📱 Tim's Realm - Payment & Refund Research

Apple IAP, Stripe, and Alternative Mechanisms

Saturday, January 24, 2026 at 03:11 AM

📄 Contents

1. Alternative Mechanisms

Alternative Payment & Refund Mechanisms for Apps

Research compiled: January 24, 2026

This document explores alternatives to traditional refunds, product patterns that reduce refund issues, and recommendations for structuring an app to minimize refund requests while maximizing customer satisfaction.

---

1. Alternative Mechanisms to Refunds

Credits/Wallet Systems

Instead of cash refunds, many apps offer store credits or wallet balance:

| Approach | How It Works | Best For | |----------|--------------|----------| | **App Store Credits** | Refund issued as credits for future purchases within the app | Games, content libraries | | **Internal Wallet** | User has a balance they can apply to purchases | Marketplace apps, multi-product apps | | **Bonus Credits** | Give 110-120% of value as credits instead of 100% cash refund | Encouraging retention over refunds |

Example Implementation:

- Offer "satisfaction credits" worth 125% of purchase value
- Credits expire after 12 months (creates urgency)
- Credits can be used on any product/tier upgrade

Pros:

- Keeps money in the ecosystem
- Users may spend more than original refund value

- Reduces administrative overhead

Cons:

- Some users will still demand cash
- May not satisfy consumer protection laws in all jurisdictions
- Requires clear terms of service

Subscription Pausing vs Cancellation

Apple and Google Play both support subscription pausing, a powerful alternative to cancellation:

How It Works:

- User can pause their subscription for 1-12 months
- No billing during pause period
- Subscription automatically resumes
- Days of pause don't count toward subscriber tenure

Implementation Options:

| Method | User Experience | Business Impact | |--------|----------------|----------------| | **System-level pause** | User pauses via App Store/Play Store settings | Lower churn, automatic resumption | | **In-app pause** | Custom pause flow with retention offers | Better UX control, can offer alternatives | | **Seasonal pause** | Automatic pause during off-seasons (e.g., summer for study apps) | Proactive churn prevention |

Best Practices: 1. Surface pause option before cancellation confirmation 2. Offer pause as alternative when user tries to cancel 3. Send re-engagement emails before pause ends 4. Consider "skip a month" for monthly subscriptions

---

Pro-Rated Refunds

Instead of full refunds, calculate based on actual usage:

Models:

1. **Time-based proration:**

- Annual subscription: Refund = (Days remaining / 365) × Price
- Monthly subscription: Refund = (Days remaining / 30) × Price

2. **Usage-based proration:**

- Content consumed / Total content available
- Features used / Features available
- API calls made / API calls included

3. **Apple's Automatic Behavior:**

- Upgrades: Apple automatically provides pro-rated refunds
- Downgrades: Take effect at next renewal (no refund)
- Cancellations: No automatic refund (user must request)

Recommendation: Offer pro-rated refunds proactively for annual plans. It feels fair and reduces full refund requests.

Trial Periods and Money-Back Guarantees

Free Trial Types:

Type	Duration	Best For
Standard free trial	3-7 days	Simple apps, quick value demonstration
Extended trial	14-30 days	Complex apps, habit-forming products
Paid trial	7 days at $0.99	Reduces trial abuse, higher conversion
Feature-limited trial	Unlimited time	Freemium conversion

Money-Back Guarantees:

Structure as explicit promises rather than relying on platform refund policies:

```
"30-Day Satisfaction Guarantee: If you're not completely satisfied
within 30 days, we'll refund your purchase — no questions asked."
```

Benefits:

- Reduces purchase anxiety → Higher conversion
- Sets clear expectations
- You control the refund flow (via web/support)
- Creates positive brand perception

Implementation Tips: 1. Make guarantee prominent on purchase screen 2. Track refund rate by acquisition source 3. Use refund requests as customer research opportunities 4. Consider offering alternatives before processing refund

---

"Satisfaction Guarantee" Models

Go beyond refunds with satisfaction-focused approaches:

Model Options:

1. **Results Guarantee:**

- "Complete 30 meditations or your money back" (Calm-style)
- Creates engagement requirement
- Most who engage won't want refund

2. **Support Guarantee:**

- "If our support can't solve your issue, full refund"
- Encourages contact before refund request
- Opportunity to save the customer

3. **Upgrade Guarantee:**

- "Not happy with Basic? We'll credit you toward Premium"
- Converts refund requests into upsells

- Customer feels like they're getting more value

4. **No-Questions-Asked:**

- Simplest to administer
- Builds trust
- May increase refund rate but also conversion rate

---

2. Product Patterns That Avoid Refund Issues

Freemium with In-App Purchases

Why It Reduces Refunds:

- Users validate app value before paying
- No expectation of "all-or-nothing" purchase
- Lower individual purchase amounts

Successful Patterns:

| Pattern | Example | Why It Works | |---------|---------|--------------| | **Feature unlock** | Pro camera modes | Users know exactly what they're buying | | **Content access** | Premium meditation packs | Can sample before purchasing | | **Capacity increase** | More storage, more projects | Pay for what you need | | **Ad removal** | One-time purchase | Clear value proposition |

Implementation Tips:

- Give enough free value that users feel the app works
- Make premium value crystal clear before purchase
- Consider "lite" versions of premium features in free tier

Consumable vs Non-Consumable Purchases

Consumables (gems, credits, tokens):

- Naturally avoid refund issues

- User "uses up" the purchase
- Lower individual transaction value
- Can be generous with bonuses

Non-Consumables (unlock forever):

- Higher refund risk
- But clearer value proposition
- Work well for utilities and tools

Hybrid Approach:

- Core features as non-consumable
- Premium content as consumable/subscription
- Bonus features with expiring credits

Subscription Tiers with Easy Downgrade

Tier Structure Example:

| Tier | Price | Use Case | |------|-------|----------| | Free | $0 | Core features, ads | | Basic | $4.99/mo | Remove ads, basic premium | | Pro | $9.99/mo | Full features | | Teams | $29.99/mo | Collaboration, admin |

Why Multiple Tiers Reduce Refunds: 1. **Exit ramp**: Users can downgrade instead of cancel 2. **Right-sizing**: Users find the tier that fits 3. **Retention path**: Can offer downgrade before refund 4. **Perceived value**: Lower tier feels like "getting something"

Apple StoreKit Behavior:

- Upgrades: Immediate, pro-rated refund applied
- Downgrades: Take effect at next renewal
- Crossgrades (same level): Immediate if same duration

Usage-Based Billing

How It Works:

- Charge based on actual consumption
- Common in API services, AI tools, cloud storage
- Examples: OpenAI API, Twilio, AWS

Benefits for Refunds:

- Users only pay for what they use
- No "unused subscription" complaints
- Natural alignment between cost and value

Implementation Models:

1. **Pure pay-as-you-go:**

- Charge per unit (API call, minute, etc.)
- Prepaid credits or postpaid billing

2. **Tiered usage:**

- Base subscription + overage charges
- Included usage per tier

3. **Hybrid credits:**

- Monthly credits included in subscription
- Additional credits purchasable
- Unused credits roll over (limited)

Considerations:

- More complex to implement than flat subscriptions
- Requires robust usage tracking
- May cause anxiety for budget-conscious users
- Consider spending alerts and caps

