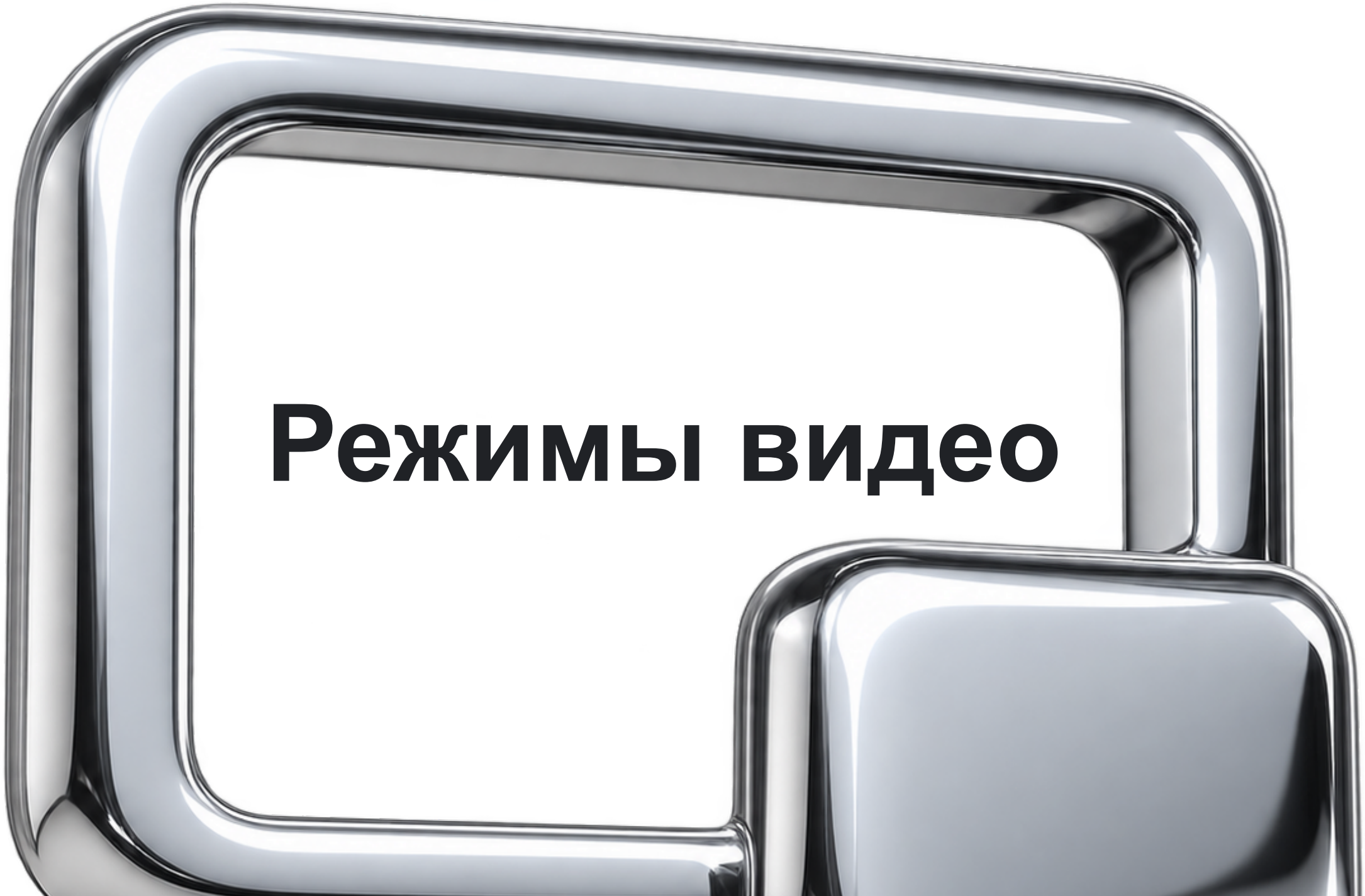
```
focusObserver = NSNotificationCenter.defaultCenter.addObserverForName(
    name = AVAudioSessionInterruptionNotification,
    `object` = null,
    queue = null,
) { notification ->
    val type = notification?.userInfo
        ?.get(AVAudioSessionInterruptionTypeKey) as? NSNumber

    when (type?.unsignedLongValue) {
        AVAudioSessionInterruptionTypeBegan -> {
            // Прерывание началось
        }
        AVAudioSessionInterruptionTypeEnded -> {
            // Прерывание завершилось
        }
    }
}
```



