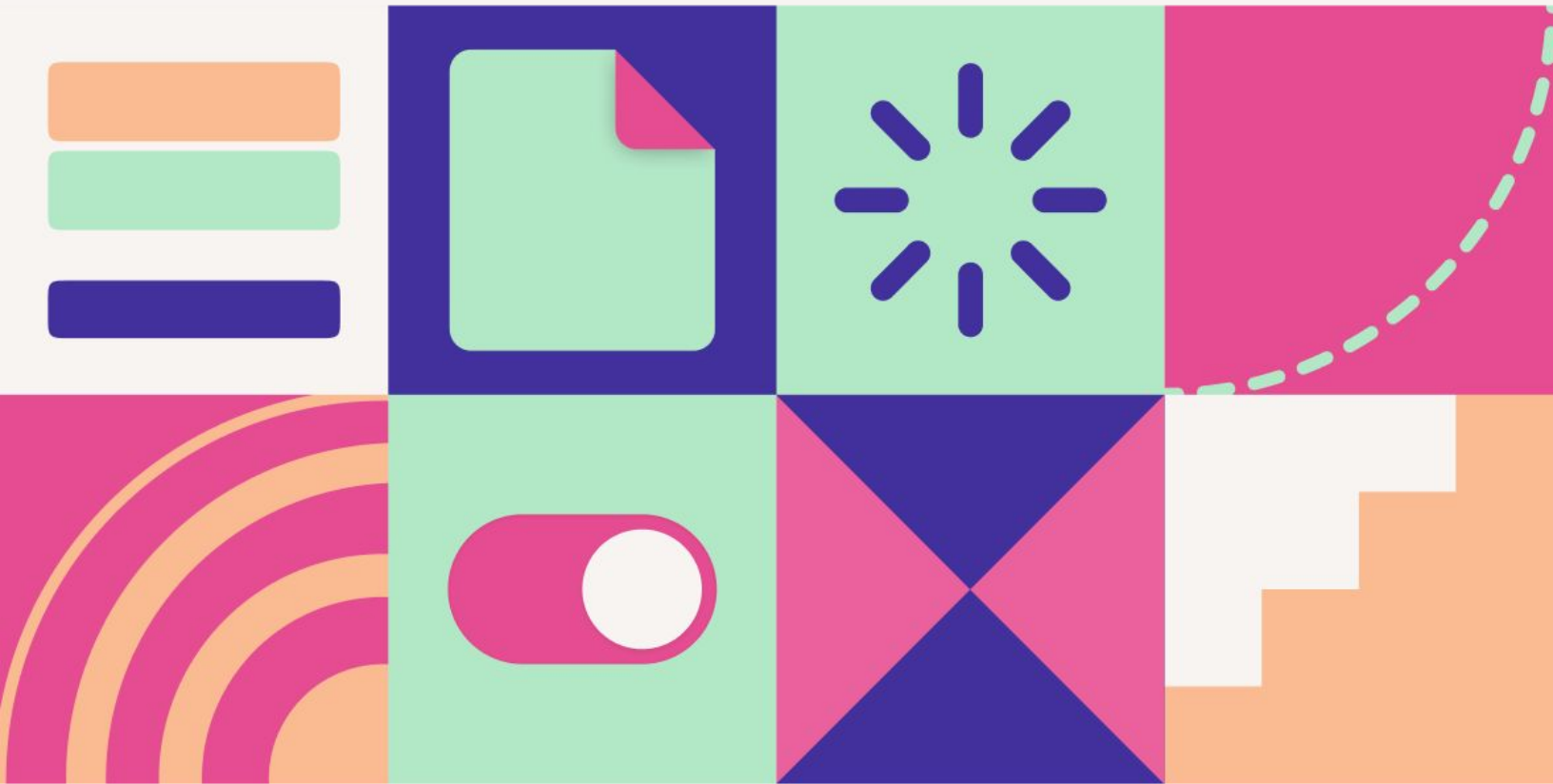
adapty

A refund guide to saving your app revenue

Refunds are inevitable. Losing revenue isn't!
Learn how keep what's yours.



Letter from Adapty's CEO

> Adapty was created with one goal: to help app developers grow their revenue without unnecessary complexity. Along the way, we ran into many challenges, and one stood out – refunds. They slowly reduce your revenue, yet most developers either don't realize their impact or don't have the tools to deal with them effectively.
>
> That's why we built Refund Saver.
>
> But this guide isn't about Adapty – it's about you. It's about taking control of your app's revenue, reducing refund losses, and making your business more profitable. We'll break down why refunds happen, what you can do about them, and how other apps are already saving thousands of dollars every month.
>
> Inside, you'll find real insights and actionable solutions. Some might challenge the way you think about refunds, while others will be quick and easy wins. But together, they'll help you stop losing money.
>
> Take what works for you. Question what doesn't. Just don't ignore refunds.

