# Mobile Programming

## Advertisement & In-App Purchase

**Lecturer:** Ali Najimi
**Faculty of Computer Engineering**
**Sharif University of Technology**
**Fall 2027**

AdMob | Google Ads | In-App Purchase | Revenue | Adivery | Cafe Bazaar



# Table of Contents

1. Mobile Monetization Overview
2. Ad Types & Ad Networks
3. Google AdMob — Setup & Integration
4. Flutter: `google_mobile_ads` Package
5. Banner Ads in Flutter
6. Interstitial Ads in Flutter
7. Rewarded Ads in Flutter
8. Native Ads in Flutter
9. Android Native: Google Mobile Ads SDK
10. iOS Native: Google Mobile Ads SDK
11. Iran Market: Adivery + Cafe Bazaar
12. Adivery Flutter — Setup
13. Adivery Banner Ads

# Table of Contents

14. Adivery Interstitial & Rewarded Ads
15. Adivery Native Ads
16. In-App Purchase Overview
17. Flutter: `in_app_purchase` Package
18. Cafe Bazaar In-App Billing for Flutter
19. Cafe Bazaar Purchase Flow
20. Cafe Bazaar Security & Verification
21. Consumable & Non-Consumable Purchases
22. Android: Google Play Billing Library
23. iOS: StoreKit 2
24. Subscription Management
25. Ad & IAP Best Practices
26. Summary & Key Takeaways

# Mobile Monetization Overview

How do free apps make money? Here are the main strategies:

## Monetization Models:

| Model | How It Works | Revenue Type |
|---|---|---|
| **Paid Apps** | User pays once to download | One-time |
| **In-App Ads** | Show ads to users | CPM / CPC / CPA |
| **In-App Purchase** | Sell digital goods/features | Per transaction |
| **Subscriptions** | Recurring payment for content | Recurring |
| **Freemium** | Free base + paid premium | Mixed |

> **Industry Trend:** Most mobile apps are free. Ads + IAP are the dominant revenue model.

# Ad Types & Ad Networks

## Common Ad Formats:

- **Banner Ads:** Rectangular ads at top/bottom of screen. Low revenue per view.
- **Interstitial Ads:** Full-screen ads between screens. Higher revenue, more intrusive.
- **Rewarded Ads:** Users watch a video ad in exchange for a reward. Highest eCPM.
- **Native Ads:** Custom-styled ads that match your app's look and feel.

## Major Ad Networks:

| Network | Platform | Notes |
|---|---|---|
| **Google AdMob** | Android, iOS | Google Ads inventory |
| **Adivery** | Android, Flutter | Useful for Iranian apps |
| **Meta Audience Network** | Android, iOS | Facebook's ad network |
| **Unity Ads** | Android, iOS | Game-focused |
| **AppLovin / ironSource** | Android, iOS | Mediation + gaming monetization |

# Google AdMob — Setup & Integration

**AdMob** is Google's mobile ad platform. It's widely used in Flutter apps.

**Setup Steps:**

1. **Create an AdMob account** at admob.google.com.
2. **Register your app** and get **App IDs**.
3. **Create ad units** and get **Ad Unit IDs**.
4. **Configure platform-specific settings:**

**Android (`AndroidManifest.xml`)**

```
<manifest>
    <application>
        <meta-data
                android:name="com.google.android.gms.ads.APPLICATION_ID"
                android:value="ca-app-pub-xxxx~xxxxxxxxxx"/>
    </application>
</manifest>
```

**iOS (`Info.plist`)**

```
<key>GADApplicationIdentifier</key>
<string>ca-app-pub-xxxx~xxxxxxxxxx</string>
<key>NSUserTrackingUsageDescription</key>
<string>This identifier will be used to deliver personalized ads.</string>
```

# Flutter: `google_mobile_ads` Package

The official Google package for ads in Flutter.

```
# pubspec.yaml
dependencies:
  google_mobile_ads: ^5.3.0
```

# Flutter: `google_mobile_ads` Package

## Initialization:

```dart
import 'package:google_mobile_ads/google_mobile_ads.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  await MobileAds.instance.initialize();

  final config = RequestConfiguration(
    testDeviceIds: ['YOUR_TEST_DEVICE_ID'],
  );
  MobileAds.instance.updateRequestConfiguration(config);

  runApp(const MyApp());
}
```

> **Important:** Always test with **test ad unit IDs** during development.