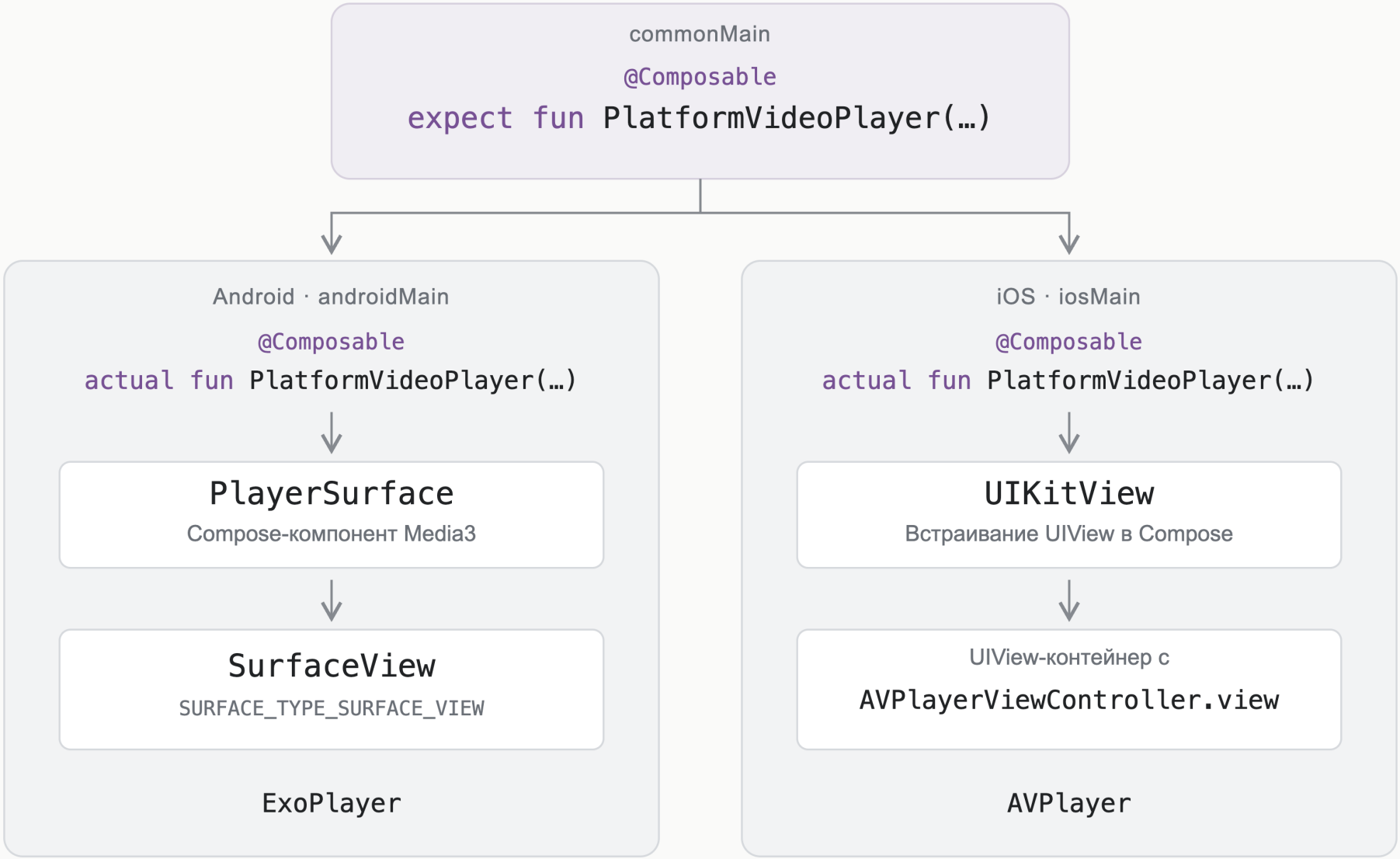
Воспроизведение независимо от экрана

Уход из приложения не прерывает прослушивание и не лишает управления

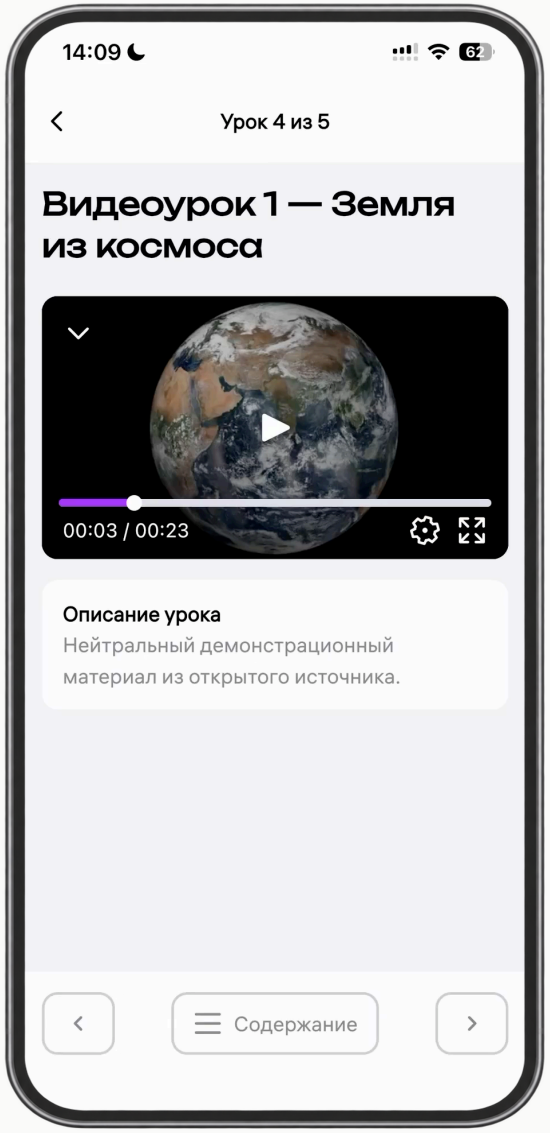


Режимы видео

Как видео встроено в общий интерфейс



iOS: PiP через AVKit



РЕЖИМЫ ВИДЕО

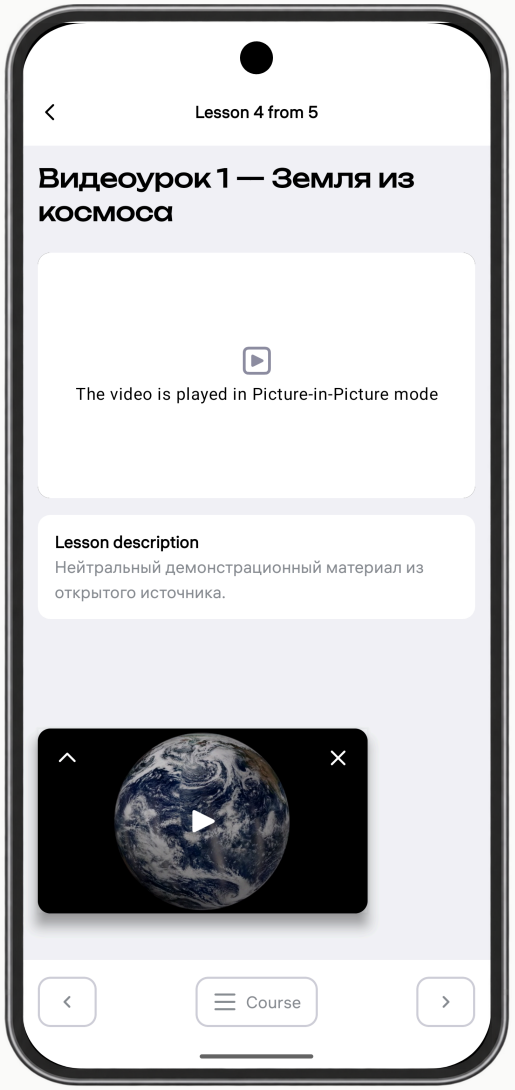
KOTLIN · iOS

Подключаем PiP через AVKit

```
val localPlayer = mediaController.getPlayer() ?: return UIView()

val newAvPlayerViewController = AVPlayerViewController().apply {
    player = localPlayer
    // …
    allowsPictureInPicturePlayback = true
    canStartPictureInPictureAutomaticallyFromInline = true
    delegate = this@VideoUiController.delegate
    // …
}
```

Android: мини-плеер внутри приложения



- Плавающее окно поверх экранов приложения
- Положение и контролы контролируем сами

Android: включаем поддержку PiP

AndroidManifest.xml

XML · Android

Разрешаем PiP для MainActivity

```xml
<activity
    android:name=".MainActivity"
    android:supportsPictureInPicture="true"
    android:configChanges=
        "screenSize|smallestScreenSize|screenLayout|orientation">
    <!-- … -->
</activity>
```

Поддержка PiP + обработка изменений конфигурации

Android: за пределами приложения системный PiP



РЕЖИМЫ ВИДЕО

KOTLIN · Android

Activity переходит в системный PiP

```kotlin
class MainActivity : ComponentActivity() {
    // …
    override fun onUserLeaveHint() {
        super.onUserLeaveHint()
        val state = videoPlayerController.playerState.replayCache.lastOrNull()

        if (state?.showBackgroundPip() == true) {
            enterPictureInPictureMode(
                PictureInPictureParams.Builder().build(),
            )
        }
    }
    // …
}
```