Vitaly Davydov,
CEO at Adapty

Follow us:

What’s inside

Introduction to refunds

Refunds might seem like a minor issue, but they directly impact your revenue, retention, and growth. If you're not tracking and managing them, you're losing money without even realizing it.

Lost revenue	Each refund means direct revenue loss.
Lower LTV	Refunded users never contribute to lifetime value.
Higher churn	Refund requests often indicate dissatisfaction.
Wasted acquisition spend	You paid to acquire the user, but the revenue disappears.
Distorted metrics	Refunded transactions can make your numbers look better than they actually are.

The problem → you don't control refunds

- Apple owns all payments and refunds.
- Developers can't approve or deny refund requests.
- Refunds often don't show up clearly in analytics.

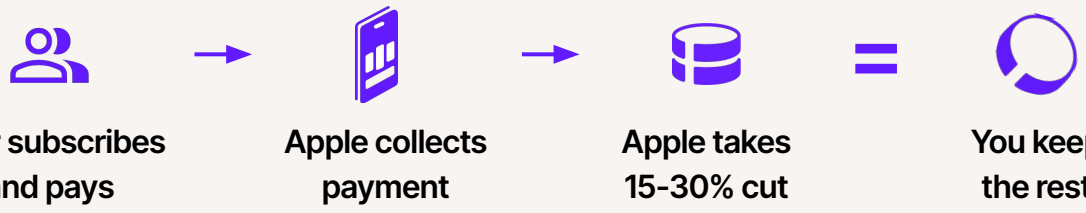
The longer users stay, the easier it is to grow without constantly spending on acquisition.

Understanding how payments and refunds work

Let's start with the basics: when it comes to payments in the App Store, **Apple owns the system.** You don't handle transactions, you don't get direct access to user payments, and you definitely don't decide who gets a refund. That's all Apple.

1. How payments work in the App Store

When a user subscribes to your app, they're not actually paying you.



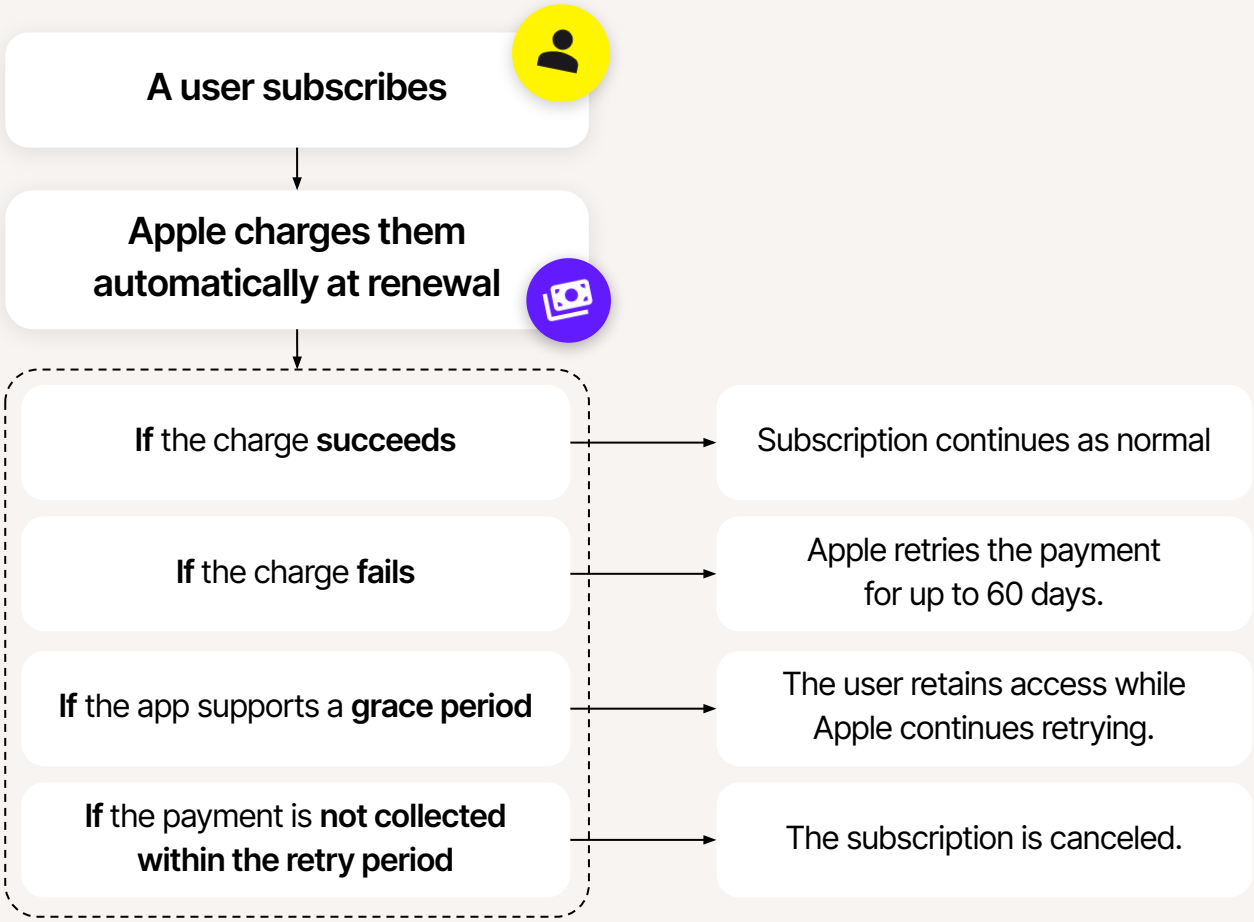
They're paying Apple. Apple collects the money, takes a cut (15-30%), and then passes the rest on to you.

