# InsureVerifyAI CMS Setup

## Subscriptions (Monthly) — Database Schema & Flow

*(for Laravel + Inertia + Vue; you already have `users` and `plans`)*

This spec makes the subscription layer production-ready with monthly billing, usage metering, invoicing, payments, and auditability. It's intentionally Stripe/Cashier-friendly but provider-agnostic.

---

### Contents



---

### Assumptions

- **Billing period:** fixed **1 month** (rolling, aligned to `current_period_start` → `current_period_end`).
- **Actor: User** owns the subscription (`user_id`). If you later add organizations, add `organization_id` nullable and index it.
- **Plan:** a single plan per subscription (you already have `plans`).

- **Usage metric:** `verifications` (counts verification jobs started or completed, your choice—be consistent).
- **Payment provider:** Stripe or similar; fields labeled `provider_*` keep you decoupled.

---

## Data Model Overview

```bash
users 1—* subscriptions *—1 plans
subscriptions 1—* subscription_usages
subscriptions 1—* invoices 1—* invoice_items
invoices 1—* payments
subscriptions 1—* subscription_events
```

---

## Tables & Columns

### `subscriptions`

**One row per active or historical subscription.**

| Column | Type | Notes |
|---|---|---|
| `id` | bigIncrements | PK |
| `user_id` | foreignId | → `users.id`, indexed |
| `plan_id` | foreignId | → `plans.id`, indexed |
| `status` | enum | `trialing`, `active`, `past_due`, `paused`, `canceled`, `incomplete` |
| `trial_ends_at` | timestamp nullable | If you offer trials |
| `current_period_start` | timestamp | Inclusive start of current monthly cycle |
| `current_period_end` | timestamp | Exclusive end of cycle (renewal boundary) |

| Column | Type | Notes |
| --- | --- | --- |
| `renews_at` | timestamp nullable | Normally = `current_period_end` unless canceled |
| `canceled_at` | timestamp nullable | When user requested cancel |
| `cancel_at_period_end` | boolean default false | Classic “let it run out” toggle |
| `currency` | string(3) | e.g., `USD` |
| `price_monthly_cents` | integer | Snapshot of price at start of cycle |
| `included_verifications` | integer nullable | Snapshot of plan monthly quota |
| `overage_price_per_unit_cents` | integer nullable | If charging overages |
| `provider` | string nullable | e.g., `stripe` |
| `provider_customer_id` | string nullable | |
| `provider_subscription_id` | string nullable | |
| `metadata` | json nullable | Free-form |
| `created_at/updated_at` | timestamps | |

### Indexes

- `INDEX (user_id, status)`
- `INDEX (status, current_period_end)`
- `INDEX (provider, provider_subscription_id) UNIQUE`

---

## `subscription_usages`

**Rolls up metered counts per cycle and metric.**

| Column | Type | Notes |
|---|---|---|
| `id` | bigIncrements | PK |
| `subscription_id` | foreignId | → `subscriptions.id` |
| `metric` | string | e.g., `verifications` |
| `used` | integer unsigned | Count within the cycle |
| `period_start` | timestamp | Mirrors subscription period |
| `period_end` | timestamp | Mirrors subscription period |
| `last_incremented_at` | timestamp nullable | |
| `created_at/updated_at` | timestamps | |

### Uniqueness

- `UNIQUE (subscription_id, metric, period_start, period_end)`

---

## `invoices`

**One invoice per billing event (new cycle, proration, manual, dunning retry).**

| Column | Type | Notes |
|---|---|---|
| `id` | bigIncrements | PK |
| `subscription_id` | foreignId | → `subscriptions.id` |
| `user_id` | foreignId | Denormalized for convenience |

| Column | Type | Notes |
| --- | --- | --- |
| `number` | string | Human-friendly, unique: `INV-2025-000123` |
| `status` | enum | `draft`, `open`, `paid`, `void`, `uncollectible` |
| `currency` | string(3) | |
| `subtotal_cents` | integer | Sum of items pre-discount/tax |
| `discount_cents` | integer default 0 | |
| `tax_cents` | integer default 0 | |
| `total_cents` | integer | `subtotal - discount + tax` |
| `period_start` | timestamp | Invoice coverage start |
| `period_end` | timestamp | Invoice coverage end |
| `provider` | string nullable | |
| `provider_invoice_id` | string nullable | |
| `metadata` | json nullable | |
| `issued_at` | timestamp nullable | |
| `due_at` | timestamp nullable | |
| `paid_at` | timestamp nullable | |
| `created_at/updated_at` | timestamps | |