Token/Credit Systems

How Credits Work:

```
Purchase: $10 → 1,000 credits
Use: Generate image = 50 credits
     Export PDF = 20 credits
     Premium template = 100 credits
```

Why Credits Reduce Refunds: 1. **Psychological**: Spending credits feels different than money 2. **Flexibility**: Users choose how to spend value 3. **Partial use**: Users don't feel "all or nothing" 4. **Rollover**: Unused credits carry forward (not lost) 5. **Bonus incentives**: Give bonus credits for larger purchases

Credit System Design Tips:

- Use round numbers (100, 500, 1000 credits)
- Price so $1 ≈ 100 credits (easy mental math)
- Offer subscription + credits bundles
- Provide clear credit balance visibility
- Send alerts when credits are low

---

3. How Successful Apps Handle Refunds

Calm

Subscription Model: Primarily annual at $69.99/year (often discounted to $39.99-49.99)

Refund Approach:

- 7-day free trial for new users
- No explicit money-back guarantee
- Refunds handled through Apple/Google (platform-level)
- Support focuses on resolving issues before refunds
- Offers subscription pausing as alternative

Retention Strategies:

- Daily reminder notifications
- Streak/habit tracking
- New content releases (Daily Calm, celebrity narrations)
- Seasonal content updates
- Family plan option

Lessons for Tim:

- Strong onboarding reduces refund requests
- Daily engagement features reduce buyer's remorse
- Content freshness justifies ongoing subscription

Headspace

Subscription Model: Monthly ($12.99) and Annual ($69.99)

Refund Approach:

- 7-14 day free trial
- Historically offered 30-day money-back guarantee for annual
- Platform-level refunds for recent purchases
- Enterprise/B2B customers: custom refund policies

Unique Strategies:

- Graduated pricing (students, healthcare workers)
- Employer/insurance partnerships (reduces individual refunds)
- Free "basics" content always available
- In-app cancel flow surfaces alternatives

Lessons for Tim:

- B2B/enterprise distribution removes consumer refund issues
- Discount programs for price-sensitive segments

- Keep some value accessible even after cancellation

Duolingo

Model: Freemium with Super Duolingo subscription

Refund Approach:

- Generous free tier reduces paid subscriber complaints
- Super subscription is enhancement, not requirement
- Platform-level refunds only
- Family plan available

Why Duolingo Has Few Refund Issues: 1. Core product is free 2. Premium is additive (no hearts, offline, etc.) 3. Strong habit formation (streaks) 4. Clear value proposition 5. Can use free version indefinitely

Lessons for Tim:

- A strong free tier is the best refund prevention
- Subscription should enhance, not gate core value
- Gamification creates emotional investment

Other Notable Approaches

Spotify:

- 1-3 month free trials (with limitations)
- Premium for podcast features
- Student/family discounts
- Ad-supported free tier

Netflix (App Store model):

- No in-app purchases on iOS (reader app exception)
- Users subscribe via web → No App Store refunds

- Full control over billing and refunds
- 30-day money-back for new subscribers

Notion:

- Generous free tier (individual use)
- Paid for teams/enterprise
- Education plans free
- Refunds handled directly (web payment)

---

4. Web vs Native Payment Options

Reader Apps Exception (Netflix/Spotify Model)

What It Is: Apps that provide access to previously purchased or subscribed content can avoid Apple's in-app purchase requirement.

Categories:

- Magazines, newspapers, books (reader apps)
- Audio/music streaming (Spotify)
- Video streaming (Netflix)
- Content subscriptions (Kindle)

How It Works:

- User purchases subscription on web
- App provides access to purchased content
- No in-app purchase buttons
- No links to purchase page (historically)
- 0% Apple commission

Limitations:

- Cannot sell new content in-app
- Cannot link to web purchase (until recent changes)

- Must be genuinely "reader" content

External Link Entitlement

Recent Changes (EU DMA & US Court Rulings):

Apple has been forced to allow: 1. **External links** to purchase pages (with conditions) 2. **Alternative payment processors** in some regions 3. **Communication about external purchases**

Current State (2026):

| Region | External Links | Alt Payment | Apple Commission | |--------|---------------|-------------|------------------| | EU (DMA) | Yes | Yes | Reduced/negotiable | | US | Limited | Limited | Standard (15-30%) | | Japan | Yes (certain apps) | No | Standard | | South Korea | Yes | Yes | Standard minus | | Rest of World | No | No | Standard |

External Link Implementation:

- Add "Manage Subscription" link to website
- Clearly communicate subscription management options
- Handle customer service directly
- Still subject to Apple's guidelines about presentation

EU DMA Implications

Digital Markets Act Requirements (Apple):

1. **Alternative App Stores:** Users can install apps from outside App Store 2. **Alternative Payment:** Apps can use third-party payment 3. **Developer Communication:** Can contact users about offers 4. **Reduced Fees:** Core Technology Fee instead of full commission (complex)

For Tim's App:

- If targeting EU, can offer web checkout

- Consider hybrid: IAP for convenience, web for savings
- Must still comply with EU consumer protection laws
- 14-day cooling-off period for digital content

Risks:

- Complex compliance requirements
- Core Technology Fee may exceed commission for small developers
- Different experience for EU vs non-EU users

Alternative Payment Platforms

Paddle

What It Is: Merchant of Record for software/SaaS

How It Works:

- Paddle is the legal seller (not you)
- Handles tax, compliance, fraud
- You receive net revenue
- Single dashboard for all payments

Fees: ~5% + $0.50 per transaction (varies)

For Mobile Apps:

- Best for web-based subscription sales
- Can be used with "web purchase" flow
- Handles refunds, chargebacks, tax

Best For:

- SaaS with web and mobile apps
- Developers who want hands-off billing
- International sales with tax complexity

Lemon Squeezy

What It Is: All-in-one platform for selling digital products

Features:

- Subscription billing
- License key management
- Digital downloads
- Tax handling (MoR)
- Affiliate program built-in

Fees: 5% + $0.50 per transaction

Unique Value:

- Very developer-friendly
- Beautiful checkout flows
- Easy setup (minutes, not days)
- Good for indie developers

For Tim:

- Consider for web-based purchases
- License key feature useful for desktop apps
- Handles recurring billing well

Gumroad

What It Is: Simple platform for selling digital products

Fees: 10% flat fee (no monthly)

Best For:

- Simple products (ebooks, courses)

- One-time purchases
- Creators over developers
- Quick launch

Limitations:

- Less sophisticated subscription management
- Higher fees than alternatives
- Less developer tooling

Stripe Direct

What It Is: Payment infrastructure (you're the merchant)

Fees: 2.9% + $0.30 per transaction

Considerations:

- You handle tax compliance
- You handle refunds/chargebacks
- You handle subscription logic
- Most flexible, most work

Best For:

- Companies with resources for billing infrastructure
- Complex custom billing needs
- Maximum control