- You never see the user's payment details.
- You have no control over chargebacks, disputes, and payment tokens
- If a refund happens, you might not even know about it until you check your analytics.

Good for security? Yes. Good for developers who want control? Not so much.

2. How payments work in the App Store

Apple designed subscriptions to be frictionless for users. It's automatic, which is great for recurring revenue until something goes wrong. Here's what happens:



Why this matters

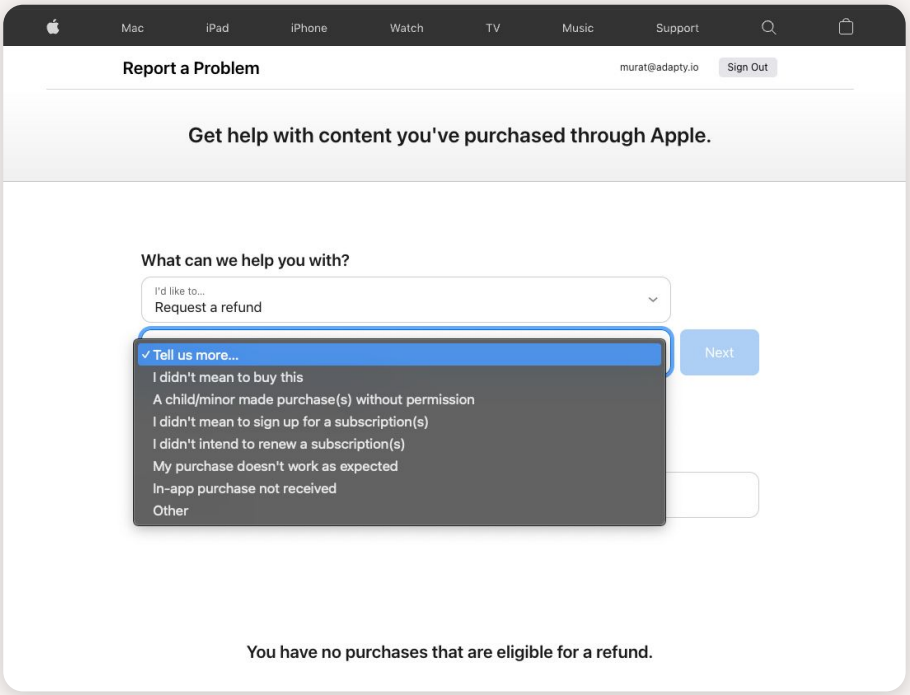
- Users often forget they subscribed, get charged, and then request a refund.
- Some users abuse free trials, cancel before renewal, and still request refunds.

Developers have no way to stop Apple's automatic retry attempts, even if they know the user won't pay.

3. How Apple issues refunds

Step 1. A user requests a refund

They go to Apple's Report a Problem page, write a quick explanation, and submit their request.