### Indexes

- `UNIQUE (number)`

- `INDEX (subscription_id, status)`

---

## `invoice_items`

**Line items for an invoice.**

| Column | Type | Notes |
|---|---|---|
| `id` | bigIncrements | PK |
| `invoice_id` | foreignId | → `invoices.id` |
| `type` | enum | `base_fee`, `overage`, `credit`, `adjustment`, `tax` |
| `description` | string | e.g., `Gold plan (Jul 24–Aug 24)` |
| `quantity` | integer | e.g., overage units |
| `unit_price_cents` | integer | Price per unit |
| `amount_cents` | integer | `quantity * unit_price_cents` or explicit |
| `metadata` | json nullable | |
| `created_at/updated_at` | timestamps | |

### Indexes

- `INDEX (invoice_id)`

---

## `payments`

**Payment attempts/records mapped to invoices.**

| Column | Type | Notes |
|---|---|---|
| `id` | bigIncrements | PK |

| Column | Type | Notes |
| --- | --- | --- |
| `invoice_id` | foreignId | → `invoices.id` |
| `provider` | string | `stripe`, `braintree`, etc. |
| `provider_payment_intent_id` | string nullable | |
| `status` | enum | `requires_action`, `processing`, `succeeded`, `failed`, `canceled` |
| `amount_cents` | integer | |
| `currency` | string(3) | |
| `error_code` | string nullable | Gateway error code |
| `error_message` | text nullable | |
| `paid_at` | timestamp nullable | When succeeded |
| `created_at/updated_at` | timestamps | |

### Indexes

- `INDEX (invoice_id, status)`

---

## `subscription_events`

**Audit trail for state changes and billable events.**

| Column | Type | Notes |
| --- | --- | --- |
| `id` | bigIncrements | PK |
| `subscription_id` | foreignId | → `subscriptions.id` |
| `actor_user_id` | foreignId nullable | Who triggered it (nullable for system/webhooks) |

| Column | Type | Notes |
| --- | --- | --- |
| `event` | string | e.g., `created`, `plan_changed`, `period_renewed`, `usage_incremented`, `invoice_generated`, `payment_failed`, `payment_succeeded`, `canceled`, `reactivated` |
| `old_values` | json nullable | |
| `new_values` | json nullable | |
| `metadata` | json nullable | |
| `created_at` | timestamp | |