# Banner Ads in Flutter

**Banner ads** are rectangular ads displayed at the top or bottom of the screen.

```
class BannerAdWidget extends StatefulWidget {
  @override
  State<BannerAdWidget> createState() => _BannerAdWidgetState();
}
```

# Banner Ads in Flutter (Widget)

```dart
class _BannerAdWidgetState extends State<BannerAdWidget> {
  BannerAd? _bannerAd;
  bool _isLoaded = false;
  @override
  void initState() {
    super.initState();
    _bannerAd = BannerAd(
      adUnitId: 'ca-app-pub-3940256099942544/6300978111',
      size: AdSize.banner,
      request: const AdRequest(),
      listener: BannerAdListener(
        onAdLoaded: (_) => setState(() => _isLoaded = true),
        onAdFailedToLoad: (ad, error) => ad.dispose(),
      ),
    ).load();
  }
  @override
  void dispose() { _bannerAd?.dispose(); super.dispose(); }
  @override
  Widget build(BuildContext context) {
    return _isLoaded && _bannerAd != null
      ? SizedBox(
          height: _bannerAd!.size.height.toDouble(),
          width: _bannerAd!.size.width.toDouble(),
          child: AdWidget(ad: _bannerAd!),
        )
      : const SizedBox.shrink();
  }
}
```

# Interstitial Ads in Flutter

**Interstitial ads** are full-screen ads shown at natural transition points.

```dart
class InterstitialAdHelper {
  InterstitialAd? _interstitialAd;
  int _numAttempts = 0;
  void createAd() {
    InterstitialAd.load(
      adUnitId: 'ca-app-pub-3940256099942544/1033173712',
      request: const AdRequest(),
      adLoadCallback: InterstitialAdLoadCallback(
        onAdLoaded: (ad) {
          _interstitialAd = ad;
          _numAttempts = 0;
        },
        onAdFailedToLoad: (error) {
          _numAttempts++;
          _interstitialAd = null;
          if (_numAttempts <= 2) createAd();
        },
      ),
    );
  }
```

# Interstitial Ads in Flutter (cont.)

```
  void showAd() {
    if (_interstitialAd == null) return;
    _interstitialAd!.fullScreenContentCallback =
      FullScreenContentCallback(
        onAdDismissedFullScreenContent: (ad) {
          ad.dispose();
          createAd();
        },
        onAdFailedToShowFullScreenContent: (ad, error) {
          ad.dispose();
          createAd();
        },
      );
    _interstitialAd!.show();
    _interstitialAd = null;
  }
}
```

# Rewarded Ads in Flutter

**Rewarded ads** let users earn in-app rewards by watching a video ad.

```
class RewardedAdHelper {
  RewardedAd? _rewardedAd;

  void loadAd() {
    RewardedAd.load(
      adUnitId: 'ca-app-pub-3940256099942544/5224354917',
      request: const AdRequest(),
      rewardedAdLoadCallback: RewardedAdLoadCallback(
        onAdLoaded: (ad) => _rewardedAd = ad,
        onAdFailedToLoad: (error) => _rewardedAd = null,
      ),
    );
  }
```

# Rewarded Ads in Flutter (cont.)

```
  void showAd({required Function(RewardItem) onReward}) {
    if (_rewardedAd == null) return;

    _rewardedAd!.fullScreenContentCallback = FullScreenContentCallback(
      onAdDismissedFullScreenContent: (ad) {
        ad.dispose();
        loadAd();
      },
    );

    _rewardedAd!.show(onUserEarnedReward: (ad, reward) {
      onReward(reward);
    });

    _rewardedAd = null;
  }
}
// Usage
adHelper.showAd(onReward: (reward) {
  player.addCoins(reward.amount.toInt());
});
```

