Preply

Securing High-Risk Django Applications

Lessons from the Payment Domain

PyCon Italia 2026 · Bologna

Dmytro Khmelenko · Software Engineer, Payments Tech Lead


INCIDENT ALERT
RESPONSE REQUIRED
CHANNEL 1


02:17
INCIDENT ALERT
RESPONSE REQUIRED
CHANNEL 1

Preply

Sounds familiar?

Preply

Securing High-Risk Django Applications

Lessons from the Payment Domain

PyCon Italia 2026 · Bologna

Dmytro Khmelenko · Software Engineer, Payments Tech Lead

Preply

Hi, I'm Dmytro

- 15+ years professional experience
- Today - Tech Lead in Payments team at Preply
- Interests - big data, cybersecurity, endurance sports
- Bias - I've spent more time on what goes wrong than what goes right

We are building a tutor-led, AI enhanced learning platform

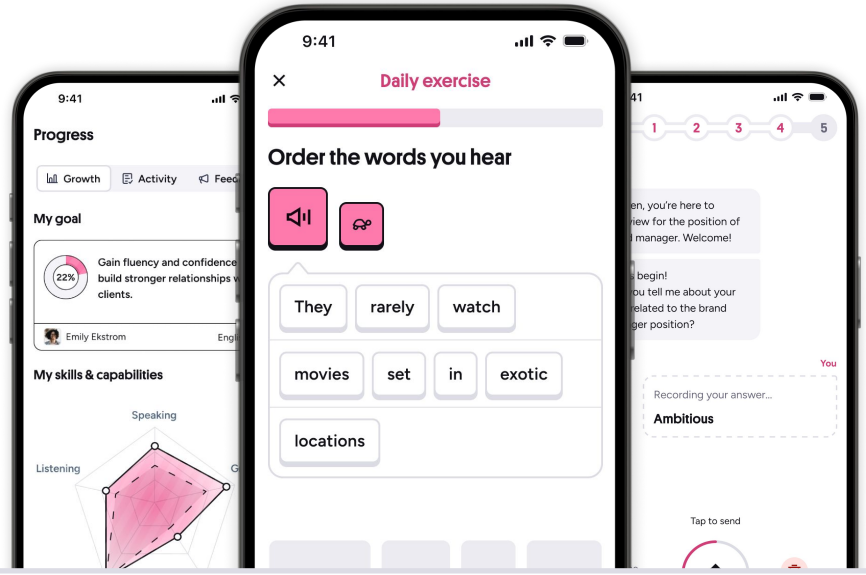
Preply

The best Marketplace

A marketplace of amazing tutors, supported by best-in-class classroom, scheduling & tools

Enhanced with AI

A learning platform where your tutor experience super-powered with AI



>2M
Active Learners

100K
Active Tutors

2M
Monthly Lessons

1000+
clients
(B2B business)


02:17
INCIDENT ALERT
RESPONSE REQUIRED
CHANNEL 1

Preply

Payments are unforgiving

Critical

Attackers are paid to find your bugs. The threat model is professional.

Auditable

Regulators, processors, and finance will ask: show me.

Irreversible

A wrong charge becomes a refund, a CS ticket, and a chargeback fee.

Preply

Django defaults are great… for blogs

Usually fine

- auto_now / auto_now_add on timestamps
- DEBUG = False in production
- CSRF + SecurityMiddleware enabled
- django.contrib.auth password hashing

Re-evaluate for money

- model.save() mid-request, no locks
- ATOMIC_REQUESTS hides transaction boundaries
- ModelAdmin with full edit on financial tables
- Errors swallowed by generic 500 handlers