### Indexes

- `INDEX (subscription_id, event, created_at)`

---

## Migrations (Laravel examples)

> Replace `enum` with `string` + check constraints if your DB doesn't support native enums.

```php
// 2025_08_24_000001_create_subscriptions_table.php
Schema::create('subscriptions', function (Blueprint $t) {
    $t->id();
    $t->foreignId('user_id')->constrained()->cascadeOnDelete();
    $t->foreignId('plan_id')->constrained()->restrictOnDelete();
    $t->enum('status', ['trialing','active','past_due','paused','canceled','incomplete'])->index();
    $t->timestamp('trial_ends_at')->nullable();

    $t->timestamp('current_period_start');
    $t->timestamp('current_period_end');
    $t->timestamp('renews_at')->nullable();
    $t->timestamp('canceled_at')->nullable();
    $t->boolean('cancel_at_period_end')->default(false);

    $t->string('currency', 3)->default('USD');
    $t->integer('price_monthly_cents');          // snapshot
```

```
    $t->integer('included_verifications')->nullable();// snapshot
    $t->integer('overage_price_per_unit_cents')->nullable();

    $t->string('provider')->nullable();
    $t->string('provider_customer_id')->nullable();
    $t->string('provider_subscription_id')->nullable()->unique();
    $t->json('metadata')->nullable();

    $t->timestamps();

    $t->index(['user_id', 'status']);
    $t->index(['status', 'current_period_end']);
});
```

```
php

// 2025_08_24_000002_create_subscription_usages_table.php
Schema::create('subscription_usages', function (Blueprint $t) {
    $t->id();
    $t->foreignId('subscription_id')->constrained()->cascadeOnDelete();
    $t->string('metric'); // 'verifications'
    $t->unsignedInteger('used')->default(0);
    $t->timestamp('period_start');
    $t->timestamp('period_end');
    $t->timestamp('last_incremented_at')->nullable();
    $t->timestamps();

    $t->unique(['subscription_id','metric','period_start','period_end'],
'subscription_usage_unique');
});
```

```
php

// 2025_08_24_000003_create_invoices_table.php
Schema::create('invoices', function (Blueprint $t) {
    $t->id();
    $t->foreignId('subscription_id')->constrained()->cascadeOnDelete();
    $t->foreignId('user_id')->constrained()->cascadeOnDelete();
    $t->string('number')->unique(); // e.g., INV-2025-000123
    $t->enum('status', ['draft','open','paid','void','uncollectible'])->default('draft')->index();
    $t->string('currency', 3)->default('USD');
```

```
    $t->integer('subtotal_cents');
    $t->integer('discount_cents')->default(0);
    $t->integer('tax_cents')->default(0);
    $t->integer('total_cents');

    $t->timestamp('period_start');
    $t->timestamp('period_end');

    $t->string('provider')->nullable();
    $t->string('provider_invoice_id')->nullable();
    $t->json('metadata')->nullable();

    $t->timestamp('issued_at')->nullable();
    $t->timestamp('due_at')->nullable();
    $t->timestamp('paid_at')->nullable();

    $t->timestamps();

    $t->index(['subscription_id','status']);
});
```

php

```
// 2025_08_24_000004_create_invoice_items_table.php
Schema::create('invoice_items', function (Blueprint $t) {
    $t->id();
    $t->foreignId('invoice_id')->constrained()->cascadeOnDelete();
    $t->enum('type', ['base_fee','overage','credit','adjustment','tax']);
    $t->string('description');
    $t->integer('quantity')->default(1);
    $t->integer('unit_price_cents')->default(0);
    $t->integer('amount_cents'); // if provided, authoritative
    $t->json('metadata')->nullable();
    $t->timestamps();

    $t->index('invoice_id');
});
```

php

```
// 2025_08_24_000005_create_payments_table.php
Schema::create('payments', function (Blueprint $t) {
```

```
    $t->id();
    $t->foreignId('invoice_id')->constrained()->cascadeOnDelete();
    $t->string('provider');
    $t->string('provider_payment_intent_id')->nullable()->index();
    $t->enum('status', ['requires_action','processing','succeeded','failed','canceled'])->index();
    $t->integer('amount_cents');
    $t->string('currency', 3)->default('USD');
    $t->string('error_code')->nullable();
    $t->text('error_message')->nullable();
    $t->timestamp('paid_at')->nullable();
    $t->timestamps();
});
```

```
php

// 2025_08_24_000006_create_subscription_events_table.php
Schema::create('subscription_events', function (Blueprint $t) {
    $t->id();
    $t->foreignId('subscription_id')->constrained()->cascadeOnDelete();
    $t->foreignId('actor_user_id')->nullable()->constrained('users')->nullOnDelete();
    $t->string('event'); // created, plan_changed, period_renewed, usage_incremented,
invoice_generated, payment_failed, payment_succeeded, canceled, reactivated
    $t->json('old_values')->nullable();
    $t->json('new_values')->nullable();
    $t->json('metadata')->nullable();
    $t->timestamp('created_at')->useCurrent();

    $t->index(['subscription_id','event','created_at'], 'subscription_events_idx');
});
```