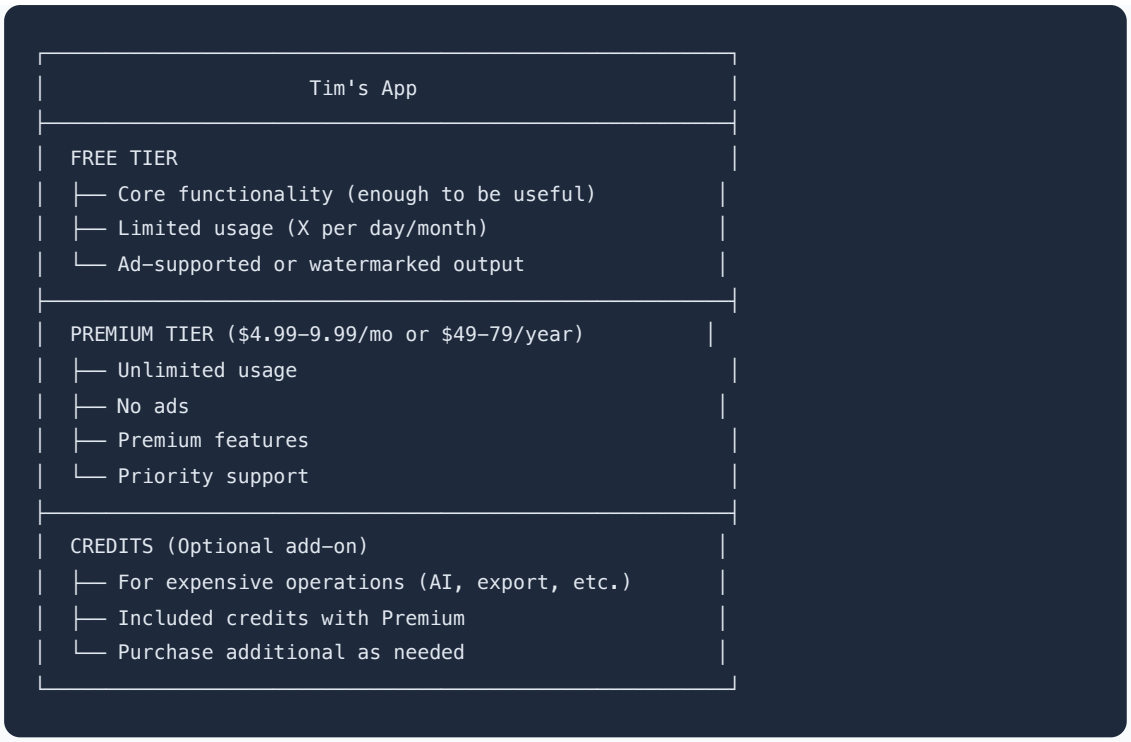
---

5. Product Recommendations for Tim

If Restructuring Tim's App

Without knowing the specific app, here are structural recommendations:

Recommended Architecture



Specific Recommendations

1. **Implement a Generous Free Tier**

- Users who've experienced value rarely request refunds
- Free tier is your best marketing and refund prevention
- Make upgrade feel like enhancement, not unlocking

2. **Offer Annual Subscriptions at Discount**

- Price annual at 2 months free (e.g., $9.99/mo or $99.99/yr)
- Annual subscribers churn less
- Pro-rate refunds if requested (fair, reduces complaints)

3. **Add Subscription Pausing**

- Surface in cancellation flow
- "Take a break for 1-3 months instead?"
- Higher save rate than discount offers

4. **Create a Downgrade Path**

- Multiple tiers allow graceful exit

- "Premium too much? Try Basic at $4.99"
- Keeps users in ecosystem

5. **Consider Usage-Based for Premium Features**

- AI features, exports, etc. as credits
- Users pay for what they use
- Reduces "unused subscription" complaints

6. **Implement 7-Day Trial (Not Longer)**

- Long enough to evaluate
- Short enough to feel urgency
- Require payment method upfront (reduces trial abuse)

7. **Add Money-Back Guarantee**

- "30-day satisfaction guarantee"
- Handle refunds graciously via support
- Use refund conversations for product feedback

Best Practices for Reducing Refund Requests

Before Purchase

| Practice | Implementation | |----------|---------------| | **Clear pricing** | Show exactly what user will pay and when | | **Feature comparison** | Table showing Free vs Premium clearly | | **Trial period** | Let users try before committing | | **Social proof** | Reviews, testimonials, user count | | **Expectation setting** | Screenshots, videos of actual product |

During Onboarding

| Practice | Implementation | |----------|---------------| | **Quick win** | Get user to core value in <2 minutes | | **Progress tracking** | Show user what they've accomplished | | **Personalization** | Customize experience based on needs | | **Feature discovery** | Gradually introduce premium features | | **Engagement hooks** | Notifications, streaks, reminders |

When User Wants to Cancel

| Practice | Implementation | |----------|---------------| | **Pause option** | "Take a break instead?" | | **Downgrade path** | Offer cheaper tier | | **Discount offer** | 50% off next billing period | | **Exit survey** | Learn why, may reveal fixable issues | | **Easy cancellation** | Don't make it hard (builds resentment) |

After Cancellation

| Practice | Implementation | |----------|---------------| | **Grace period** | Access for X days after billing ends | | **Win-back offers** | Email after 30/60/90 days | | **Product updates** | "Here's what you're missing" | | **Easy reactivation** | One-click return |

Onboarding Improvements That Reduce Churn

The First Session Matters Most

```
Goal: Get user to "aha moment" as fast as possible

Aha Moment Examples:

  • Meditation app: Complete first meditation


  • Photo editor: Edit and export first photo


  • Fitness app: Log first workout


  • Note app: Create and sync first note
```

Onboarding Checklist:

- [ ] Skip optional steps (name, profile, etc.)
- [ ] Show core value immediately
- [ ] Celebrate first accomplishment
- [ ] Explain subscription value (not just features)
- [ ] Set expectations for ongoing value

- [ ] Request permission for notifications (after value shown)
- [ ] Prompt subscription at natural point (after aha moment)

Reduce Buyer's Remorse:

1. **Day 1 Email:** "Thanks for subscribing! Here's how to get started" 2. **Day 3 Check-in:** "Have you tried [key feature]?" 3. **Day 7 Value summary:** "You've accomplished X this week" 4. **Day 14 Milestone:** "Two weeks in! Here's your progress"

Summary of Key Recommendations

1. **Adopt freemium** if possible — users who've experienced value don't request refunds

2. **Offer subscription pausing** as alternative to cancellation

3. **Implement multiple tiers** with easy downgrade paths

4. **Consider credit/usage-based billing** for expensive features

5. **Use web payments (Paddle/Lemon Squeezy)** for lower fees and direct customer relationships

6. **Add explicit money-back guarantee** — reduces anxiety, increases conversion

7. **Focus on onboarding** — first session experience predicts refund requests

8. **Make cancellation easy** but surface alternatives (pause, downgrade, discount)

9. **Handle refunds graciously** — unhappy customers become advocates when treated well

10. **Use refund requests for feedback** — "What could we have done better?"

Resources

- Apple App Store Subscriptions Guide
- Google Play Billing Subscriptions
- Paddle Documentation
- Lemon Squeezy Documentation
- RevenueCat Blog (subscription analytics)
- Apple StoreKit Documentation

2. Apple Refunds Legal

Apple In-App Subscription Refunds: Legal & Technical Deep Dive

Research Date: January 24, 2026 **For:** Tim (Realm App) **Purpose:** Understanding legal obligations and technical implementation for Apple subscription refunds

---

Executive Summary

When you sell subscriptions through Apple's App Store, **Apple is the merchant of record** — not you. This has major implications:

1. **Apple processes all refunds directly** — users request refunds through Apple, not you 2. **Apple decides whether to approve refunds** — you have no veto power 3. **You are notified after the fact** via Server Notifications 4. **You must revoke access** when notified of a refund 5. **The money Apple paid you is clawed back** from your next payout

Key Legal Principle: You are essentially a content/service provider, not a merchant. Apple handles the consumer relationship for payment purposes.