Android: Как узнаём о смене режима



MAINACTIVITY · РЕЖИМЫ ВИДЕО

KOTLIN · Android

Система сообщает о смене режима PiP

```kotlin
override fun onPictureInPictureModeChanged(
    isInPictureInPictureMode: Boolean,
    newConfig: Configuration,
) {
    super.onPictureInPictureModeChanged(
        isInPictureInPictureMode, newConfig,
    )

    if (isInPictureInPictureMode) {
        // Вошли в PiP
    } else {
        // Вышли из PiP
    }
}
```

true — вошли в PiP · false — вышли из PiP

iOS: Как узнаём о смене режима

РЕЖИМЫ ВИДЕО

KOTLIN · iOS

AVKit сообщает о переходах PiP

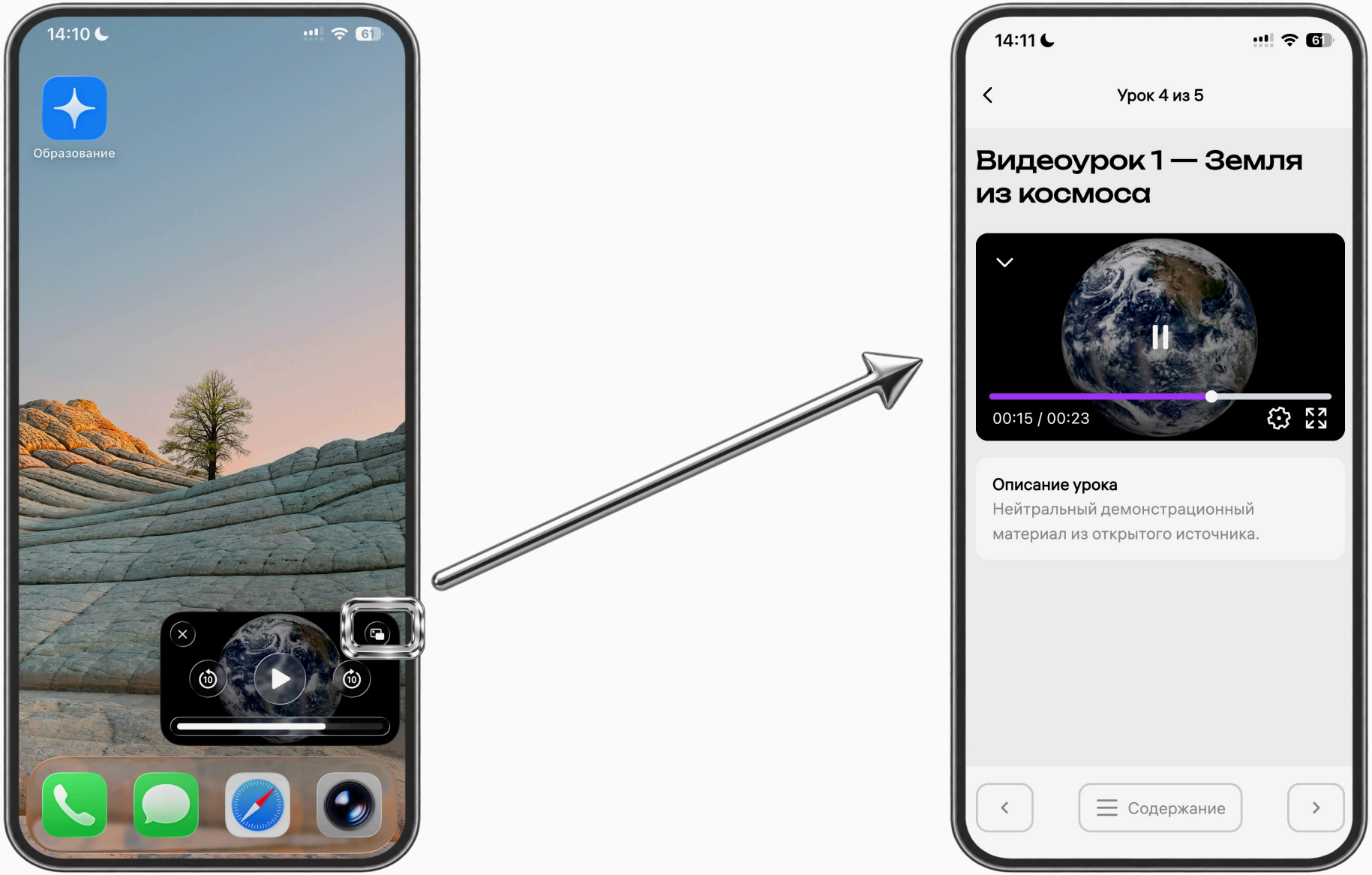
```
val delegate = object : NSObject(), AVPlayerViewControllerDelegateProtocol {
    override fun playerViewControllerWillStartPictureInPicture(
        playerViewController: AVPlayerViewController,
    ) {
        // Начинается вход в PiP
    }

    override fun playerViewControllerWillStopPictureInPicture(
        playerViewController: AVPlayerViewController,
    ) {
        // Начинается выход из PiP
    }
}

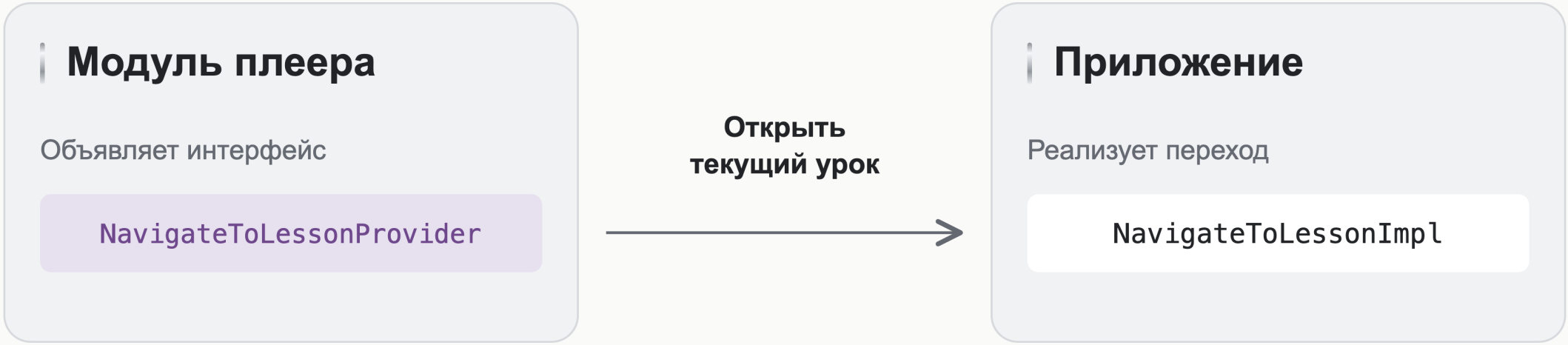
avPlayerViewController?.delegate = delegate
```