---

## Relationships (Eloquent)

```
php

class Subscription extends Model {
    protected $casts = [
        'metadata' => 'array',
        'cancel_at_period_end' => 'boolean',
        'current_period_start' => 'datetime',
        'current_period_end'   => 'datetime',
```

```
        'renews_at'          => 'datetime',
        'canceled_at'        => 'datetime',
        'trial_ends_at'      => 'datetime',
    ];

    public function user()  { return $this->belongsTo(User::class); }
    public function plan()  { return $this->belongsTo(Plan::class); }
    public function usages(){ return $this->hasMany(SubscriptionUsage::class); }
    public function invoices(){ return $this->hasMany(Invoice::class); }
    public function events(){ return $this->hasMany(SubscriptionEvent::class); }
}

class SubscriptionUsage extends Model {
    protected $casts = ['last_incremented_at' => 'datetime', 'period_start'=>'datetime',
'period_end'=>'datetime'];
    public function subscription(){ return $this->belongsTo(Subscription::class); }
}

class Invoice extends Model {
    protected $casts =
['issued_at'=>'datetime','due_at'=>'datetime','paid_at'=>'datetime','period_start'=>'datetime','per
    public function subscription(){ return $this->belongsTo(Subscription::class); }
    public function user(){ return $this->belongsTo(User::class); }
    public function items(){ return $this->hasMany(InvoiceItem::class); }
    public function payments(){ return $this->hasMany(Payment::class); }
}

class InvoiceItem extends Model {
    public function invoice(){ return $this->belongsTo(Invoice::class); }
}

class Payment extends Model {
    protected $casts = ['paid_at'=>'datetime'];
    public function invoice(){ return $this->belongsTo(Invoice::class); }
}

class SubscriptionEvent extends Model {
    public $timestamps = false; // has created_at only
    protected $casts =
['old_values'=>'array','new_values'=>'array','metadata'=>'array','created_at'=>'datetime'];
    public function subscription(){ return $this->belongsTo(Subscription::class); }
```

```
    public function actor(){ return $this->belongsTo(User::class, 'actor_user_id'); }
}
```

---

## Lifecycle & “How it works”

1. **Create subscription**
   - On checkout, create `subscriptions` with `status='active'` (or `trialing`).
   - Set `current_period_start = now()`, `current_period_end = now()->addMonth()`, `renews_at = current_period_end`.
   - Copy plan snapshots (`price_monthly_cents`, `included_verifications`, optional `overage_price_per_unit_cents`).
   - Insert `subscription_usages` row for `metric='verifications'` for the current period with `used=0`.
   - Log `subscription_events.created`.
2. **Meter usage**
   - Each time a verification is started (or completed), **increment** the `subscription_usages.used`.
   - If `used >= included_verifications`, either **block** new verifications or allow **overage**.
   - Log `subscription_events.usage_incremented`.
3. **Renewal (monthly cron/queue)**
   - On `current_period_end`, generate an **invoice** for base fee + any **overages**:
     - Add `invoice_items`: `base_fee` = `price_monthly_cents`, `overage` = `max(used - included, 0) * overage_price_per_unit_cents`.
     - Mark invoice `open`, attempt **payment** via provider; update `payments` and invoice `status`.
     - If payment fails: set subscription `status='past_due'` and start dunning.
   - **Advance period**: set `current_period_start = current_period_end`, `current_period_end = current_period_start->addMonth()`, refresh `renews_at`.
   - **Reset usage**: create new `subscription_usages` row with `used=0` for the new period.
   - Log `subscription_events.period_renewed` and `invoice_generated`.
4. **Cancellation**

- If `cancel_at_period_end = true`, keep running until `current_period_end`, then mark `status='canceled'`, set `canceled_at`.
- If immediate cancel: set `status='canceled'`, disable new usage, pro-rate refund if needed (create credit `invoice_items`).
- Log `subscription_events.canceled`.