Apple's Report a Problem page

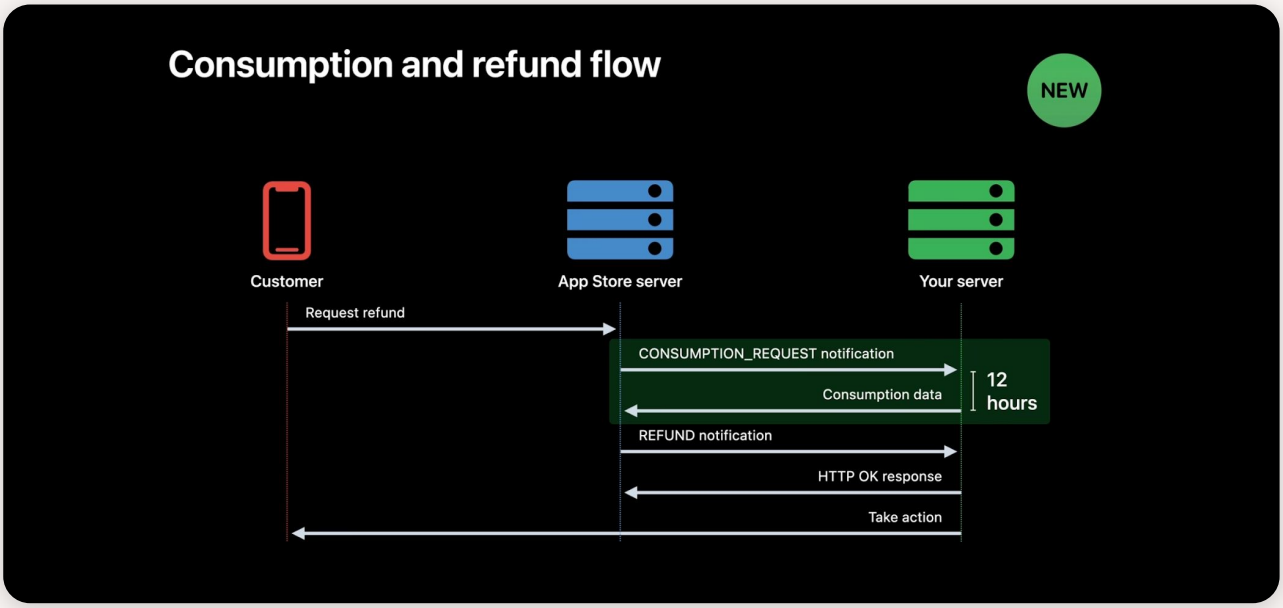
Step 2. You get notified

Apple sends an event to your server, which you can handle with the V2 API. This is your chance to prove the user actually consumed the purchase. You have just 12 hours to respond using Apple's API. Required data includes:

- How long the user's had an account.
- How much they used the app (playtime, features, etc.).
- Whether the subscription was delivered properly. Your stance on the refund – approve, deny, or no preference.

Step 3. Apple makes the final call

Even if you submit all the data on time, Apple makes the final call. But if they deny a refund, you can request a review for that transaction. Their system prioritizes user satisfaction, so refunds often get approved.



Apple refund flow presented at WWDC21

Developers do have the option to implement an in-app customer portal where users can view subscriptions and purchase history. Yet, few bother to do it. It's complicated to set up. In the end, it still routes users back to Apple's system, leaving them with no real control.

Some refunds make sense. Maybe they clicked "buy" by accident, or the app didn't work as it should. But let's be honest: not all refund requests are that straightforward.
For example:

A user sends a questionable refund request

Use-and-refund

Someone buys a VPN subscription, uses it for one task, and requests a refund.

"Accidental" purchases

A user enjoys your app but insists they bought the subscription by mistake.

Trial abuse

They sign up for a free trial, use premium features, and request a refund the moment they're charged.

The impact of refunds on your app business

Refunds can affect your growth, retention, and the way you track your app. Let's break it down.

1. How to calculate your refund rate

To understand the weight of refunds, you first need to track how often they happen. Here's a simple formula:

$$\text{Refund rate (\%)} = \frac{\text{Total refunds}}{\text{Total transactions}} \times 100$$

For example, if you had 1,000 transactions last month and 50 of them were refunded, your refund rate is 5%.

> **Note:** A high refund rate means you're losing revenue, distorting your data, and potentially hurting your app's reputation. Keeping this number low should be a priority.

2. Refund rate benchmarks

Not all refunds are bad but too many can be a red flag. The question is: how do you compare to the rest of the industry?