Preply

6 Lessons to protect your payment system

Preply

... and your sleep

Preply

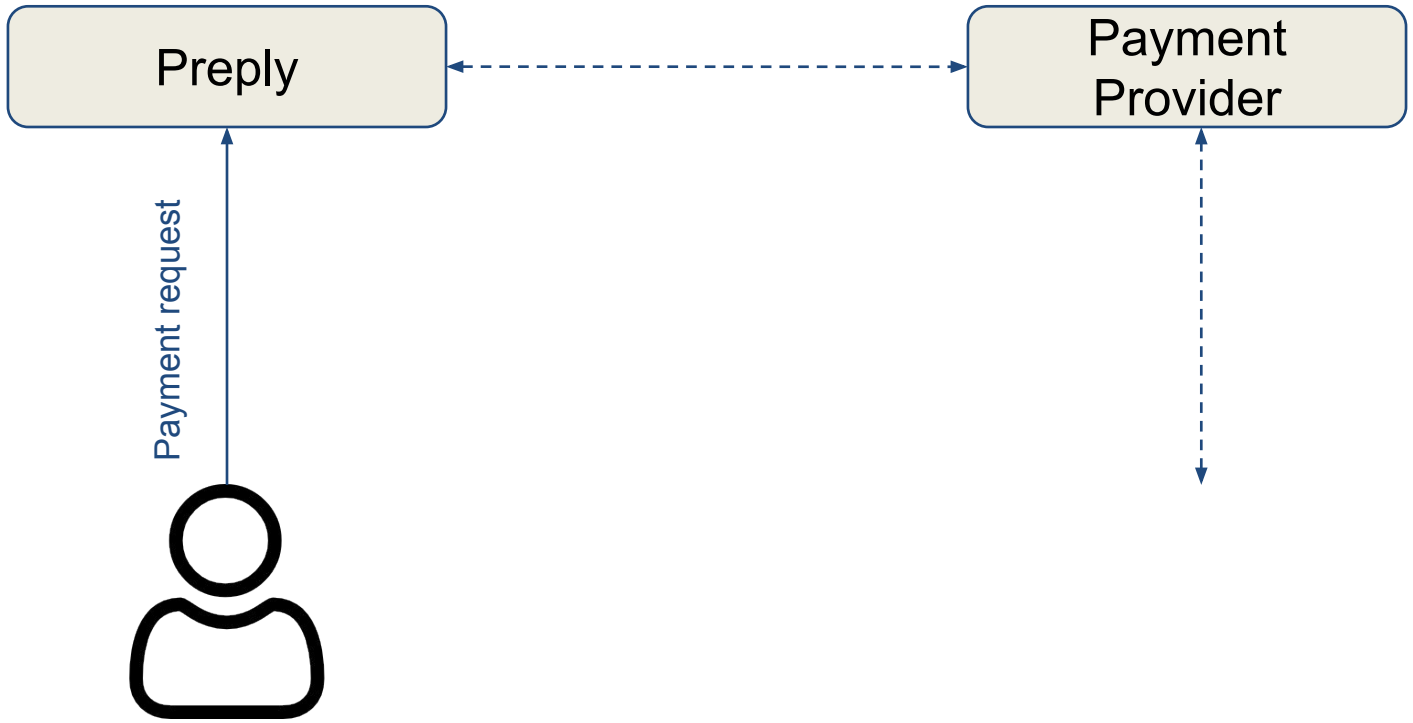
01

Transaction integrity & race conditions

If two requests can race, they will.