Part 1: How Apple's Refund System Works

1.1 The Refund Process (User Perspective)

Users request refunds through Apple, not through your app:

1. User visits reportaproblem.apple.com 2. Signs in with their Apple Account 3. Selects "I'd like to" → "Request a refund" 4. Chooses a reason and selects

the subscription/purchase 5. Apple reviews and decides (usually within 48 hours)

Important: Users cannot request refunds for pending charges or if they have unpaid orders.

1.2 Who Processes the Refund?

Apple processes all refunds. From Apple's Terms:

> "Each Transaction is an electronic contract between you and Apple, and/or you and the entity providing the Content on our Services."

For most users:

- **Apple Distribution International Ltd.** (Ireland) — EU, UK, and most international
- **Apple Services Pte. Ltd.** (Singapore) — Parts of Asia
- **Apple Inc.** — United States

You (the developer) are the "App Provider" — you license your app to Apple, who then licenses it to users. The direct financial relationship is between Apple and the user.

1.3 The "Custodian" Question

Q: Are developers "custodians" of user funds?

A: No. Developers are not custodians of funds. Here's why:

1. **Apple collects payment directly** from users 2. **Apple holds the funds** during the standard settlement period 3. **Apple pays developers** their share (70% or 85% after year one) minus applicable taxes 4. **When refunds occur**, Apple claws back from future payouts

You never actually hold user funds — you receive your commission after Apple's processing. This is fundamentally different from being a custodian.

From Apple's Developer Program License Agreement (Schedule 2):

> "Apple will collect payment from end users for each sale of a Licensed Application..." > "Apple shall be entitled to the following commissions..." > "In

the event of a refund... Apple will deduct the full refund amount from the amount that would otherwise be paid to you."

1.4 Apple's Refund Policies

Apple's official stance (from Apple Media Services Terms):

> "All Transactions are final. Content prices may change at any time. If technical problems prevent or unreasonably delay delivery of Content, your exclusive and sole remedy is either replacement of the Content or **refund of the price paid, as determined by Apple.**"

However, Apple also states:

> "From time to time, Apple may suspend or cancel payment or **refuse a refund request** if we find evidence of fraud, abuse, or unlawful or other manipulative behavior."

Refund eligibility varies by country/region and is subject to:

- Apple Media Services Terms and Conditions
- Local consumer protection laws (especially important in EU, Australia, New Zealand)

Part 2: Legal Considerations

2.1 Who Is Liable for Refunds?

Short Answer: Apple handles refund liability as the merchant of record.

Detailed Analysis:

| Aspect | Responsible Party | |--------|-------------------| | Processing refunds | Apple | | Deciding refund approval | Apple | | Returning funds to user | Apple | | Deducting from developer payout | Apple | | Product/service quality | Developer (App Provider) | | Revoking access post-refund | Developer | | Consumer protection compliance | Both (varies by jurisdiction) |

From the Standard EULA:

> "Subject to local law, the App Provider of any Third Party App is solely responsible for its content and warranties, as well as any claims that you may have related to the Third Party App."

This means:

- **For payment disputes:** Apple handles (as merchant)
- **For product quality claims:** Developer handles (as provider)

2.2 Developer Obligations

Contractual Obligations (Apple Developer Program Agreement)

1. **Accept refund clawbacks** — Apple deducts refunded amounts from your payouts 2. **Implement Server Notifications** — You must handle `REFUND` notifications 3. **Revoke entitlements appropriately** — When notified of refund, revoke access 4. **Maintain accurate records** — Reconcile your transactions with Apple's

Practical Obligations

1. **Don't double-dip** — Don't retain access AND have Apple refund 2. **Handle gracefully** — Users who got refunds shouldn't see errors; just revoked access 3. **Keep consumables separate** — If user consumed value before refund, that's often unrecoverable

2.3 Consumer Protection Laws

United States

- **No federal right to digital refunds** — Unlike physical goods, digital products have no automatic refund right
- **State laws vary** — California (where Apple is based) has limited digital refund requirements
- **FTC Act** — Prohibits unfair/deceptive practices, but doesn't mandate refunds
- **Credit card chargebacks** — Users can dispute through their card issuer (separate from Apple refunds)

European Union

Strong consumer protections apply:

From EU Consumer Rights: > "You have a legal guarantee also when buying digital content and digital services like videos, music, mobile apps, video games or subscriptions."

Key EU rules:

- **2-year minimum guarantee** for defective digital content
- **14-day cooling-off period** for online purchases (but often waived for digital content once download begins)
- **Right to repair/replace** if content doesn't work as advertised
- **Refund if fix impossible** within reasonable time

Important for subscriptions: > "In the case of continuous supply... the supplier is responsible for any defect that becomes apparent throughout the time that the digital content or service is to be supplied."

Apple acknowledges this in their Terms: > "In countries and regions with consumer law protections, users retain their rights under these protections. In Australia and New Zealand, consumers retain their rights under the applicable consumer protection laws and regulations."

Australia

- **Australian Consumer Law** provides strong protections
- Apps and subscriptions must be "fit for purpose"
- Automatic remedies for faulty products
- Apple cannot contract out of these protections

Key Legal Principle

You cannot waive consumer statutory rights through Terms of Service. Even though Apple says "All Transactions are final," consumer protection laws in many jurisdictions override this for defective products/services.

2.4 Chargebacks vs. Refunds

| Aspect | Apple Refund | Chargeback | |--------|--------------|------------| | Initiated by | User (via Apple) | User (via bank/card issuer) | | Decided by |

Apple | Bank/card network | | Developer notification | Server Notification | Often none directly | | Fees | No extra fee | Can incur chargeback fees | | Your control | None | Can dispute with evidence | | Impact on account | Minimal | High chargeback rates = problems |

Chargebacks are separate from Apple refunds. A user could: 1. Request an Apple refund (Apple denies) 2. Then file a chargeback with their credit card company 3. Bank may reverse the charge regardless of Apple's decision

When chargebacks happen on Apple transactions, Apple handles the dispute process with the bank, but repeated chargebacks can affect your developer account standing.