# Native Ads in Flutter

**Native ads** match your app's visual design — you control the layout.

```
final NativeAd _nativeAd = NativeAd(
  adUnitId: 'ca-app-pub-3940256099942544/2247696110',
  request: const AdRequest(),
  factoryId: 'my_native_ad',
  listener: NativeAdListener(
    onAdLoaded: (_) => setState(() => _isLoaded = true),
    onAdFailedToLoad: (ad, error) => ad.dispose(),
  ),
).load();
// Display
if (_isLoaded)
  SizedBox(
    height: 300,
    width: 300,
    child: AdWidget(ad: _nativeAd),
  )
```

> **Note:** Native ads require platform-specific code to define the ad layout.

# Android Native: Google Mobile Ads SDK

For **native Android** (Kotlin), integrate Google Mobile Ads SDK directly.

**Setup (`build.gradle`):**

```
dependencies {
  implementation 'com.google.android.gms:play-services-ads:23.6.0'
}
```

# Android Native: Google Mobile Ads SDK

## Initialize and Load Banner (Kotlin):

```kotlin
class MainActivity : AppCompatActivity() {
  lateinit var adView: AdView

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    MobileAds.initialize(this) {
      adView = findViewById(R.id.adView)
      adView.adUnitId = "ca-app-pub-3940256099942544/6300978111"
      adView.setAdSize(AdSize.BANNER)
      adView.loadAd(AdRequest.Builder().build())
    }
  }
}
```

# Android Native: Google Mobile Ads SDK

## Interstitial (Kotlin):

```kotlin
InterstitialAd.load(this,
  "ca-app-pub-3940256099942544/1033173712",
  AdRequest.Builder().build(),
  object : InterstitialAdLoadCallback() {
    override fun onAdLoaded(ad: InterstitialAd) { mAd = ad }
    override fun onAdFailedToLoad(e: LoadAdError) { mAd = null }
  })
```

# iOS Native: Google Mobile Ads SDK

For **native iOS** (Swift), use Google Mobile Ads SDK via CocoaPods or SPM.

**Setup (`Podfile`):**

```
pod 'Google-Mobile-Ads-SDK'
```

# iOS Native: Google Mobile Ads SDK

## Initialize and Load Banner (Swift):

```swift
import GoogleMobileAds

class ViewController: UIViewController {
  var bannerView: GADBannerView!

  override func viewDidLoad() {
    super.viewDidLoad()
    MobileAds.shared.start(completionHandler: nil)

    bannerView = GADBannerView(adSize: GADAdSizeBanner)
    bannerView.adUnitID = "ca-app-pub-3940256099942544/2934735716"
    bannerView.rootViewController = self
    bannerView.load(GADRequest())
    view.addSubview(bannerView)
  }
}
```

# iOS Native: Google Mobile Ads SDK

## Interstitial (Swift):

```swift
GADInterstitialAd.load(
  withAdUnitID: "ca-app-pub-3940256099942544/4411468910",
  request: GADRequest()
) { ad, error in
  if let error = error { print(error); return }
  self.interstitial = ad
  self.interstitial?.present(fromRootViewController: self)
}
```

# Iran Market: Adivery + Cafe Bazaar

For apps targeting Iranian Android users, two local services are commonly relevant:

## Adivery

- Local ad network / ad SDK
- Flutter package: `adivery`
- Supports Android Flutter apps
- Common formats: banner, interstitial, rewarded, native
- Uses **App ID** + **Placement IDs**

## Cafe Bazaar IAB

- Local Android app store billing
- Flutter options include `flutter_poolakey` and `cafebazaar_flutter`
- Uses Bazaar product IDs and purchase tokens
- Important for apps distributed through Cafe Bazaar

# Iran Market: Adivery + Cafe Bazaar

> Use AdMob / Google Play for global distribution, and Adivery / Cafe Bazaar when your main distribution channel is
> Iran-focused Android.