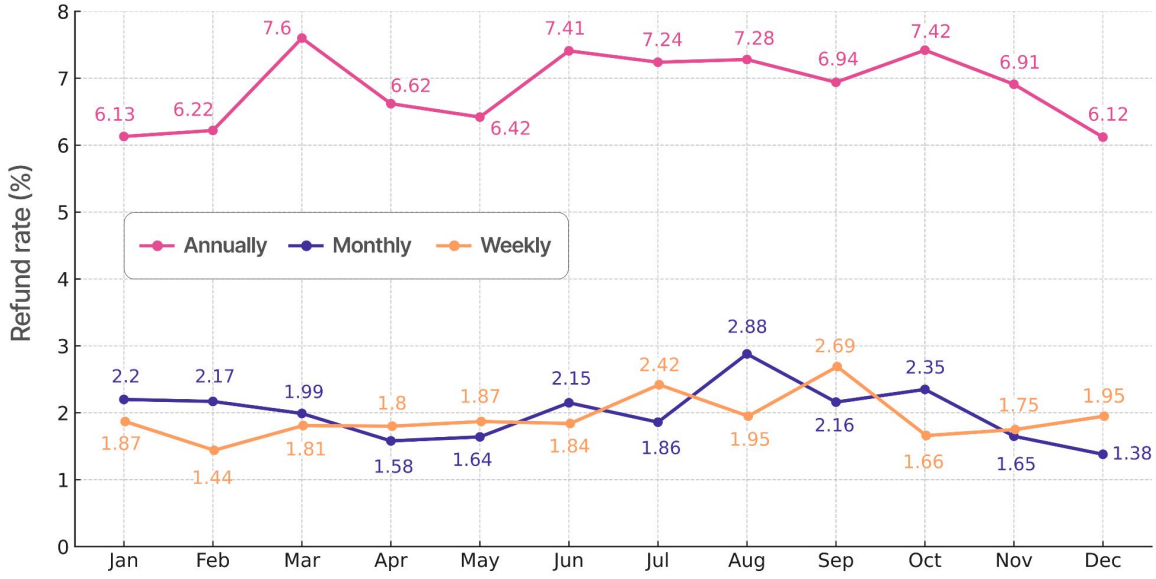
We've analyzed iOS refund data in the U.S. across different subscription durations, and categories to help you understand what's "normal" and when it's time to act.

Refund rate by subscription duration:

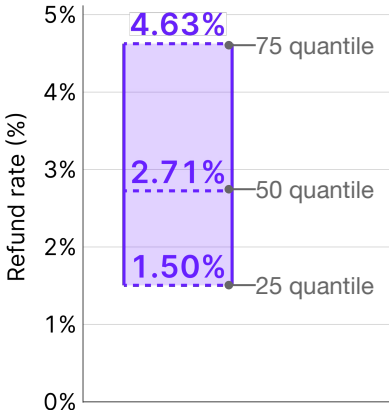
Annual subscriptions	Have the highest refund rates (6%–7.6%) likely due to cost and trial periods.
Monthly subscriptions	They are more stable but vulnerable to seasonal shifts.
Weekly subscriptions	They are the most resilient but generate lower ARPU.

What's a 'normal' refund rate?
Median refund rate: 2.71% (5% of apps have refunds below 1.5%; 75% of apps stay under) 4.63% - aim to be here!

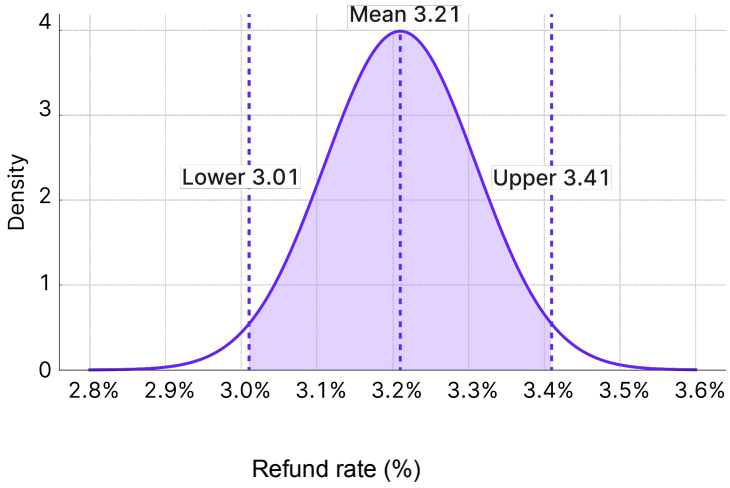
Average Refund rate by month & subscription duration (App Store) [1]



Refund rate distribution (App Store)[2]



95% confidence interval for the mean Refund rate (App Store)



1. How we calculated the the refund rate:
   The graph shows the refund rate based on the initial payment date. The conversion represents the share of those payments that were refunded. This is essentially a cohort analysis of refunds, which is better for tracking trends and long-term behavior.

   Refund rates in recent months appear lower, especially for annual plans at the end of 2024. As time passes, these numbers will increase. By the end of 2025, for example, the refund rate for late 2024 will be higher.