willStart / willStop — начало входа и выхода из PiP

Вернуться к видео, которое сейчас играет



Как связать плеер с навигацией приложения



ИНВЕРСИЯ ЗАВИСИМОСТЕЙ

Плеер зависит от контракта, а не от приложения

iOS: Как узнать, что нужно открыть урок

KOTLIN · iOS

```
private val delegate: AVPlayerViewControllerDelegateProtocol by lazy {
    object : NSObject(), AVPlayerViewControllerDelegateProtocol {
        ...

        override fun playerViewController(
            playerViewController: AVPlayerViewController,
            restoreUserInterfaceForPictureInPictureStopWithCompletionHandler: (Boolean) -> Unit
        ) {
            navigateToLessonProvider.navigate(
                ...,
                onCompleteNavigation = {
                    restoreUserInterfaceForPictureInPictureStopWithCompletionHandler(true)
                },
            )
        }
    }
}
```

Android: Как узнать, что нужно открыть урок

Режим PiP

```
onPictureInPictureModeChanged(
    false
)
```

Вышли из PiP.
Причина не передаётся.

Жизненный цикл Activity

`onResume()`	Снова активна
`onStop()`	Больше не видна

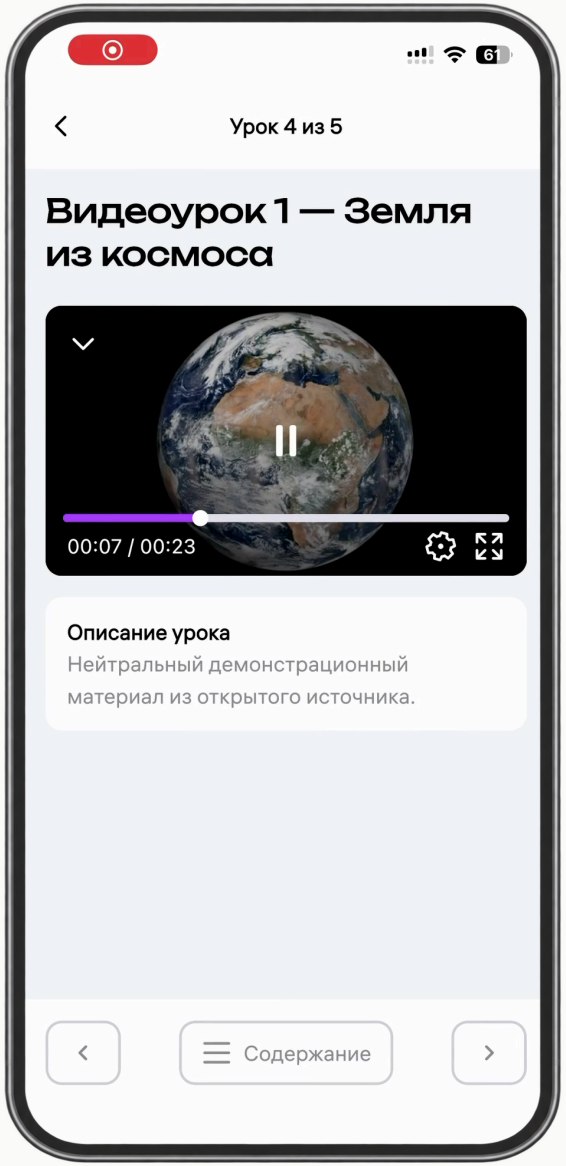
Порядок этих вызовов относительно друг друга не закреплён документацией