# Adivery Flutter — Setup

Adivery provides a Flutter plugin for activating ads inside Flutter apps.

```
# pubspec.yaml
dependencies:
  adivery: ^4.8.8
```

# Adivery Flutter — Setup

## Import and initialize:

```dart
import 'package:adivery/adivery.dart';
import 'package:adivery/adivery_ads.dart';

void initAds() {
  AdiveryPlugin.initialize("YOUR_ADIVERY_APP_ID");
  AdiveryPlugin.setLoggingEnabled(true); // development only
}
```

## Core identifiers:

| Identifier | Meaning |
|---|---|
| **App ID** | Your Adivery application identifier |
| **Placement ID** | Identifier for each ad placement |
| **Banner Placement** | Used inside `BannerAd` widget |
| **Interstitial / Rewarded Placement** | Prepared, loaded, checked, then shown |

# Adivery Banner Ads

Adivery banners are added as Flutter widgets.

```
BannerAd(
  "YOUR_BANNER_PLACEMENT_ID",
  BannerAdSize.BANNER,
  onAdLoaded: (ad) {
    print("banner loaded");
  },
  onAdClicked: (ad) {
    print("banner clicked");
  },
  onError: (ad, error) {
    print("banner failed: $error");
  },
)
```

# Adivery Banner Ads

### Common banner sizes:

| Size | Use Case |
| --- | --- |
| `BannerAdSize.BANNER` | Standard bottom/top banner |
| `BannerAdSize.LARGE_BANNER` | Larger banner area |
| `BannerAdSize.MEDIUM_RECTANGLE` | Feed/card placement |

**Best practice:** Put banners where they do not cover navigation, forms, or game controls.

# Adivery Interstitial & Rewarded Ads

Interstitial and rewarded ads should be **prepared before showing**.

```
void prepareFullScreenAds() {
  AdiveryPlugin.prepareInterstitialAd("INTERSTITIAL_PLACEMENT_ID");
  AdiveryPlugin.prepareRewardedAd("REWARDED_PLACEMENT_ID");

  AdiveryPlugin.addListener(
    onInterstitialLoaded: (placement) {
      print("interstitial loaded");
    },
    onRewardedClosed: (placement, isRewarded) {
      if (isRewarded) {
        // deliver reward here
      }
    },
    onError: (placement, error) {
      print("ad error: $error");
    },
  );
}
```

# Adivery Interstitial & Rewarded Ads

## Show only when loaded:

```
Future<void> showPlacement(String placementId) async {
  final isLoaded = await AdiveryPlugin.isLoaded(placementId);
  if (isLoaded) {
    AdiveryPlugin.show(placementId);
  }
}
```

# Adivery Native Ads

Native ads let you build the UI yourself using ad assets.

```
NativeAd? nativeAd;

void loadNative() {
  nativeAd = NativeAd(
    "YOUR_NATIVE_PLACEMENT_ID",
    onAdLoaded: () {
      setState(() {});
    },
  );

  nativeAd!.loadAd();
}
```

# Adivery Native Ads

## Example rendering idea:

```
if (nativeAd != null && nativeAd!.isLoaded) {
  return Column(
    children: [
      nativeAd!.image,
      Row(
        children: [
          nativeAd!.icon,
          Text(nativeAd!.headline),
        ],
      ),
      Text(nativeAd!.description),
      ElevatedButton(
        onPressed: nativeAd!.recordClick,
        child: Text(nativeAd!.callToAction),
      ),
    ],
  );
}
```

# Adivery Native Ads

> Always respect the ad network policy: keep the ad recognizable, do not fake clicks, and call click/impression methods
> correctly.

# In-App Purchase Overview

**In-App Purchase (IAP)** lets users buy digital goods and services inside your app.

## Product Types:

| Type | Description | Example |
|---|---|---|
| **Consumable** | Used once, can be repurchased | Coins, extra lives |
| **Non-Consumable** | Bought once, permanent | Remove ads, premium unlock |
| **Auto-Renewable Sub** | Recurring billing | Monthly premium, cloud storage |
| **Non-Renewing Sub** | Fixed period access | 1-year membership |

## Platform Stores:

- **Android:** Google Play Billing, Cafe Bazaar Billing, Samsung Galaxy Store, Amazon Appstore
- **iOS:** Apple App Store / StoreKit

# In-App Purchase Overview

> **Rule:** For digital goods, use the billing system required by the store that distributes your app.