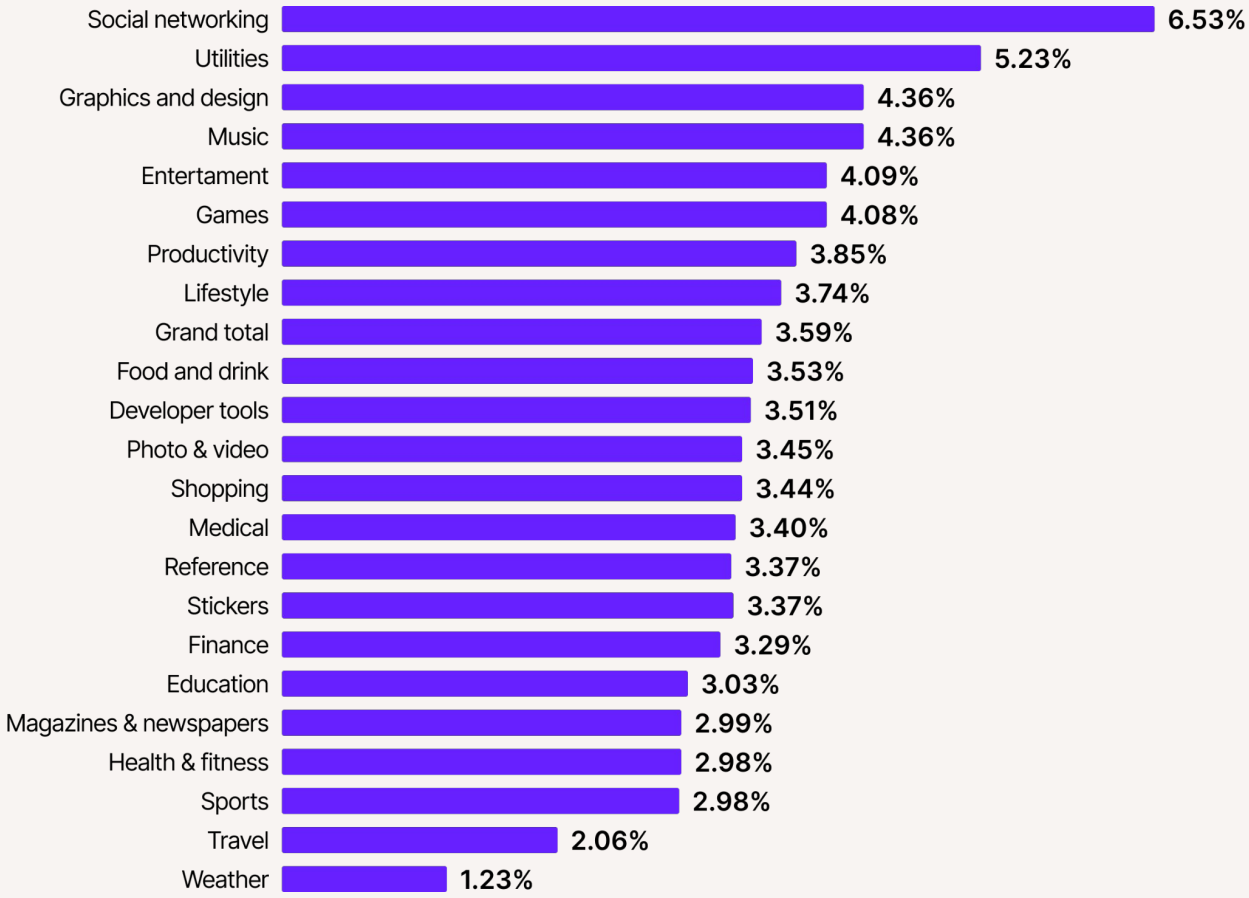
2. Box plots to better illustrate refund rate distribution.

Refund rate by app category

Not all apps are created equal when it comes to refunds.

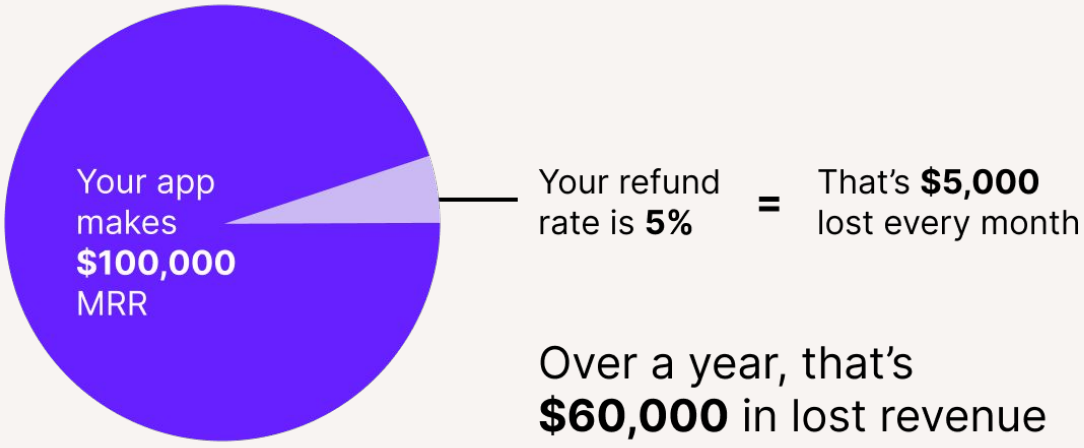
1	**Social Networking**	Top the charts, with refund rates hitting 6.5% – likely due to subscription-based dating, communities, or premium messaging.
2	**Utilities & Productivity**	Follow closely, with refund rates around 5% – probably because users realize they don't need them long-term.
3	**Finance, Health & Fitness, and Entertainment**	Land between 3-4%, which is considered average.
4	**Education & Books**	Tend to have lower refund rates (~2-3%), as users see them as an investment.