Закрытие можно обработать явно

Android 13+

`setCloseAction(RemoteAction)` → Наш `PendingIntent`

Полный экран: как управлять ориентацией



Запрос ориентации нужно согласовать с системой

Намерение регистрируем в общем коде

ОБЩИЙ UI — commonMain

Обёртка над интерфейсом плеера

```
@Composable
expect fun LandscapeOrientation(
  isLandscape: Boolean,
  content: @Composable () -> Unit
)
```

Android — androidMain

`actual fun LandscapeOrientation(…)`

Запрашивает ориентацию средствами Android

iOS — iosMain

`actual fun LandscapeOrientation(…)`

Запрашивает ориентацию средствами iOS

iOS: фиксируем ориентацию в хосте приложения



Основной контейнер приложения SWIFT

`OrientationViewController`

В полном экране разрешает только landscape

Compose-интерфейс

Плеер

При вертикальном расположении телефона оставляем горизонтальное видео

Android: фиксируем ориентацию

ПОЛНОЭКРАННЫЙ РЕЖИМ

KOTLIN · Android

Задаём режим ориентации

```
val activity = LocalContext.current as? Activity

...

LaunchedEffect(isLandscape) {
    activity?.requestedOrientation = if (isLandscape) {
        ...
        ActivityInfo.SCREEN_ORIENTATION_SENSOR_LANDSCAPE
    } else {
        ...
        ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED
    }
}
```



Один материал в разных режимах

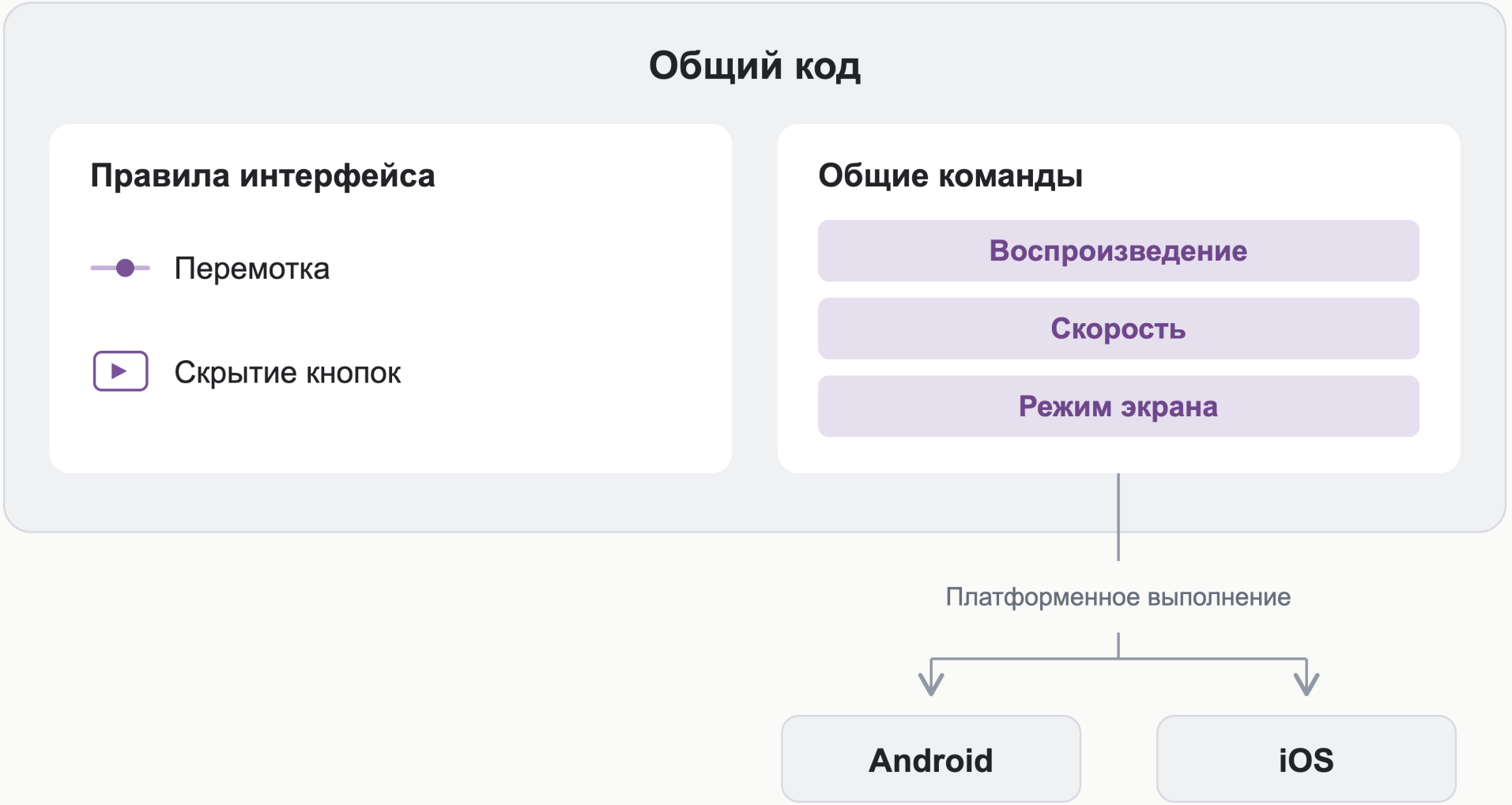
Переход между режимами сохраняет связь с текущим уроком