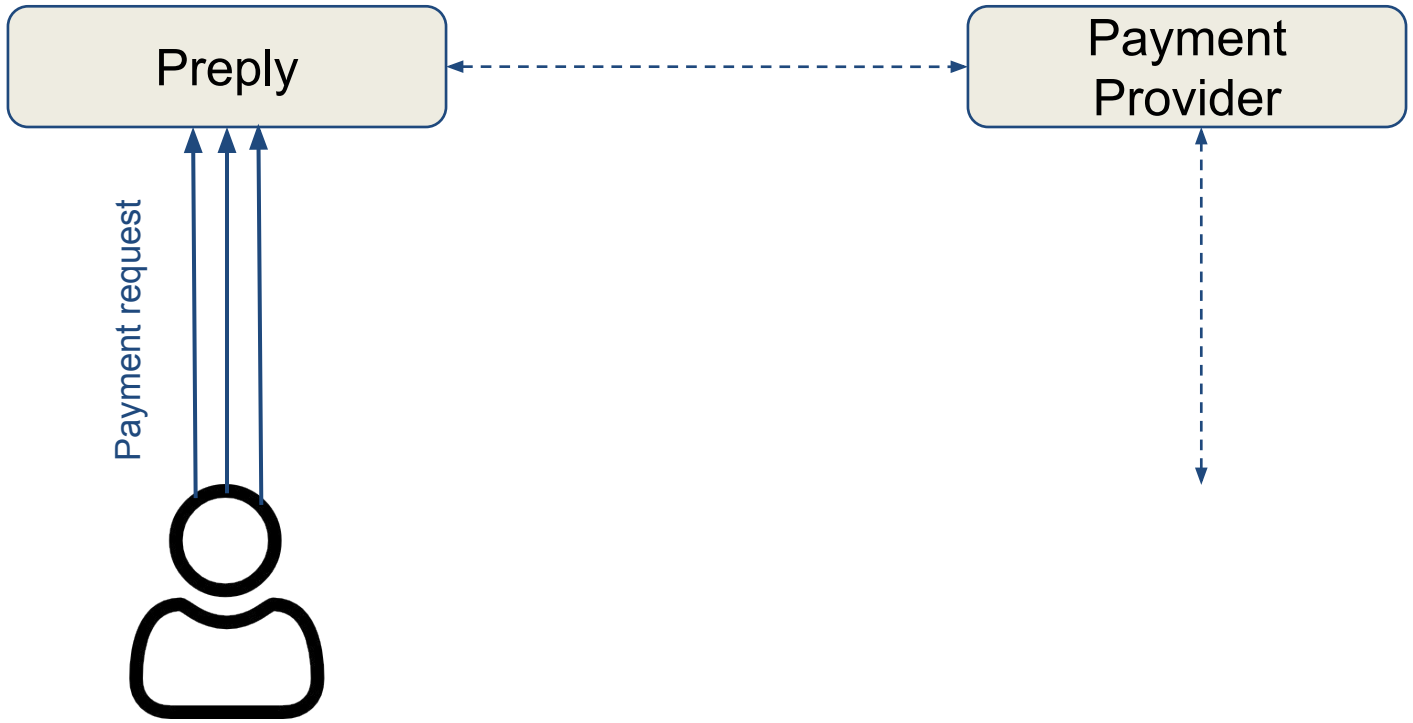
Preply

Payment flow



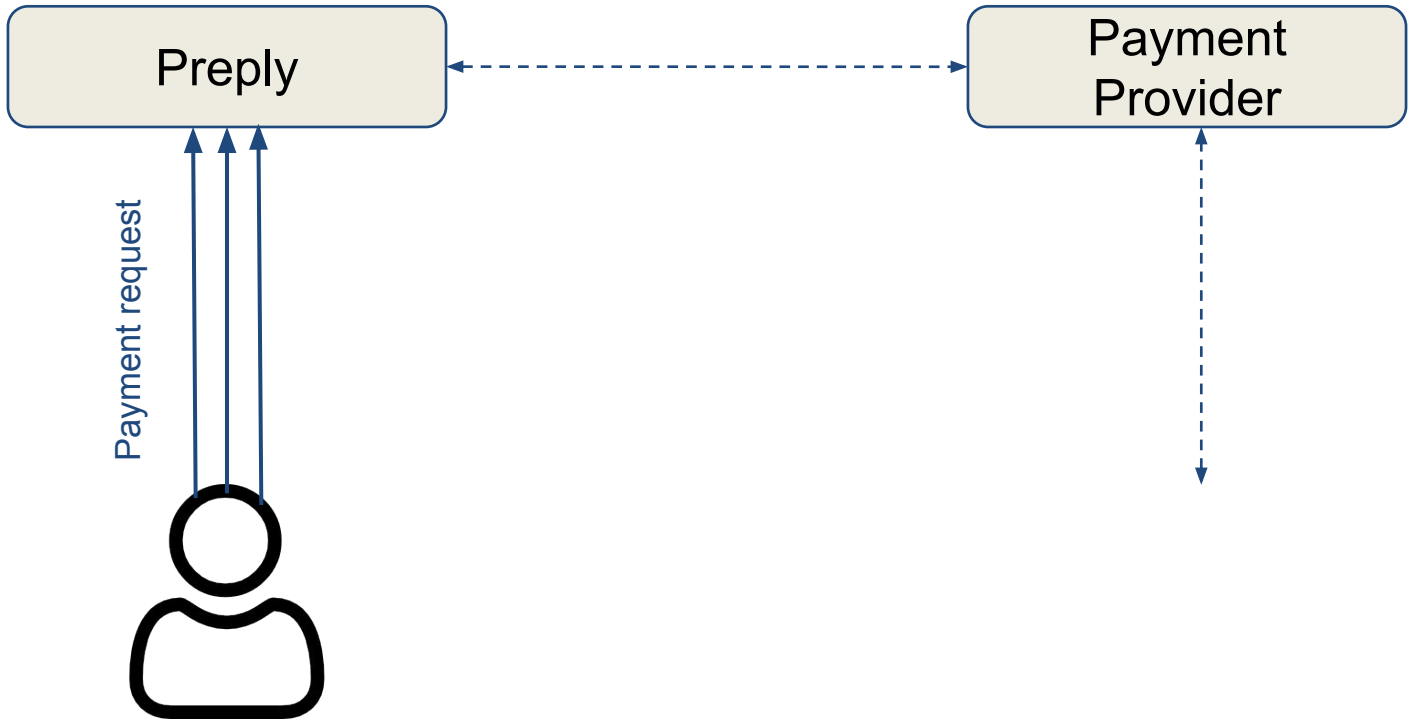
Preply

Payment flow



Preply

Payment flow



Preply

Reads, then writes - without protection

```
def add_funds(user_id, amount):
    wallet = Wallet.objects.get(user_id=user_id)
    wallet.balance += amount
    wallet.save()
```

Two requests, same wallet

- **Both read balance = 100**
- Both add 50
- Both write 150
- One deposit silently lost

Preply

Reads, then writes - without protection

```
def add_funds(user_id, amount):
    wallet = Wallet.objects.get(user_id=user_id)
    wallet.balance += amount
    wallet.save()
```

Two requests, same wallet

- Both read balance = 100
- **Both add 50**
- Both write 150
- One deposit silently lost

Preply

Reads, then writes - without protection

```
def add_funds(user_id, amount):
    wallet = Wallet.objects.get(user_id=user_id)
    wallet.balance += amount
    wallet.save()
```

Two requests, same wallet

- Both read balance = 100
- Both add 50
- **Both write 150**
- One deposit silently lost

Preply

Reads, then writes - without protection

```
def add_funds(user_id, amount):
    wallet = Wallet.objects.get(user_id=user_id)
    wallet.balance += amount
    wallet.save()
```

Two requests, same wallet

- Both read balance = 100
- Both add 50
- Both write 150
- **One deposit silently lost**

Preply

Lock the row, not the request

```
from django.db import transaction

with transaction.atomic():
    wallet = Wallet.objects
              .select_for_update()
              .get(user_id=user_id)
    wallet.balance += amount
    wallet.save()
```

- **Atomic boundary:** explicit, request-scoped, short-lived.
- **select_for_update:** row-level pessimistic lock at the DB.
- **Long locks =** outages. Keep transactions tiny.

Preply

Lock the row, not the request

```
from django.db import transaction

with transaction.atomic():
    wallet = Wallet.objects
              .select_for_update()
              .get(user_id=user_id)
    wallet.balance += amount
    wallet.save()
```

- **Atomic boundary:** explicit, request-scoped, short-lived.
- **select_for_update:** row-level pessimistic lock at the DB.
- **Long locks =** outages. Keep transactions tiny.