Average refund rate by category (App Store)



3. The true cost of refunds

It's easy to overlook refunds as just another line item in your revenue reports. But when you zoom out, the numbers start to add up:



That's a full-time salary. A marketing budget. A new feature you could have built. And yet, refunds quietly chip away at your business.

How to handle refunds

You can't stop refunds completely. Some users will always ask for their money back. But you can take steps to minimize unnecessary refunds, prevent abuse, and even influence Apple's decision when a refund request is made.

1. Apple controls refunds. Not you.

Apple owns all subscription payments, and developers don't have direct control over them. When a user requests a refund, Apple decides whether to approve or deny it.

In 2021, Apple introduced the Consumption API for in-app purchases (excluding subscriptions) to address refund-related issues in gaming apps.

Imagine a user buys in-game currency, spends it on a character skin, plays for a while, and then requests a refund. Since Apple doesn't track the exact in-game usage, this creates a loophole where users can abuse the system while developers struggle to prove what was actually consumed.

To combat this, Apple extended the Consumption API to subscription-based apps in 2024. This update allows developers to send additional consumption data to Apple, helping reduce refund fraud when users have actively used the app but still request a refund.

2. What data can you provide to Apple?

Apple's Refund API allows developers to submit details about user activity, helping Apple make a more informed refund decision. However, before sharing any data, you must obtain valid user consent, as Apple requires compliance with data privacy regulations. **Consider adding this requirement to your privacy policy or terms of service.**

When a user requests a refund, Apple sends a `CONSUMPTION_REQUEST` notification via AppStore Server API V2 to your webhook. You have 12 hours to respond, and you reply must include `transactionId.`

To respond to the API request, you need to send a **ConsumptionRequest** object. Here's what it includes:

Name	Type	Description
`accountTenure`	integer	The duration of time that the customer has been using your app. Values: 0 (Undisclosed), 1 (0 to 3 days), 2 (3 to 10 days), 3 (10 to 30 days), 4 (30 to 90 days), 5 (90 to 180 days), 6 (180 to 365 days), 7 (Greater than 365 days)
`appAccountToken`	string	A UUID that associates the consumption data with an account in your app. Must match the `appAccountToken` from the original transaction, if provided. Max length: 36 characters.
`consumptionStatus`	integer	The extent to which the customer has consumed the in-app purchase. Values: 0 (Undisclosed), 1 (Not consumed), 2 (Partially consumed), 3 (Fully consumed). We advise using 3.

Name	Type	Description
`customerConsented`	boolean	Indicates whether the customer explicitly consented to provide consumption data to the App Store.
`deliveryStatus`	integer	The delivery status of the in-app purchase. Values: 0 — The app delivered the consumable in-app purchase and it's working properly, 1 — The app didn't deliver the consumable in-app purchase due to a quality issue, 2 — The app delivered the wrong item, 3 — The app didn't deliver the consumable in-app purchase due to a server outage, 4 — The app didn't deliver the consumable in-app purchase due to an in-game currency change, 5 — The app didn't deliver the consumable in-app purchase for other reasons. We advise using 0.
`lifetimeDollarsPurchased`	integer	The total amount in USD spent by the customer in your app over their lifetime. Values: 0 (Undisclosed), 1 ($0 to $50), 2 ($50 to $100), 3 ($100 to $200), 4 ($200 to $500), 5 ($500 to $1,000), 6 ($1,000 to $2,000), 7 (Greater than $2,000). You need to calculate it in your history of user purchases. If you use Adapty, we do it for you.
`lifetimeDollarsRefunded`	integer	The total amount in USD refunded to the customer from your app over their lifetime. Values: 0 (Undisclosed), 1 ($0 to $50), 2 ($50 to $100), 3 ($100 to $200), 4 ($200 to $500), 5 ($500 to $1,000), 6 ($1,000 to $2,000), 7 (Greater than $2,000).

Name	Type	Description
`sampleContentProvided`	boolean	Indicates whether free sample content was provided to the customer for the in-app purchase (e.g., a trial or promotional content).
`userStatus`	integer	The current subscription status of the customer in your app. Values: 0 (Undisclosed), 1 (Active subscriber), 2 (Canceled subscriber), 3 (Lapsed subscriber), 4 (Never subscribed).
`platform`	integer	The platform on which the customer consumed the in-app purchase. Values: 0 (Undisclosed), 1 (Apple platform), 2 (Non-Apple platform)
`playTime`	integer	The amount of time the customer has spent in your app. Values: 0 (Undisclosed), 1 (0–5 minutes), 2 (5–60 minutes), 3 (1–6 hours), 4 (6–24 hours), 5 (1–4 days), 6 (4–16 days), 7 (Over 16 days). This is a legacy property from games, but you still use it for apps.
`refundPreference`	integer	You preferred outcome from the refund request. Values: 0 (Undisclosed), 1 (Prefer Apple grants refund), 2 (Prefer Apple declines refund), 3 (No preference). We think it only makes sense to send 2.