# Flutter: `in_app_purchase` Package

The official Flutter plugin for handling in-app purchases on Google Play and Apple App Store.

```
# pubspec.yaml
dependencies:
  in_app_purchase: ^3.2.1
  in_app_purchase_storekit: ^0.3.7
  in_app_purchase_android: ^0.3.6
```

# Flutter: `in_app_purchase` Package

## Initialization and Purchase Stream:

```dart
import 'package:in_app_purchase/in_app_purchase.dart';
import 'dart:async';
class IAPManager {
  final InAppPurchase _iap = InAppPurchase.instance;
  late StreamSubscription<List<PurchaseDetails>> _subscription;
  List<ProductDetails> _products = [];
  Future<void> init() async {
    final available = await _iap.isAvailable();
    if (!available) return;
    _subscription = _iap.purchaseStream.listen(_onPurchaseUpdate);

    final response = await _iap.queryProductDetails(
      {'coins_100', 'premium_upgrade', 'monthly_sub'},
    );
    _products = response.productDetails.toList();
  }
  void _onPurchaseUpdate(List<PurchaseDetails> purchases) {
    for (final purchase in purchases) {
      if (purchase.status == PurchaseStatus.purchased) {
        _verifyAndDeliver(purchase);
      }
      if (purchase.pendingCompletePurchase) {
        _iap.completePurchase(purchase);
      }
    }
  }
```

# Cafe Bazaar In-App Billing for Flutter

Cafe Bazaar billing is Android-only and is used for apps distributed through Cafe Bazaar.

### Option A: `flutter_poolakey`

```
dependencies:
  flutter_poolakey: ^2.2.0+1.0.0
```

```
import 'package:flutter_poolakey/flutter_poolakey.dart';
```

Also add JitPack in Gradle when required:

```
allprojects {
  repositories {
    maven { url 'https://jitpack.io' }
  }
}
```

### Option B: `cafebazaar_flutter`

```
dependencies:
  cafebazaar_flutter: ^2.0.0
```

```
import 'package:cafebazaar_flutter/cafebazaar_flutter.dart';

final bazaar = CafebazaarFlutter.instance;
final inapp = bazaar.inAppPurchase(PUBLIC_KEY);
```

`PUBLIC_KEY` is taken from the Cafe Bazaar developer panel.

# Cafe Bazaar Purchase Flow

A typical Cafe Bazaar purchase flow:

1. Create products in the Cafe Bazaar developer panel.
2. Get your public/RSA key from the panel.
3. Initialize the billing plugin.
4. Start a purchase or subscription flow.
5. Receive purchase data.
6. Verify signature, payload, product ID, and order ID.
7. Deliver content.
8. Consume the purchase if it is consumable.
9. Disconnect / dispose billing connection when done.

# Cafe Bazaar Purchase Flow

```
final PurchaseInfo purchaseInfo = await inapp.purchase(
  "productId",
  payLoad: "unique_payload_per_request",
);

if (purchaseInfo != null) {
  // send purchaseInfo to backend for verification
  // then unlock item or add coins
}
```

## Subscription:

```
final PurchaseInfo purchaseInfo = await inapp.subscribe(
  "subscriptionProductId",
  payLoad: "unique_payload_per_request",
);
```

# Cafe Bazaar Security & Verification

Never deliver paid content only because the client says “success”.