Preply

Lock the row, not the request

```
from django.db import transaction

with transaction.atomic():
    wallet = Wallet.objects
              .select_for_update()
              .get(user_id=user_id)
    wallet.balance += amount
    wallet.save()
```

- **Atomic boundary:** explicit, request-scoped, short-lived.
- **select_for_update:** row-level pessimistic lock at the DB.
- **Long locks =** outages. Keep transactions tiny.

Preply

02

Idempotency

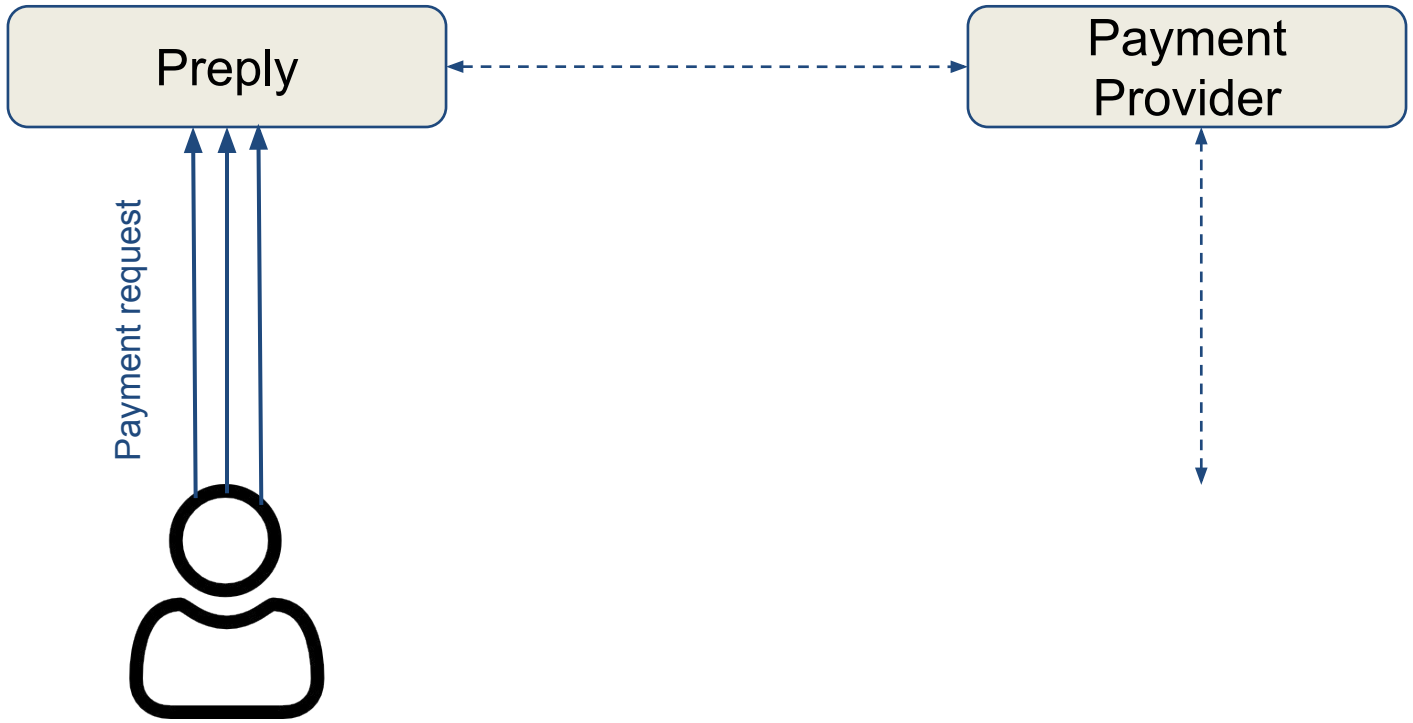
If two requests can race, they will.

Preply

Payment flow

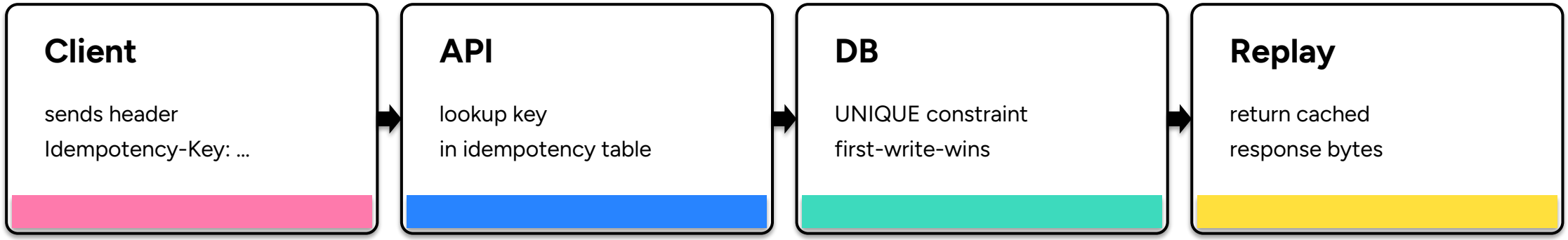
Preply

Payment flow



Preply

Idempotency keys - same key, same outcome



01 Key is the client's responsibility, scoped per endpoint.

02 First-write-wins via a DB UNIQUE constraint, not app logic.

03 Cache the response, not just the fact - replays must be byte-identical.

Preply

03

State machines

If two requests can race, they will.