3. How different apps handle refund requests

Not all refunds are bad but too many can be a red flag. The question is: how do you compare to the rest of the industry?

	Issue	Solution
Gaming apps (in-app purchases & items)	Users buy virtual currency, spend it, then request a refund.	Provide transaction history, in-game playtime, and currency usage logs to Apple to prove the purchase was consumed.
VPN & security apps	Users subscribe, use premium features, and request refunds after months.	Submit session logs (anonymized), connection frequency, and feature usage to Apple.
Productivity & subscription apps	Users forget about their trial, get charged, and dispute the payment.	Send login activity, premium feature usage, and time spent in the app to prove engagement.

How to stop losing money on refunds

1. Apple's Refund API vs. Refund Saver: What's the difference?

Apple's Refund API	Adapty's Refund Saver
✅ Helps you fight refunds	✅ Fully automated process
✅ Allows you to send Apple proof that a user consumed premium features before requesting a refund	✅ Reduces refunds without the manual effort
❌ No automation — you have only 12 hours to submit the right data	

2. Is building your own refund automating system worth it?

At first, developing an in-house refund management system seems like a smart move. But here's what it really takes:

Step 1. Integrate Server Notifications V2

- Create a secure HTTPS endpoint for Apple's refund notifications.
- Handle multiple notification types (e.g., `CONSUMPTION_REQUEST`, `DID_RENEW`).
- Verify security tokens and ensure data integrity.

Step 2. Collect consumption data

- Track user behavior (playtime, features used, account tenure).
- Pull data from Firebase, Amplitude, or your internal analytics tools.

Step 3. Stay compliant with Apple's changing policies

- Apple frequently updates refund policies, requiring constant maintenance.

Step 4. Develop custom analytics & reporting

- Monitor refund trends and track recovery rates.
- Build dashboards or integrate with external BI tools.

For most teams, the cost, time, and complexity just don't add up.

	Build your own system	**Refund Saver**
Set up time	4-6 months for development, testing, and launch	One click! Enable it directly from your Adapty dashboard
Maintenance	Continuous updates to stay compliant	Always updated to Apple's latest policies
Data integration	Manually consolidate data from multiple sources	Pulls data automatically from your Adapty dashboard and integrations
Cost	High upfront cost + ongoing resources	Included in your Adapty Pro, Pro+, and Enterprise subscription
Refund dispute success rate	Depends on your setup's accuracy	80% success rate across clients. Cut down refund rate by 50%
Data insights	Build your own or integrate 3rd-party tools	Built-in dashboard for tracking refund trends
Scalability	Continuous engineering efforts to process higher volumes within tight deadlines	No limit on volumes
Personalized refund policies	Custom-built logic required	Coming soon: control who gets a refund and who doesn't

3. Real-world examples: How apps are saving money

Fotorama — AI Photo Generator
Health & Fitness

How Fotorama reduced app's subscription refund rate by 40% with Refund Saver.
And made Adapty free.

Fotorama, an AI photo generator app, was losing revenue to high refund rates. By implementing Refund Saver, they:

- **Reduced** their subscription **refund rate by 40%** in just two weeks.
- **Increased daily revenue by $2,500+** without gaining additional users.
- **Saved over $75,000** per month by cutting down refunds.

Read the full case study

Pepapp
Health & Fitness

How Pepapp earned a 400% ROI on Adapty in just 7 days with Refund Saver

Pepapp is a female health app with over 5 million users run by a small indie team. To scale their refund management system, they used Refund Saver. In the first seven days:

- They **recovered 96.7%** of potentially lost **revenue** from unfair refunds.
- **The money saved** was nearly triple the cost of their Adapty's subscription

Read the full case study

Why Adapty Refund Saver?

Cut refunds by 40%

Keep more of your hard-earned revenue with smarter refund handling.

Zero extra effort

It's fully automated - sit back and let it do the work for you.

Adapty pays for itself

Recover enough revenue, and Adapty can practically cost you nothing.

Refund Saver recovers $576K weekly for developers, delivering an instant 3x ROI that covers Adapty's cost and more.

$3.38M
Total money saved from refunds

1.22K
Apps with enabled refund saver

Up to **80%**
of requests declined

Schedule a demo

See how Refund Saver can protect your app revenue.