Общая логика плеера



Что объединили

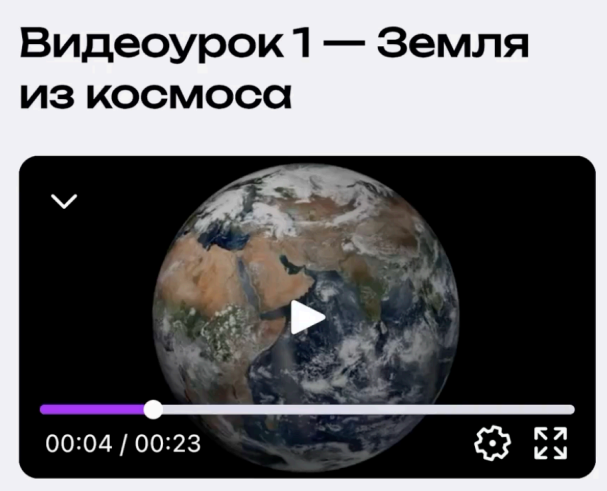




Что получили

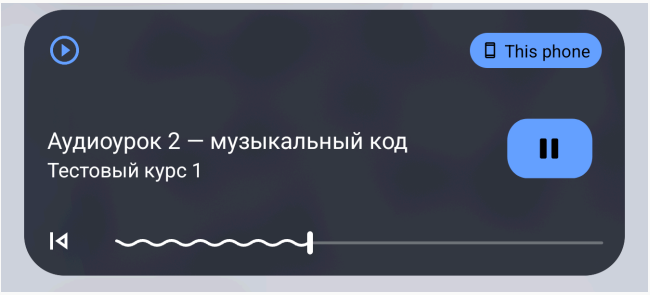


Поддержка аудио и видео

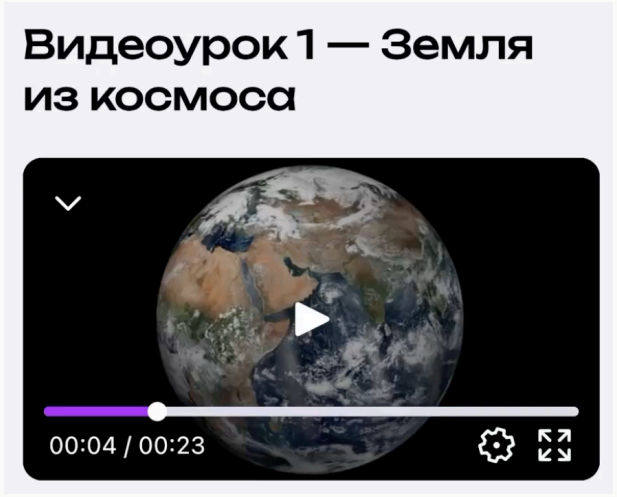


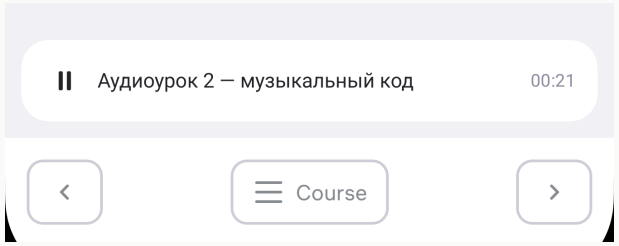


Аудио и видео доступны в приложении и в фоне



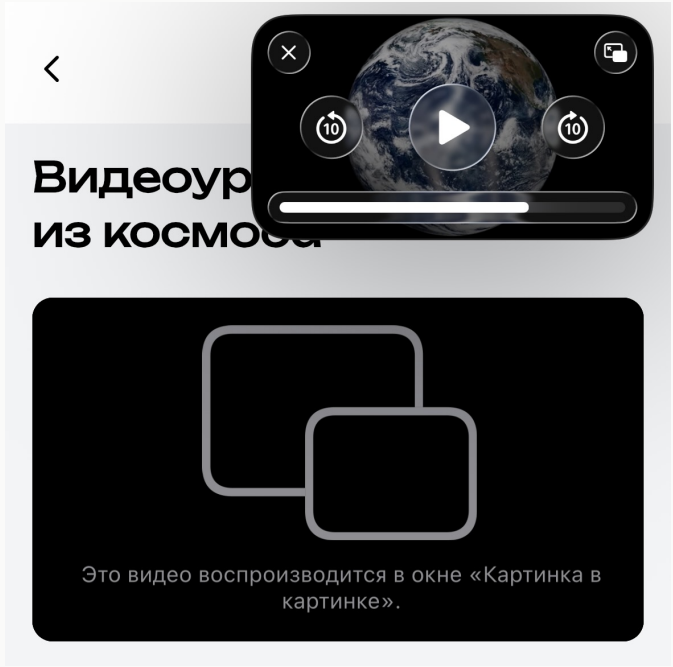
Один активный материал

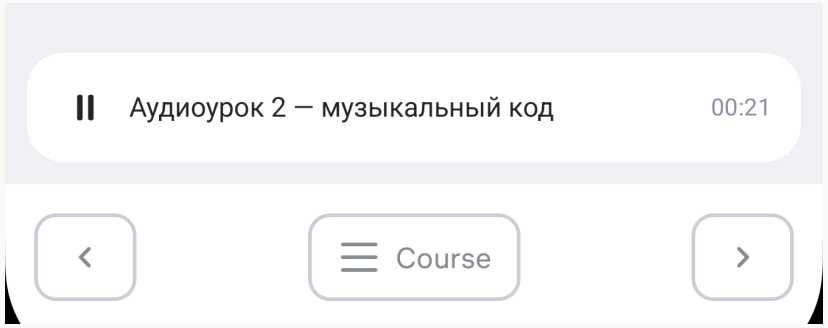




Единый медиаконтроллер хранит текущий материал

Несколько режимов отображения



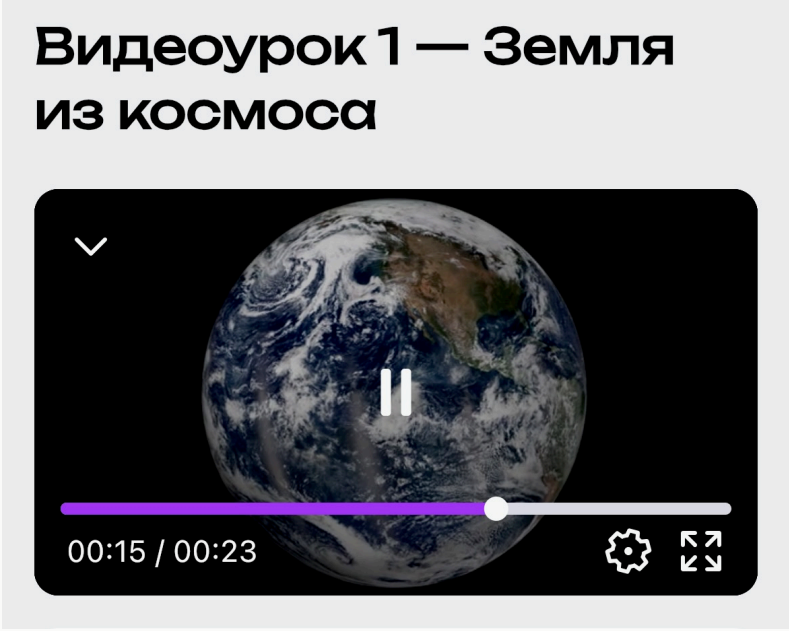


Жизненный цикл плеера отделён от интерфейса

Возврат к текущему уроку









Из PiP возвращаемся к воспроизводимому уроку

Что стоит учесть

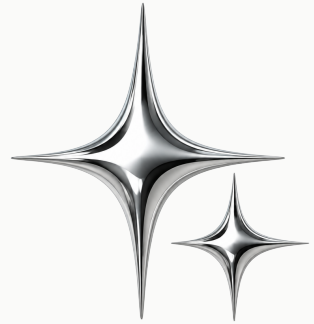
- Одинаковые действия могут иметь разные последствия
- Состояния платформ не совпадают один к одному
- Команды приходят из приложения и от системы
- Плеер и экран живут разное время

Общие команды ещё не делают поведение общим



Вопросы

Время для обсуждения



Спасибо



KMP Player: нативный опыт без дублирования логики

Мальцева Ангелина

независимый консультант