Preply

State machines, not status strings

```
# Anywhere in the codebase:
payment.status = "captured"   # unenforced
payment.save()
```

- **Invalid transitions** raise loudly - at the source.
- **Audit trail** is a free side-effect.
- **Reviewers** can reason about flows in one place.

Preply

State machines, not status strings

```
# Anywhere in the codebase:
payment.status = "captured"   # unenforced
payment.clean()
payment.save()
```

- **Invalid transitions** raise loudly - at the source.
- **Audit trail** is a free side-effect.
- **Reviewers** can reason about flows in one place.

Preply

State machines, not status strings

```
# Anywhere in the codebase:
payment.status = "captured"   # unenforced
payment.save()

# Explicit transitions instead:
ALLOWED_STATUSES = {
  "authorized": {"captured", "voided"},
  "captured":   {"refunded"},
  "refunded":   set(),
  "voided":     set(),
}
def transition(payment, new_status):
    if new_status not in ALLOWED_STATUSES[payment.status]:
        raise InvalidTransition(payment.status, new_status)
    payment.status = new_status
    payment.save()
```

- **Invalid transitions** raise loudly - at the source.
- **Audit trail** is a free side-effect.
- **Reviewers** can reason about flows in one place.

Preply

State machines, not status strings

```
# Anywhere in the codebase:
payment.status = "captured"   # unenforced
payment.save()

# Explicit transitions instead:
ALLOWED_STATUSES = {
  "authorized": {"captured", "voided"},
  "captured":   {"refunded"},
  "refunded":   set(),
  "voided":     set(),
}
def transition(payment, new_status):
    if new_status not in ALLOWED_STATUSES[payment.status]:
        raise InvalidTransition(payment.status, new_status)
    payment.status = new_status
    payment.save()
```

- **Invalid transitions** raise loudly - at the source.
- **Audit trail** is a free side-effect.
- **Reviewers** can reason about flows in one place.

Preply

State machines, not status strings

```
# Anywhere in the codebase:
payment.status = "captured"   # unenforced
payment.save()

# Explicit transitions instead:
ALLOWED_STATUSES = {
  "authorized": {"captured", "voided"},
  "captured":   {"refunded"},
  "refunded":   set(),
  "voided":     set(),
}
def transition(payment, new_status):
    if new_status not in ALLOWED_STATUSES[payment.status]:
        raise InvalidTransition(payment.status, new_status)
    payment.status = new_status
    payment.save()
```

- **Invalid transitions** raise loudly - at the source.
- **Audit trail** is a free side-effect.
- **Reviewers** can reason about flows in one place.

Preply

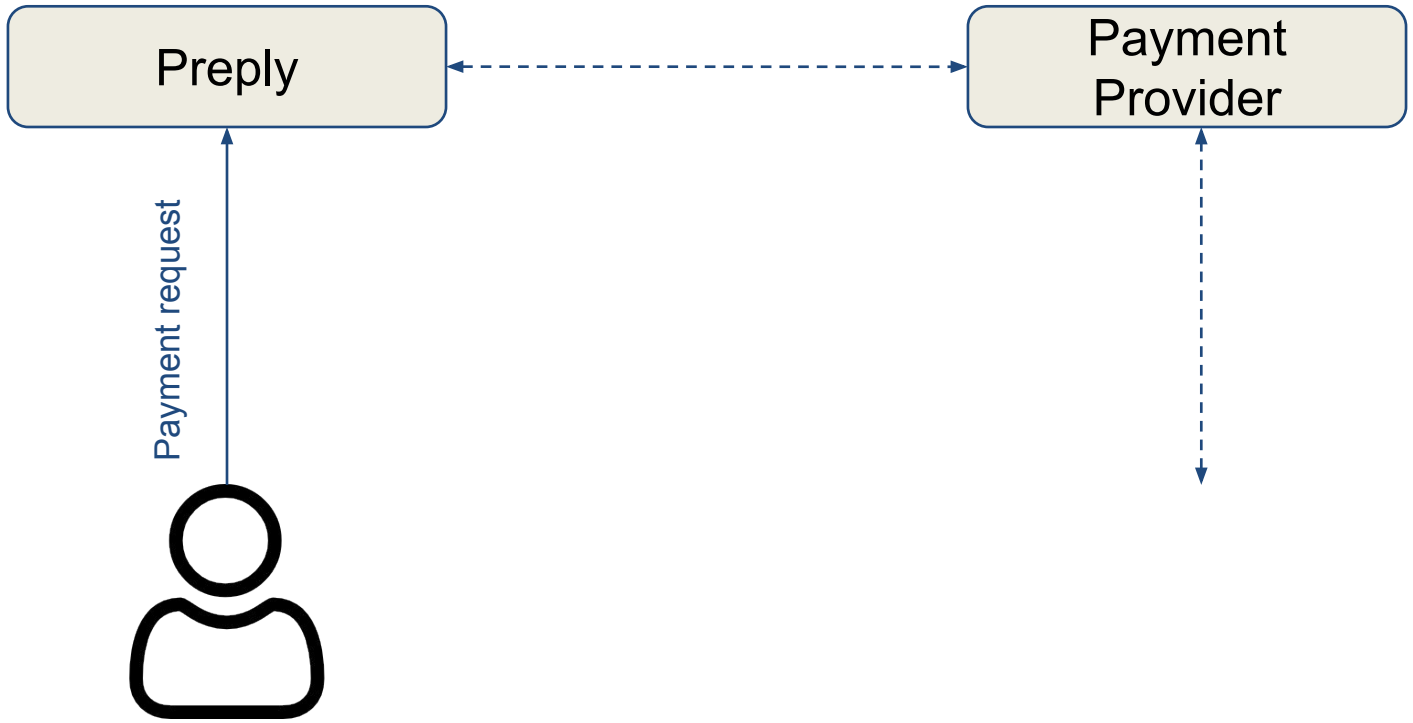
04

Abuse defenses

Anything you accept once, you'll be asked to accept a million times.