**Important fields to validate:**

| Field | Why It Matters |
|---|---|
| `productId` | Confirms the purchased item |
| `purchaseToken` | Used for tracking and consuming purchases |
| `orderId` | Must be unique; prevents replay attacks |
| `payload` / `developerPayload` | Should match the request token you generated |
| `dataSignature` | Used to verify authenticity |
| `packageName` | Confirms the purchase belongs to your app |

**Consumable purchase:**

```
final bool consumeResult = await inapp.consume(
  purchaseInfo.purchaseToken,
);
```

# Consumable and Non-Consumable Purchases

### Consumable (coins, lives):

```
final param = PurchaseParam(productDetails: product);
_iap.buyConsumable(purchaseParam: param);

// On purchase success:
// 1. Verify server-side
// 2. Add coins
// 3. Mark token consumed
```

### Non-Consumable (premium):

```
final param = PurchaseParam(productDetails: product);
_iap.buyNonConsumable(purchaseParam: param);

// On purchase success:
// 1. Verify server-side
// 2. Unlock permanently
// 3. Store entitlement
```

### Purchase Verification Flow:

1. User taps "Buy"
2. Store processes payment
3. App receives receipt / token
4. **Send token to server**
5. Server validates with store
6. Server records entitlement
7. App delivers content
8. App completes / acknowledges purchase

# Consumable and Non-Consumable Purchases

**Key methods:**

- `buyConsumable()`
- `buyNonConsumable()`
- `completePurchase()`
- `queryProductDetails()`

# Android: Google Play Billing Library

For **native Android** (Kotlin), use Google Play Billing Library directly.

**Setup (`build.gradle`):**

```
dependencies {
  implementation 'com.android.billingclient:billing-ktx:7.1.1'
}
```

# Android: Launching a Purchase

## Launching a purchase flow (Kotlin):

```
val billingFlowParams = BillingFlowParams.newBuilder()
  .setProductDetailsParamsList(
    listOf(
      BillingFlowParams.ProductDetailsParams.newBuilder()
        .setProductDetails(productDetails)
        .build()
    )
  )
  .build()

val result = billingClient.launchBillingFlow(activity, billingFlowParams)
```

# Android: Launching a Purchase

### Acknowledging a purchase:

```
val acknowledgeParams = AcknowledgePurchaseParams.newBuilder()
  .setPurchaseToken(purchase.purchaseToken)
  .build()

billingClient.acknowledgePurchase(acknowledgeParams) { result ->
  if (result.responseCode == BillingClient.BillingResponseCode.OK) {
    deliverPremium()
  }
}
```

# Android: Launching a Purchase

## Consuming a consumable:

```
val consumeParams = ConsumeParams.newBuilder()
  .setPurchaseToken(purchase.purchaseToken)
  .build()

billingClient.consumeAsync(consumeParams) { _, _ ->
  addCoins(100)
}
```

# iOS: StoreKit 2

For **native iOS** (Swift), Apple provides **StoreKit 2** with a modern async API.

**Query Products (Swift):**

```swift
import StoreKit

let products: [Product] = try await Product.products(
  for: ["coins_100", "premium_upgrade", "monthly_sub"]
)

for product in products {
  print("\(product.displayName): \(product.displayPrice)")
}
```

# iOS: StoreKit 2

## Purchase a Product (Swift):

```swift
func purchase(_ product: Product) async throws -> Transaction? {
  let result = try await product.purchase()

  switch result {
  case .success(let verification):
    let transaction = try checkVerified(verification)
    deliverContent(transaction)
    await transaction.finish()
    return transaction

  case .userCancelled:
    return nil

  case .pending:
    return nil

  @unknown default:
    return nil
  }
}
```

# iOS: Transaction Verification

## StoreKit 2 verification (Swift):

```swift
func checkVerified<T>(_ result: VerificationResult<T>) throws -> T {
  switch result {
  case .unverified(_, let error):
    throw error
  case .verified(let safe):
    return safe
  }
}
func listenForTransactions() async {
  for await result in Transaction.updates {
    do {
      let transaction = try checkVerified(result)
      deliverContent(transaction)
      await transaction.finish()
    } catch {
      print("Transaction failed verification")
    }
  }
}
func restorePurchases() async {
  try await AppStore.sync()
}
```

# iOS: Transaction Verification

> StoreKit 2 verifies transactions on-device using Apple's certificate chain. Production systems can also verify server-side.