Part 3: Technical Implementation

3.1 App Store Server Notifications

Apple sends real-time notifications when subscription states change. **Version 2 notifications** (current standard) include:

Notification Types for Refunds

| Notification Type | Subtype | Meaning | |-------------------|---------|---------| | `REFUND` | — | Apple refunded a transaction | | `REFUND_REVERSED` | — | Apple reversed a previously-reported refund (rare) | | `REFUND_DECLINED` | — | Apple declined a refund request (for your info) |

Setting Up Server Notifications

1. **App Store Connect** → Your App → App Information → App Store Server Notifications 2. Enter your HTTPS endpoint URL (must be HTTPS) 3. Select Version 2 notifications 4. Apple will POST signed JSON payloads to your endpoint

Notification Payload Structure (V2)

```
{
  "signedPayload": "eyJ...(JWT)..."
}
```

Decode the JWT to get:

```
{
  "notificationType": "REFUND",
  "subtype": null,
  "data": {
    "signedTransactionInfo": "eyJ...(JWT)...",
    "signedRenewalInfo": "eyJ...(JWT)..."
  },
  "notificationUUID": "unique-notification-id",
  "version": "2.0",
  "signedDate": 1640000000000
}
```

3.2 Handling REFUND Notifications

```
// Pseudo-code for handling refund
func handleRefundNotification(notification: AppStoreNotification) {
    // 1. Verify the JWT signature (critical for security!)
    guard verifyAppleSignature(notification.signedPayload) else {
        return // Reject invalid notifications
    }

    // 2. Decode transaction info
    let transaction = decodeTransactionInfo(notification.data.signedTransactionInfo)

    // 3. Find user by originalTransactionId
    let user = findUserByOriginalTransactionId(transaction.originalTransactionId)

    // 4. Revoke entitlement
    revokeSubscriptionAccess(
        userId: user.id,
        productId: transaction.productId,
        transactionId: transaction.transactionId
    )

    // 5. Log for reconciliation
    logRefund(
        userId: user.id,
        transactionId: transaction.transactionId,
        refundDate: transaction.revocationDate,
        reason: transaction.revocationReason
    )

    // 6. Respond with 200 OK
    return HTTP.ok
}
```

3.3 Key Fields in Refund Transaction

| Field | Description | |-------|-------------| | `originalTransactionId` | Links to original purchase (use to find user) | | `transactionId` | The specific transaction refunded | | `revocationDate` | When Apple processed the refund | | `revocationReason` | `REFUNDED_BY_SUPPORT` or `REFUNDED_FOR_ISSUE` | | `productId` | Your subscription product identifier |

3.4 Detecting Refunds via API (Polling)

If you miss notifications, you can verify status via the App Store Server API:

```
GET /inApps/v1/subscriptions/{originalTransactionId}
```

Check `status` field for each subscription group:

- Active subscriptions: `status: 1` (Active)
- Refunded: Transaction will show `revocationDate` and `revocationReason`

3.5 Reconciliation Best Practices

1. **Store originalTransactionId** with user records 2. **Webhook idempotency** — Handle duplicate notifications gracefully 3. **Async processing** — Return 200 immediately, process in background 4. **Retry handling** — Apple retries failed notifications; dedupe by notificationUUID 5. **Audit trail** — Log all state changes for financial reconciliation 6. **Graceful degradation** — If webhook fails, periodic polling catches missed refunds

3.6 StoreKit 2 (Client-Side) Refund Info

In StoreKit 2, transactions have a `revocationDate` property:

```
for await result in Transaction.currentEntitlements {
    if case .verified(let transaction) = result {
        if transaction.revocationDate != nil {
            // This transaction was refunded
            // Revoke access to the associated content
        }
    }
}
```

Important: Client-side checks are supplementary. Always use server notifications as the source of truth for access control.

Part 4: Stripe vs. Apple IAP

4.1 When Can You Use Stripe Instead?

Apple's rules (Guideline 3.1.1):

> "If you want to unlock features or functionality within your app, you must use in-app purchase."

You MUST use Apple IAP for:

- Digital content consumed in the app
- Subscription access to app features
- Virtual currencies, game items
- Premium app features/upgrades

You CAN use Stripe (external payment) for:

- Physical goods/services delivered outside the app
- Person-to-person services (tutoring, consultations)
- "Reader" apps (accessing content purchased elsewhere)
- Enterprise/B2B sales (with restrictions)
- Web-based services accessed outside the app

Recent changes:

- US App Store now allows links to external payment options (with Apple still taking a commission on resulting purchases)
- EU has additional allowances under DMA regulations

4.2 Stripe Subscription Refunds

If you're using Stripe for web subscriptions (where allowed):

| Aspect | Stripe | Apple IAP | |--------|--------|-----------| | Merchant of record | **You** | Apple | | Refund decision | **You decide** | Apple decides | | Processing | You initiate via API/Dashboard | Apple handles | | Notification |

Webhook events | Server Notifications | | Chargeback risk | **You bear it** | Apple bears it | | User relationship | Direct | Through Apple |

Stripe Refund Flow

```
// Full refund
const refund = await stripe.refunds.create({
  payment_intent: 'pi_xxx',
});

// Partial refund
const refund = await stripe.refunds.create({
  payment_intent: 'pi_xxx',
  amount: 500, // $5.00 in cents
});

// Refund and cancel subscription
await stripe.subscriptions.cancel('sub_xxx');
```

Stripe Webhook Events for Refunds

- `charge.refunded` — Refund was processed
- `charge.refund.updated` — Refund status changed
- `charge.dispute.created` — Chargeback filed
- `customer.subscription.deleted` — Subscription canceled

4.3 Pros/Cons Comparison

| Factor | Apple IAP | Stripe (web) | |--------|-----------|--------------| | **Revenue** | 70-85% (Apple takes 15-30%) | ~97% (Stripe takes ~3%) | | **Refund control** | None | Full control | | **Chargeback liability** | Apple's | Yours | | **User trust** | High (Apple brand) | Depends on your brand | | **Implementation** | StoreKit API | Stripe SDK + custom backend | | **Cross-platform** | iOS only | Works everywhere | | **Compliance burden** | Low (Apple handles) | Higher (you handle) | | **User friction** | Low (Apple Pay built-in) | Higher (enter card info) |

4.4 Legal Differences

With Apple IAP:

- Apple's EULA governs user relationship
- Apple handles consumer complaints

- Apple liable for payment disputes
- Your liability limited to service delivery

With Stripe:

- **You create Terms of Service**
- **You handle consumer complaints**
- **You're liable for chargebacks and disputes**
- **You must comply with PCI DSS** (simplified with Stripe.js)
- **You must handle consumer protection law compliance directly**

Part 5: Recommendations for Tim/Realm

5.1 If Using Apple IAP (Most Likely)

1. **Implement Server Notifications V2** — This is mandatory for proper subscription management 2. **Handle REFUND notifications** — Revoke access immediately upon receipt 3. **Don't fight refunds** — You have no mechanism to do so; just process them gracefully 4. **Track for abuse** — If a user repeatedly gets refunds, you can choose not to provide services 5. **Communicate clearly** — Your app's support should direct refund requests to Apple 6. **Reconcile monthly** — Match your records with App Store Connect reports

5.2 Refund Policy Messaging

In your app/website, be clear:

> "Subscriptions are billed by Apple. To request a refund, please visit Apple's Report a Problem page. We cannot process refunds directly."

5.3 If Considering Stripe Alternative

Only viable if:

- Selling web-only access (no app features)
- Offering physical goods/services

- Operating in exempt categories (reader apps, enterprise)

If viable, Stripe gives you:

- Higher revenue (97% vs 70-85%)
- Full refund control
- Direct customer relationship
- But: Higher complexity and compliance burden

Key Takeaways

1. **You're not the merchant** — Apple is. This fundamentally changes your liability profile.

2. **Refunds happen without your consent** — Accept this and build your systems accordingly.

3. **Implement Server Notifications** — This is how you stay synchronized with Apple.

4. **Consumer protection laws vary** — EU/Australia have strong protections that override Apple's "all sales final" language.

5. **Chargebacks are separate** — Even if Apple denies a refund, users can dispute with their bank.

6. **Stripe is an option only for limited use cases** — If your monetization is app features, you're locked to Apple IAP.

7. **Document everything** — For accounting, compliance, and dispute resolution.