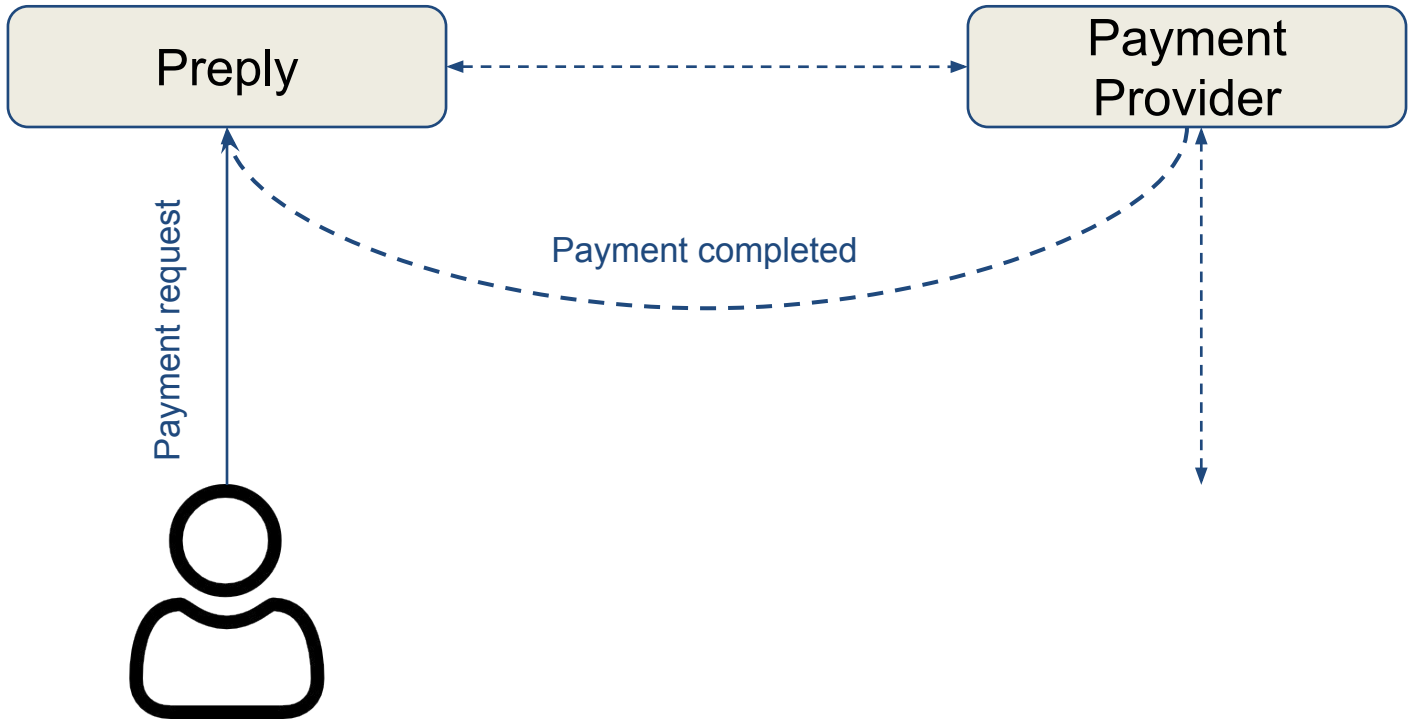
Preply

Payment flow



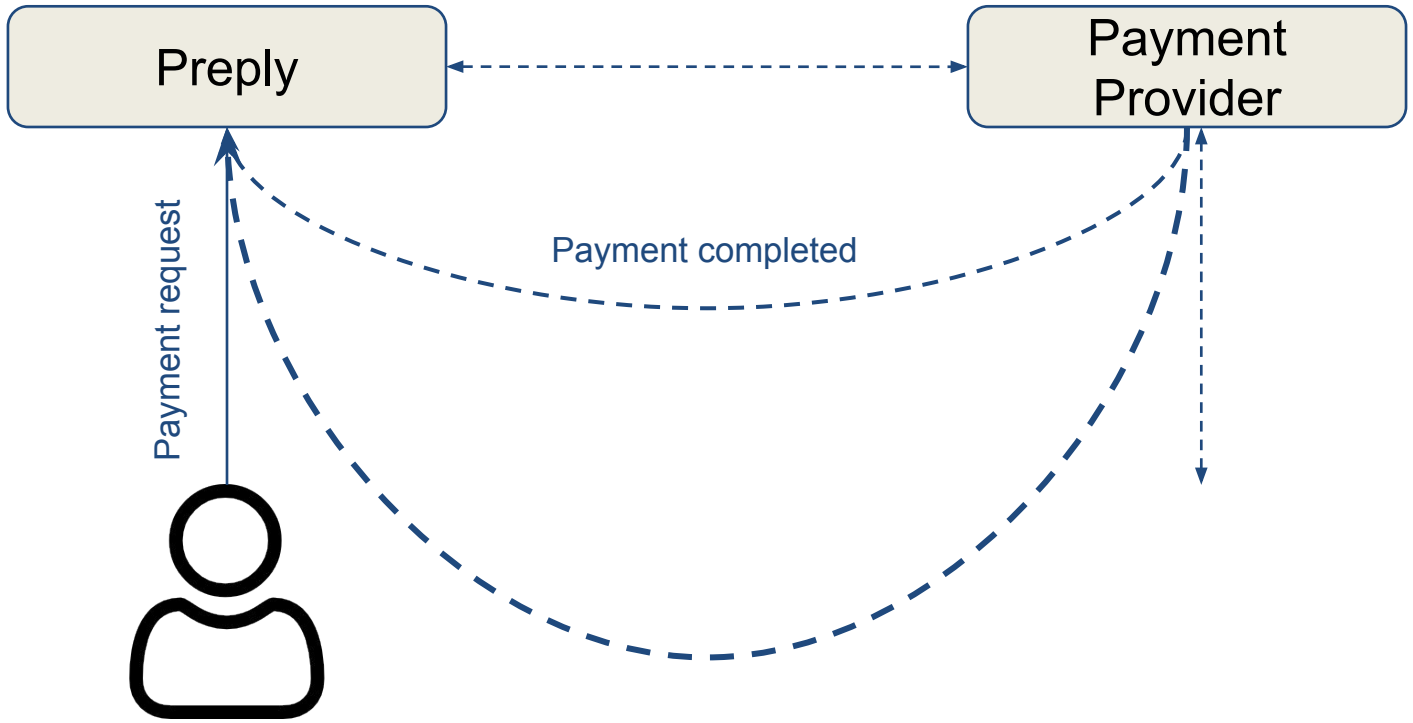
Preply

Payment flow



Preply

Payment flow



Preply

Webhook signatures, done right

```
import hmac, hashlib

def verify(raw_body, header_sig, secret, timestamp):
    if abs(now - timestamp) > MAX_SKEW_SECONDS:
        raise PermissionDenied("stale webhook")
    expected = hmac.new(
        secret, raw_body, hashlib.sha256
    ).hexdigest()
    if not hmac.compare_digest(expected, header_sig):
        raise PermissionDenied("bad signature")
```

- **Timestamp** - reject outside a ±5 min window.
- **Raw bytes** - sign-then-parse, not parse-then-sign.
- **compare_digest** - reduces timing attacks.
- **Key rotation** - accept N and N-1 during overlap.

Preply

Webhook signatures, done right

```
import hmac, hashlib

def verify(raw_body, header_sig, secret, timestamp):
    if abs(now - timestamp) > MAX_SKEW_SECONDS:
        raise PermissionDenied("stale webhook")
    expected = hmac.new(
        secret, raw_body, hashlib.sha256
    ).hexdigest()
    if not hmac.compare_digest(expected, header_sig):
        raise PermissionDenied("bad signature")
```

- **Timestamp** - reject outside a ±5 min window.
- **Raw bytes** - sign-then-parse, not parse-then-sign.
- **compare_digest** - reduces timing attacks.
- **Key rotation** - accept N and N-1 during overlap.

Preply

Webhook signatures, done right

```
import hmac, hashlib