# Subscription Management

**Subscriptions** require special handling for billing periods, grace periods, and upgrades.

**Flutter Subscription Purchase:**

```
final param = PurchaseParam(productDetails: subscriptionProduct);
_iap.buyNonConsumable(purchaseParam: param);
```

# Subscription Management

## Key Subscription Concepts:

| Concept | Description |
|---|---|
| **Billing Period** | Weekly, monthly, yearly |
| **Free Trial** | N days free before first charge |
| **Introductory Price** | Discounted first period |
| **Grace Period** | Extra days if payment fails |
| **Upgrade/Downgrade** | Switch between subscription tiers |
| **Proration Mode** | How billing changes on upgrade |

# Subscription Management

## Checking Subscription Status:

```
if (purchase.productID.contains('sub_')) {
  final isActive = await server.verifySubscription(
    purchase.verificationData.serverVerificationData,
  );
  if (isActive) unlockPremium();
}
```

# Ad and IAP Best Practices

**Ad Best Practices:**

- Don't show ads too frequently.
- Use **rewarded ads** for opt-in engagement.
- Preload interstitial ads before showing.
- Use test ad IDs during development.
- Use mediation where appropriate.
- Respect privacy and platform rules.
- For Iran-focused apps, evaluate local fill rate with Adivery.

**IAP Best Practices:**

- Always **verify purchases** on a backend server.
- Call `completePurchase()` / `finish()` after delivery.
- Handle pending purchases.
- Implement restore purchases.
- Test with sandbox accounts.
- Log purchase lifecycle events.
- For Cafe Bazaar, verify payload, order ID, token, signature, and package name.

# Ad and IAP Best Practices

**Revenue strategy:** Ads work best with high traffic; IAP/subscriptions work best when users clearly understand the premium value.

# Recommended Monetization Architecture

## Client App

- Shows ads
- Starts purchase flow
- Sends purchase token to backend
- Shows entitlement state
- Never decides payment validity alone

## Backend Server

- Verifies purchase / receipt / token
- Checks duplicate order IDs
- Stores entitlements
- Tracks subscriptions
- Delivers rewards safely

```
User → App → Store SDK → Purchase Token → Backend Verification → Entitlement
```

> A safe monetization system treats the app as a UI layer and the backend as the source of truth.

# Summary: Key Takeaways

## Ads

- `google_mobile_ads` for global Flutter ads.
- Adivery for Iran-focused Android Flutter.
- Banner, Interstitial, Rewarded, Native.
- Rewarded ads are best when real value.
- Never overload the user with ads.

## IAP

- `in_app_purchase` for Google Play / App Store.
- Cafe Bazaar billing for Bazaar apps.
- Consumable, Non-Consumable, Subscription types.
- Always verify purchases server-side.

## Strategy

- Ads for free apps with high traffic.
- IAP for premium features and digital goods.
- Subscriptions for recurring value.
- Combine ads + IAP carefully.

## Compliance

- Use the correct billing system for store.
- Test with sandbox/test accounts.
- Implement restore purchases.
- Provide clear pricing and subscription management.
- Keep logs for support and fraud detect.

# References

- Google AdMob: https://admob.google.com
- Flutter `google_mobile_ads`: https://pub.dev/packages/google_mobile_ads
- Flutter `in_app_purchase`: https://pub.dev/packages/in_app_purchase
- Cafe Bazaar Flutter In-App Billing: https://developers.cafebazaar.ir/fa/guidelines/in-app-billing/implementation/flutter#
- `flutter_poolakey`: https://pub.dev/packages/flutter_poolakey
- `cafebazaar_flutter`: https://pub.dev/packages/cafebazaar_flutter
- Adivery Flutter Documentation: https://docs.adivery.com/flutter
- Adivery package: https://pub.dev/packages/adivery

# Thank You! 🙌

## Questions?

> *"My app makes $0.01 per day from ads.*
> *At this rate, I'll be a millionaire in*
> *273,972 years."*
> *— Every indie mobile developer*