5. **Plan change (upgrade/downgrade)**
   - Write `subscription_events.plan_changed` with `old_values/new_values`.
   - Choose your approach:
     - **Proration mid-cycle** (create interim invoice for difference)
     - **Defer change** to next cycle (simpler)

---

## Usage & Enforcement Examples

**Increment usage safely (single SQL):**

```php
DB::transaction(function () use ($subscription) {
    $usage = SubscriptionUsage::lockForUpdate()->firstOrCreate([
        'subscription_id' => $subscription->id,
        'metric' => 'verifications',
        'period_start' => $subscription->current_period_start,
        'period_end' => $subscription->current_period_end,
    ], ['used' => 0]);

    $usage->increment('used');
    $usage->last_incremented_at = now();
    $usage->save();

    // Enforce limits
    $limit = $subscription->included_verifications;
    if (!is_null($limit) && $usage->used > $limit) {
        if (is_null($subscription->overage_price_per_unit_cents)) {
            throw new \DomainException('Verification quota exceeded for this period.');
        }
        // else: allowed; will bill as overage on next invoice
    }

    SubscriptionEvent::create([
        'subscription_id' => $subscription->id,
```

```
        'event' => 'usage_incremented',
        'new_values' => ['metric' => 'verifications', 'used' => $usage->used],
        'created_at' => now(),
    ]);
});
```

**Generate a monthly invoice:**

```
php

$usage = $subscription->usages()
    ->where('metric','verifications')
    ->where('period_start', $subscription->current_period_start)
    ->where('period_end', $subscription->current_period_end)
    ->first();

$over = max(($usage->used ?? 0) - ($subscription->included_verifications ?? 0), 0);
$overCents = $over * (int) ($subscription->overage_price_per_unit_cents ?? 0);

$invoice = Invoice::create([
    'subscription_id' => $subscription->id,
    'user_id' => $subscription->user_id,
    'number' => generateInvoiceNumber(), // your helper
    'status' => 'open',
    'currency' => $subscription->currency,
    'subtotal_cents' => 0,
    'discount_cents' => 0,
    'tax_cents' => 0,
    'total_cents' => 0,
    'period_start' => $subscription->current_period_start,
    'period_end' => $subscription->current_period_end,
    'issued_at' => now(),
]);

$items = [
    [
        'type'=>'base_fee',
        'description'=> "{$subscription->plan->name} plan ({$invoice->period_start->toDateString()}—{$invoice->period_end->toDateString()})",
        'quantity'=>1,
        'unit_price_cents'=>$subscription->price_monthly_cents,
        'amount_cents'=>$subscription->price_monthly_cents,
    ],
```

```
];

if ($over > 0 && $overCents > 0) {
    $items[] = [
        'type'=>'overage',
        'description'=> "Overage: {$over} verification(s)",
        'quantity'=>$over,
        'unit_price_cents'=>$subscription->overage_price_per_unit_cents,
        'amount_cents'=>$overCents,
    ];
}

$total = collect($items)->sum('amount_cents');

foreach ($items as $i) {
    $invoice->items()->create($i);
}

$invoice->update(['subtotal_cents'=>$total, 'total_cents'=>$total]);
```

---

## Operational Notes & Indexing

- **Clock drift:** Use a single trusted source of time (DB/server) for period boundaries.
- **Idempotency:** For renewal/invoicing jobs, use idempotency keys or lock the subscription row.
- **Snapshots:** Keep price & limits on the subscription row to avoid retroactive billing changes if the plan table is edited.
- **Dunning:** If a payment fails, create events, move `status` → `past_due`, retry payments on a backoff schedule; auto-cancel after N days if needed.
- **Indices:**
  - Subscriptions: `(user_id, status)`, `(status, current_period_end)`
  - Usages: unique composite on `(subscription_id, metric, period_start, period_end)`
  - Invoices: `number` unique, `(subscription_id, status)`
  - Payments: `(invoice_id, status)`
  - Events: `(subscription_id, event, created_at)`

---

**That’s the full subscription layer**—clean monthly cycles, usage, invoicing, payments, and auditable events. If you want, I can also add **policies/guards** for who can see invoices, and a **simple Inertia admin** to view subscription status, usage, invoices, and payments.