def verify(raw_body, header_sig, secret, timestamp):
    if abs(now - timestamp) > MAX_SKEW_SECONDS:
        raise PermissionDenied("stale webhook")
    expected = hmac.new(
        secret, raw_body, hashlib.sha256
    ).hexdigest()
    if not hmac.compare_digest(expected, header_sig):
        raise PermissionDenied("bad signature")
```

- **Timestamp** - reject outside a ±5 min window.
- **Raw bytes** - sign-then-parse, not parse-then-sign.
- **compare_digest** - reduces timing attacks.
- **Key rotation** - accept N and N-1 during overlap.

Preply

Rate limiting and abuse signals

1	**In-process**	Django throttling: per user, per IP, per endpoint.
2	**At the edge**	WAF / API gateway rules for tracking attacks.
3	**Velocity**	Too many cards per user, users per card, failed auths.
4	**Signals**	Feed anomalies to a fraud system and don't block in the request path.

Preply

05

Django admin & internal tools

Your admin is a vault door with a 'push' sign on it.


MAKE STUDENTS HAPPY
Our mission: Every customer. Every time.
LIVE REQUESTS
1,248
-23% vs last hour
REQUESTS BY TYPE
Account 38%
Billing 28%
Technical 23%
Other 11%
AVERAGE RESPONSE TIME
2m 45s
-15% vs last hour
Great service builds strong connections.
LATEST TICKETS
#128793 I can't access my account 2m ago
#128792 Payment failed 3m ago
#128790 How to reset password? 5m ago
#128789 Error on checkout page 6m ago
#128799 Update billing info 7m ago
SERVICE STATUS
All Systems Operational
We're here to help.