Sources & References

1. **Apple Developer Documentation**

- App Store Server Notifications

- StoreKit Documentation
- App Store Review Guidelines

2. **Apple Legal**

- Apple Media Services Terms and Conditions
- Apple Developer Program License Agreement
- Licensed Application EULA

3. **Apple User Support**

- Request a Refund

4. **Consumer Protection**

- EU Consumer Rights for Digital Content
- Australian Consumer Law

5. **Stripe Documentation**

- Subscription Overview
- Webhook Handling
- Cancel Subscriptions

Last Updated: January 24, 2026

3. Stripe Subscriptions

Stripe Subscriptions & Refund Handling Research

Research compiled: January 24, 2026

This document covers Stripe's subscription billing system, refund handling, compliance requirements, and comparisons with Apple IAP for Tim's app considerations.

Table of Contents

---

1. Stripe Subscription Refunds

How Refunds Work

Stripe refunds are processed back to the **original payment method** only—you cannot redirect refunds to different cards or accounts. When you issue a refund:

1. Stripe submits the request to the customer's bank/card issuer 2. The refund appears on the customer's statement within **5-10 business days** 3. If the card is expired/canceled, the issuer typically credits a replacement card or provides alternative delivery (check, bank deposit)

Key Points:

- Refunds use your **available Stripe balance** (not pending amounts)

- If balance is insufficient, card refunds are held as pending; other payment methods fail
- Fully refunded charges **cannot** be disputed/chargebacked
- Stripe can optionally email customers about refunds (configurable in Dashboard)

Partial vs Full Refunds

```
// Full Refund
const refund = await stripe.refunds.create({
  charge: 'ch_xxx', // or payment_intent: 'pi_xxx'
});

// Partial Refund
const partialRefund = await stripe.refunds.create({
  charge: 'ch_xxx',
  amount: 500, // $5.00 in cents
});
```

Rules:

- Multiple partial refunds are allowed
- Total refunds cannot exceed original charge amount
- Each refund creates a separate `refund` object
- Bulk full refunds supported in Dashboard (partial must be individual)

Proration Calculations

Prorations ensure fair billing when subscriptions change mid-cycle. Key scenarios that trigger prorations:

| Change | Effect | |--------|--------| | Upgrade plan | Credit for unused time + charge for new plan's remaining time | | Downgrade plan | Credit for unused time + charge for cheaper plan's remaining time | | Add/remove items | Prorated based on current billing period | | Change quantity | Prorated adjustment | | Add trial to active sub | Prorated credit | | Reset billing anchor | Prorated adjustment |

Example Calculation:

- Customer on $10/month plan upgrades to $20/month halfway through

- Credit: -$5 (unused half of $10 plan)
- Charge: +$10 (half month at $20 plan)
- Net: Customer pays additional $5

```
// Preview proration before making changes
const previewInvoice = await stripe.invoices.createPreview({
  customer: 'cus_xxx',
  subscription: 'sub_xxx',
  subscription_details: {
    items: [{
      id: subscription.items.data[0].id,
      price: 'price_new_xxx',
    }],
    proration_date: Math.floor(Date.now() / 1000),
  },
});

// Control proration behavior
const subscription = await stripe.subscriptions.update('sub_xxx', {
  items: [{ id: 'si_xxx', price: 'price_new_xxx' }],
  proration_behavior: 'create_prorations', // default
  // Options: 'create_prorations' | 'always_invoice' | 'none'
});
```

Webhook Events for Refunds

Essential refund-related webhooks to handle:

| Event | Description | Action | |-------|-------------|--------| | `refund.created` | Refund initiated | Log refund, notify customer | | `refund.updated` | Refund details changed (e.g., ARN added) | Update records | | `refund.failed` | Refund couldn't be processed | Contact customer for alternative | | `charge.refunded` | Charge was refunded (includes partials) | Update access/records | | `charge.dispute.funds_reinstated` | Dispute closed, funds returned | Update accounting |

```
// Webhook handler example
app.post('/webhook', async (req, res) => {
  const event = stripe.webhooks.constructEvent(
    req.body,
    req.headers['stripe-signature'],
    webhookSecret
  );

  switch (event.type) {
    case 'refund.created':
```

```
      const refund = event.data.object;
      await logRefund(refund);
      await notifyCustomer(refund.charge);
      break;

    case 'refund.failed':
      const failedRefund = event.data.object;
      // Handle failed refund - contact customer
      console.log('Refund failed:', failedRefund.failure_reason);
      // Reasons: declined, expired_or_canceled_card,
      // insufficient_funds, lost_or_stolen_card, etc.
      break;
  }

  res.json({ received: true });
});
```

Customer Portal for Self-Service

Stripe's Customer Portal allows customers to manage their own subscriptions without developer intervention.

Features:

- Download invoices
- Update payment methods
- Cancel subscriptions (immediate or end-of-period)
- Upgrade/downgrade plans
- Update billing information and tax IDs

Setup:

```
// Create a portal session
const session = await stripe.billingPortal.sessions.create({
  customer: 'cus_xxx',
  return_url: 'https://yourapp.com/account',
});

// Redirect customer to session.url
```

Configuration (Dashboard → Settings → Billing → Customer Portal):

- Enable/disable specific features
- Set allowed products for plan switching (max 10)

- Configure cancellation options
- Add branding (logo, colors)
- Enable cancellation deflection (offer coupons)

Limitations:

- Can't update subscriptions with multiple products, usage-based billing, or scheduled changes
- Can't display in iframe
- Customers modifying trialing subscriptions end the trial immediately
- Sessions expire after 5 minutes unused, then 1 hour after first use

---

2. Technical Implementation

Stripe Billing Setup

1. Create Products and Prices:

```
// Create a product
const product = await stripe.products.create({
  name: 'Pro Plan',
  description: 'Full access to all features',
});

// Create a recurring price
const monthlyPrice = await stripe.prices.create({
  product: product.id,
  unit_amount: 999, // $9.99
  currency: 'usd',
  recurring: {
    interval: 'month',
  },
});

const yearlyPrice = await stripe.prices.create({
  product: product.id,
  unit_amount: 9900, // $99.00 (save ~17%)
  currency: 'usd',
  recurring: {
    interval: 'year',
  },
});
```

2. Create a Subscription:

```
// Create customer
const customer = await stripe.customers.create({
  email: 'customer@example.com',
  payment_method: 'pm_xxx',
  invoice_settings: {
    default_payment_method: 'pm_xxx',
  },
});

// Create subscription
const subscription = await stripe.subscriptions.create({
  customer: customer.id,
  items: [{ price: monthlyPrice.id }],
  payment_behavior: 'default_incomplete', // Recommended
  expand: ['latest_invoice.payment_intent'],
});

// Handle the payment intent for 3DS if needed
if (subscription.latest_invoice.payment_intent.status === 'requires_action') {
  // Send client_secret to frontend for 3DS authentication
}
```

Subscription Lifecycle Events

| Status | Description | Action | |--------|-------------|--------| | `trialing` | In trial period | Provision access | | `active` | Payment successful, subscription running | Full access | | `incomplete` | Payment required within 23 hours | Prompt for payment | | `incomplete_expired` | First payment failed after 23 hours | No access, cleanup | | `past_due` | Payment failed, retrying | Notify customer, may continue access | | `canceled` | Subscription ended | Revoke access | | `unpaid` | Retries exhausted, no payment | Revoke access | | `paused` | Trial ended without payment method | No invoicing until resumed |

Key Webhooks to Handle:

```
// Essential subscription webhooks
const essentialWebhooks = [
  'customer.subscription.created',
  'customer.subscription.updated',
  'customer.subscription.deleted',
  'customer.subscription.trial_will_end', // 3 days before trial ends
  'customer.subscription.paused',
  'customer.subscription.resumed',
  'invoice.paid',
  'invoice.payment_failed',
  'invoice.payment_action_required', // 3DS needed
```

```
  'invoice.upcoming', // Few days before renewal
];
```

Handling Failed Payments

Automatic Smart Retries:

Stripe's Smart Retries use ML to determine optimal retry timing. Enable in Dashboard → Settings → Billing → Automatic.