Preply

What goes wrong with default admin

Direct model edits

Bypass your state machine

delete_selected on money

Almost never what you want on a payment, refund, or ledger entry

Support 'quick fixes'

Unauditable changes to ledger state

is_superuser = True

Accountability black hole

Preply

Make the admin boring on purpose

Read-only by default

Financial models read-only in admin

Preply

Make the admin boring on purpose

Four-eyes / dual control

Refunds above a threshold require a second approver

Preply

Make the admin boring on purpose

Action-level audit log

Who, when, why, before/after, ticket ID…

Preply

Make the admin boring on purpose

Permission groups

No superusers

Preply

Make the admin boring on purpose

Read-only by default

Financial models read-only in admin

Four-eyes / dual control

Refunds above a threshold require a second approver

Action-level audit log

Who, when, why, before/after, ticket ID...

Permission groups

No superusers.

Preply

06

Observability & incident response

You will be on call for this. Make 3 a.m. you grateful.

Preply

Log decisions, not payloads

```
logger.info(
    "payment.captured",
    extra={
        "payment_id": payment.id,
        "amount": payment.amount,
        "currency": payment.currency,
        "status": "captured",
        "user_id": user_id,
        "request_id": request_id,
    },
)
```

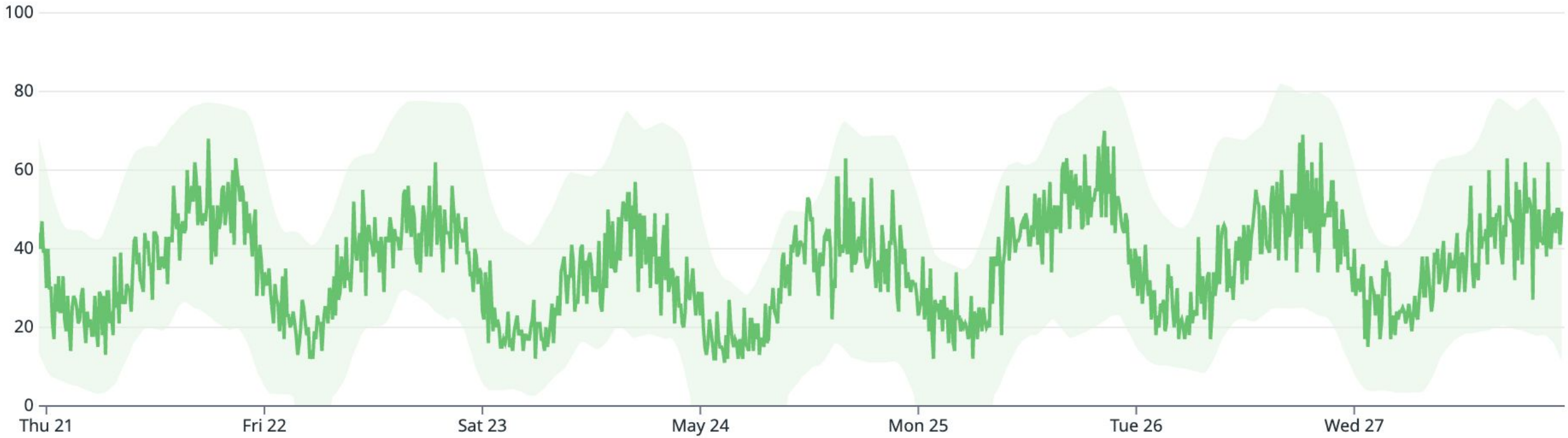
- **Structured logs** with a stable schema. extra={…}
- **Redact centrally** PII / PAN / tokens
- **Decision + hash** log what was decided plus a hash of the input
- **Correlate by both** request id and a business id

Alert on shape changes, not absolutes

- **Ratio over absolutes** — Compare ratio values: errors / (success + errors)

Alert on shape changes, not absolutes

- **Ratio over absolutes** — Compare ratio values: errors / (success + errors)
- **Failed verifications** — Investigate spikes in any segment


100
80
60
40
20
0
Thu 21
Fri 22
Sat 23
May 24
Mon 25
Tue 26
Wed 27

Preply

Alert on shape changes, not absolutes

- **Ratio over absolutes** — Compare ratio values: errors / (success + errors)
- **Failed verifications** — Investigate spikes in any segment
- **Any deviations** — Drift in either direction is a warning sign

Alert on shape changes, not absolutes

- **Ratio over absolutes** — Compare ratio values: errors / (success + errors)
- **Failed verifications** — Investigate spikes in any segment
- **Any deviations** — Drift in either direction is a warning sign
- **New everything** — First-seen card BIN, geolocation is an anomaly


WORK.
EXPLOIT.
REPEAT.
ACCESS GRANTED

Preply

Pre-merge checklist

Think like an attacker

1	What if this runs twice?
2	What if this runs simultaneously?
3	What if this request is replayed in 30 days?
4	What does the audit log look like? Who would notice if it didn't?
5	If I had your access for 5 minutes, what would I do?

Preply

Six things to take home

1

Lock the row,

not the request.

2

Idempotency is a feature,

not a workaround.

3

State machines beat

status strings.

4

Verify with defence layers

compare_digest, raw bytes, a clock.

5

Admin is production,

treat it like production.

6

Log decisions, observe anomalies

watch any deviations carefully.

Preply

Thank You

Build payments like the world is watching.

Dmytro Khmelenko

PyCon Italia 2026