Failed Payment Flow:

```
// Handle invoice.payment_failed
async function handleFailedPayment(invoice) {
  const subscription = await stripe.subscriptions.retrieve(invoice.subscription);

  // 1. Notify customer
  await sendEmail(subscription.customer, {
    subject: 'Payment Failed',
    template: 'payment-failed',
    data: {
      amount: invoice.amount_due,
      nextRetry: invoice.next_payment_attempt,
    },
  });

  // 2. Check subscription status
  if (subscription.status === 'past_due') {
    // Subscription is in grace period, still has access
    await flagAccountForFollowUp(subscription.customer);
  } else if (subscription.status === 'canceled' || subscription.status === 'unpaid') {
    // Access should be revoked
    await revokeAccess(subscription.customer);
  }
}
```

Dunning (Retry Logic)

Configure in Dashboard or programmatically:

Dashboard Settings:

- Number of retry attempts (1-4)
- Days between retries
- Action after all retries fail (cancel, mark unpaid, leave past_due)
- Send failed payment emails

- Send upcoming renewal reminders

Dunning Email Sequence Example: 1. Day 0: Payment failed, will retry 2. Day 3: Second attempt failed 3. Day 7: Final attempt before cancellation 4. Day 10: Subscription canceled

Refund API Calls

```
// Basic refund
const refund = await stripe.refunds.create({
  payment_intent: 'pi_xxx',
  reason: 'requested_by_customer', // or 'duplicate', 'fraudulent'
});

// Partial refund
const partialRefund = await stripe.refunds.create({
  payment_intent: 'pi_xxx',
  amount: 500, // $5.00
});

// Refund with metadata
const refundWithMeta = await stripe.refunds.create({
  payment_intent: 'pi_xxx',
  metadata: {
    reason: 'Customer requested downgrade',
    support_ticket: 'TICKET-123',
  },
});

// Get refund status
const refundStatus = await stripe.refunds.retrieve('re_xxx');
// status: succeeded, pending, failed, canceled
```

---

3. Legal/Compliance

PCI Compliance Requirements

Stripe's PCI Level:

- Stripe is **PCI DSS Level 1** certified (highest level)
- Processes, stores, and transmits cardholder data in a secure vault

Your Responsibilities:

| Integration Method | SAQ Type | Effort | |------------------|----------|--------| | Stripe Elements/Checkout | SAQ A | Minimal (13 questions) | | Direct API

(card data touches your server) | SAQ D | Extensive (300+ questions) | | Stripe.js tokenization | SAQ A-EP | Moderate |

Recommendations:

- **Always use Stripe Elements, Checkout, or Payment Links** - card data never touches your servers
- Stripe auto-detects your integration and provides the correct SAQ form in Dashboard
- No PCI audit needed for SAQ A merchants

Who is the Merchant of Record?

With Stripe Direct Integration:

- **YOU are the Merchant of Record (MoR)**
- Your business name appears on customer statements
- You are responsible for:
- Refund policies
- Customer support
- Chargebacks (you pay $15 per dispute)
- Tax collection and remittance
- Legal compliance in each jurisdiction

With Apple IAP:

- **Apple is the Merchant of Record**
- Apple handles refunds, disputes, and taxes
- You have less control but less liability

Stripe's Role vs Developer's Role

| Responsibility | Stripe | Developer | |---|---|---| | Payment processing | ✅ | | | Card data security | ✅ | | | PCI compliance (infrastructure) | ✅ | | | Fraud detection (basic) | ✅ | | | Dispute filing | ✅ | | | Refund decisions | | ✅ | | Tax calculation (optional) | ✅ (Stripe Tax) | ✅ (if

not using Stripe Tax) | | Tax remittance | | ✅ | | Customer support | | ✅ | | Dispute evidence | | ✅ | | Pricing decisions | | ✅ | | Refund policies | | ✅ |

Tax Implications (Stripe Tax)

Stripe Tax Features:

- Automatic tax calculation in 100+ countries
- 600+ product tax codes
- Threshold monitoring (alerts when you may need to register)
- Registration management
- Filing assistance through partners

Pricing:

- **Subscription**: Starting at $90/month (annual contract)
- **Pay-as-you-go**: 0.5% per transaction (no-code) or $0.50/transaction (API)

```
// Enable automatic tax calculation
const subscription = await stripe.subscriptions.create({
  customer: 'cus_xxx',
  items: [{ price: 'price_xxx' }],
  automatic_tax: { enabled: true },
});
```

Without Stripe Tax:

- You must calculate and collect sales tax/VAT/GST yourself
- You must track nexus/registration thresholds
- You must file and remit taxes in each jurisdiction
- Consider third-party tax services (Avalara, TaxJar)

International Considerations

Multi-Currency:

```
// Create prices in multiple currencies
const usdPrice = await stripe.prices.create({
  product: 'prod_xxx',
```

```
  unit_amount: 999,
  currency: 'usd',
  recurring: { interval: 'month' },
});

const eurPrice = await stripe.prices.create({
  product: 'prod_xxx',
  unit_amount: 899,
  currency: 'eur',
  recurring: { interval: 'month' },
});
```

Additional Fees for International:

- +1.5% for international cards
- +1% if currency conversion required
- Local payment methods may have different rates

Regional Considerations:

- **EU**: GDPR compliance, SCA (Strong Customer Authentication) requirements
- **India**: RBI recurring payment guidelines, additional verification
- **Brazil**: CPF/CNPJ requirements
- **Australia**: GST collection requirements

---

4. Stripe vs Apple IAP Comparison

Fee Comparison

| Metric | Stripe | Apple IAP | |--------|--------|-----------| | Standard Fee | 2.9% + $0.30 | 30% (or 15% for Small Business Program*) | | International Cards | +1.5% | Included | | Currency Conversion | +1% | Included | | Subscription Fee | +0.7% (standard) | Included | | Dispute Fee | $15 per dispute | Apple handles | | Refund Fee | Original fee not returned | Apple handles |

*Apple Small Business Program: 15% for developers earning < $1M/year

Example: $9.99/month subscription, US customer: | | Stripe | Apple IAP (30%) | Apple IAP (15%) | |-|--------|-----------------|-----------------| | Gross | $9.99 | $9.99 | $9.99 | | Processing | -$0.59 (2.9% + $0.30 + 0.7%) | -$3.00 | -$1.50 | | **Net** | **$9.40** | **$6.99** | **$8.49** |

User Experience Differences

| Aspect | Stripe | Apple IAP | |--------|--------|-----------| | Payment Flow | Redirect to Stripe or embedded form | Native iOS sheet | | Payment Methods | Cards, ACH, PayPal, Klarna, etc. | Apple Pay, cards on file | | Saved Cards | Stripe saves across your apps | Apple ID saves across all apps | | Family Sharing | Must implement yourself | Built-in | | Receipt Management | Email receipts, Dashboard | iOS Settings → Apple ID | | Refund Request | Contact developer | Settings → Apple ID (easy) |

Refund Handling Differences

| Aspect | Stripe | Apple IAP | |--------|--------|-----------| | Who decides | Developer | Apple (usually auto-approved) | | Time to refund | 5-10 business days | Often instant | | Notification | Webhook + optional email | App Store notification (delayed) | | Developer control | Full control | Almost none | | Proration | Automatic calculations | No proration | | Partial refunds | Supported | Not supported |

Apple Refund Problem:

- Apple often approves refunds without notifying developers promptly
- Customer may keep access while refund is processed
- Server-side receipt validation is essential to detect refunds

```
// iOS: Check subscription status server-side
// Apple's App Store Server API or StoreKit 2
```

When to Use Which

Use Apple IAP When:

- ✅ Required by App Store guidelines (digital goods consumed in-app)

- ✅ Selling content like premium features, subscriptions for app functionality
- ✅ Want simple implementation
- ✅ Targeting Apple ecosystem heavily
- ✅ Want Family Sharing support

Use Stripe When:

- ✅ Selling physical goods or services consumed outside the app
- ✅ Selling person-to-person services (e.g., coaching, consulting)
- ✅ Cross-platform subscriptions (iOS, Android, Web)
- ✅ Want maximum revenue retention
- ✅ Need complex billing (metered, tiered, prorations)
- ✅ Want control over refund policies
- ✅ B2B sales with invoicing needs

Can Use Both:

- Different SKUs for in-app vs web signups
- "Reader" apps (content purchased elsewhere, consumed in-app)
- Physical goods alongside digital subscriptions

⚠️ **App Store Guidelines Warning:** Apple requires IAP for:

- Subscriptions to content in the app
- Premium features
- Consumables (coins, gems, etc.)
- Non-consumables (unlock levels, filters)

Stripe is allowed for:

- Physical goods/services
- Services consumed outside the app
- Person-to-person services
- Business services (B2B)

---

5. Best Practices

Subscription Management UX

Onboarding:

```
1. Show clear pricing with monthly/annual toggle
2. Highlight savings for annual (e.g., "Save 17%")
3. Offer trial period (7-14 days common)
4. Collect payment method upfront (higher conversion)
5. Send welcome email with key features
```

During Subscription:

```
1. Show current plan and renewal date in-app
2. Remind before trial ends (3 days, 1 day, day-of)
3. Send receipt emails for each charge
4. Provide easy access to billing portal
5. Show usage/value delivered (engagement emails)
```

Cancellation Flow:

```
1. Ask why they're canceling (collect feedback)
2. Offer alternatives:
   • Pause subscription instead
   • Downgrade to cheaper plan
   • Offer discount/coupon
3. Confirm cancellation clearly
4. Send confirmation email
5. Remind them before access ends
6. Win-back email after 30/60/90 days
```

Reducing Involuntary Churn

Involuntary churn = customers who want to stay but payment fails.

Strategies:

1. **Enable Smart Retries** (Dashboard → Billing → Automatic)

- Stripe uses ML to find optimal retry times
- Much better than fixed schedules

2. **Card Updater**

- Stripe automatically updates expired cards
- Works with major card networks

3. **Pre-dunning Emails**

- Notify before card expires
- Link to update payment method

4. **Multiple Payment Methods**

- Allow backup payment methods
- Fallback automatically

5. **Grace Period**

- Don't revoke access immediately
- Give 3-7 days to fix payment issues

```
// Example: Card expiry check
const customer = await stripe.customers.retrieve('cus_xxx', {
  expand: ['default_source'],
});

if (customer.default_source) {
  const expMonth = customer.default_source.exp_month;
  const expYear = customer.default_source.exp_year;
  const now = new Date();

  // Warn if expiring within 30 days
  const expiry = new Date(expYear, expMonth - 1);
  const daysUntilExpiry = (expiry - now) / (1000  60  60 * 24);

  if (daysUntilExpiry < 30) {
    await sendCardExpiryWarning(customer.email);
  }
}
```

Win-Back Strategies

1. **Cancellation Survey** - Understand why they left 2. **Immediate Offer** - Discount if they stay 3. **30-Day Email** - "We miss you" + what's new 4. **60-Day Email** - Limited-time offer to return 5. **90-Day Email** - Feature highlights + discount

```
// Track cancellation reason
const subscription = await stripe.subscriptions.update('sub_xxx', {
  cancel_at_period_end: true,
  cancellation_details: {
    comment: 'Too expensive',
    feedback: 'too_expensive', // or: missing_features, switched_service, unused, other
  },
});

// Later: Win-back campaign targeting 'too_expensive' segment
```

Pause Subscription Options

Instead of canceling, offer pause:

```
// Pause collection - keep subscription active, don't charge
const subscription = await stripe.subscriptions.update('sub_xxx', {
  pause_collection: {
    behavior: 'void', // void invoices during pause
    resumes_at: Math.floor(Date.now() / 1000) + (30  24  60 * 60), // 30 days
  },
});

// Options:
// - 'void': Invoices are voided (customer not charged, period doesn't count)
// - 'keep_as_draft': Invoices stay draft (can charge later)
// - 'mark_uncollectible': Mark invoices uncollectible (for accounting)

// Resume early
const resumed = await stripe.subscriptions.update('sub_xxx', {
  pause_collection: '', // Unset to resume
});
```

Pause UX Tips:

- Limit pause duration (e.g., max 3 months)
- Limit pause frequency (e.g., once per year)
- Extend features during pause (maintain engagement)

- Send "welcome back" email on resume

Quick Reference: Essential Webhooks

```
// Minimum webhooks to handle for subscriptions
const requiredWebhooks = {
  // Subscription lifecycle
  'customer.subscription.created': 'Provision access',
  'customer.subscription.updated': 'Check for status changes',
  'customer.subscription.deleted': 'Revoke access',
  'customer.subscription.trial_will_end': 'Remind customer (3 days before)',

  // Payments
  'invoice.paid': 'Confirm access, send receipt',
  'invoice.payment_failed': 'Notify customer, may pause access',
  'invoice.payment_action_required': 'Customer needs to authenticate',

  // Refunds
  'refund.created': 'Log refund',
  'refund.failed': 'Handle failed refund',
  'charge.refunded': 'Update access/records',
};
```

Summary Recommendations for Tim

1. **Use Stripe for web/cross-platform signups** - Higher revenue retention (keep ~94% vs ~70%) 2. **Use Apple IAP where required** - In-app digital goods per App Store guidelines 3. **Implement proper webhook handling** - Critical for subscription state management 4. **Enable Stripe Tax** if selling internationally - Simplifies compliance significantly 5. **Use Customer Portal** - Reduces support burden, customers can self-serve 6. **Implement pause instead of cancel** - Reduces churn significantly 7. **Enable Smart Retries** - Recovers ~30% of failed payments automatically 8. **Plan for refund handling** - With Stripe you control policies; with Apple you don't

---

Last updated: January 24, 2026