🎯 PROMPT COMPLET - DÉVELOPPEMENT SECTION ADMIN OLIVER PLATFORM

📋 CONTEXTE DU PROJET

Tu vas développer **l'intégralité de la section Admin** de la plateforme Oliver (clone OnlyFans professionnel) en te basant sur:

- ✅ 9 interfaces visuelles fournies (référence design exacte)
- ✅ PDF de spécifications techniques complètes
- ✅ Architecture monorepo Turborepo existante dans `C:\dev`

IMPÉRATIF: Code **COMPLET et PRODUCTION-READY**, pas d'exemples ou de placeholders.

📁 ANALYSE DU CODE EXISTANT

Structure du Projet

```
C:\dev/
├── apps/
│   ├── web/          # Next.js 15 - Frontend
│   └── api/          # NestJS - Backend
├── packages/
│   ├── database/     # Prisma schema partagé
│   ├── shared/       # Types et utilitaires
│   └── config/       # Configuration partagée
├── infra/            # Infrastructure (Docker, K8s)
├── .github/          # CI/CD workflows
└── docker-compose.yml
```

Éléments à Réutiliser

1. Design System Existant

Localisation: `apps/web/components/ui/` et `apps/web/styles/globals.css`

Palette de couleurs à utiliser:

```css
--primary: #00B8A9       /* Turquoise principal */
--primary-hover: #00A395  /* Hover state */
--success: #22C55E
--warning: #F59E0B
--error: #EF4444
--background-main: #FFFFFF
--background-secondary: #F8FAFB
--text-primary: #1A1A1A
--text-secondary: #6B7280
--border: #E5E7EB
```

Components UI à réutiliser:

- Buttons (primary, secondary, ghost)
- Cards avec border-radius: 16px
- Input fields avec validation
- Modals/Dialogs
- Toasts pour notifications
- Tables avec tri et filtres
- Charts (Recharts configuré)

2. Architecture Backend

Localisation: `apps/api/src/`

Modules existants à étendre:

- `auth/` - Système d'authentification JWT
- `users/` - Gestion utilisateurs (à étendre avec admin features)
- `payments/` - Intégration paiements (à connecter avec accounting)
- `common/guards/` - Guards d'authentification (créer RoleGuard pour admin)

Configurations existantes:

- Database connection (Prisma)
- Redis cache
- File upload (local, S3 ready)
- Email service (SMTP configuré)

3. Database Schema

Localisation: `packages/database/prisma/schema.prisma`

Modèles existants:

- User (avec role: FAN | CREATOR | ADMIN | MODERATOR)
- CreatorProfile
- FanProfile
- Post
- Subscription
- Message
- Payment
- Like
- Purchase

À AJOUTER pour Admin (section complète plus bas)

🎨 SECTION 1: INTERFACES FRONTEND (9 ÉCRANS)

Contraintes de Design

- **Framework**: Next.js 15 avec App Router
- **Styling**: Tailwind CSS (classes utilitaires seulement)
- **Components**: Shadcn/ui + Radix UI
- **Charts**: Recharts pour tous les graphiques
- **Icons**: Lucide React
- **State**: Zustand pour state global, React Query pour data fetching
- **Forms**: React Hook Form + Zod validation

Design Principles STRICTS

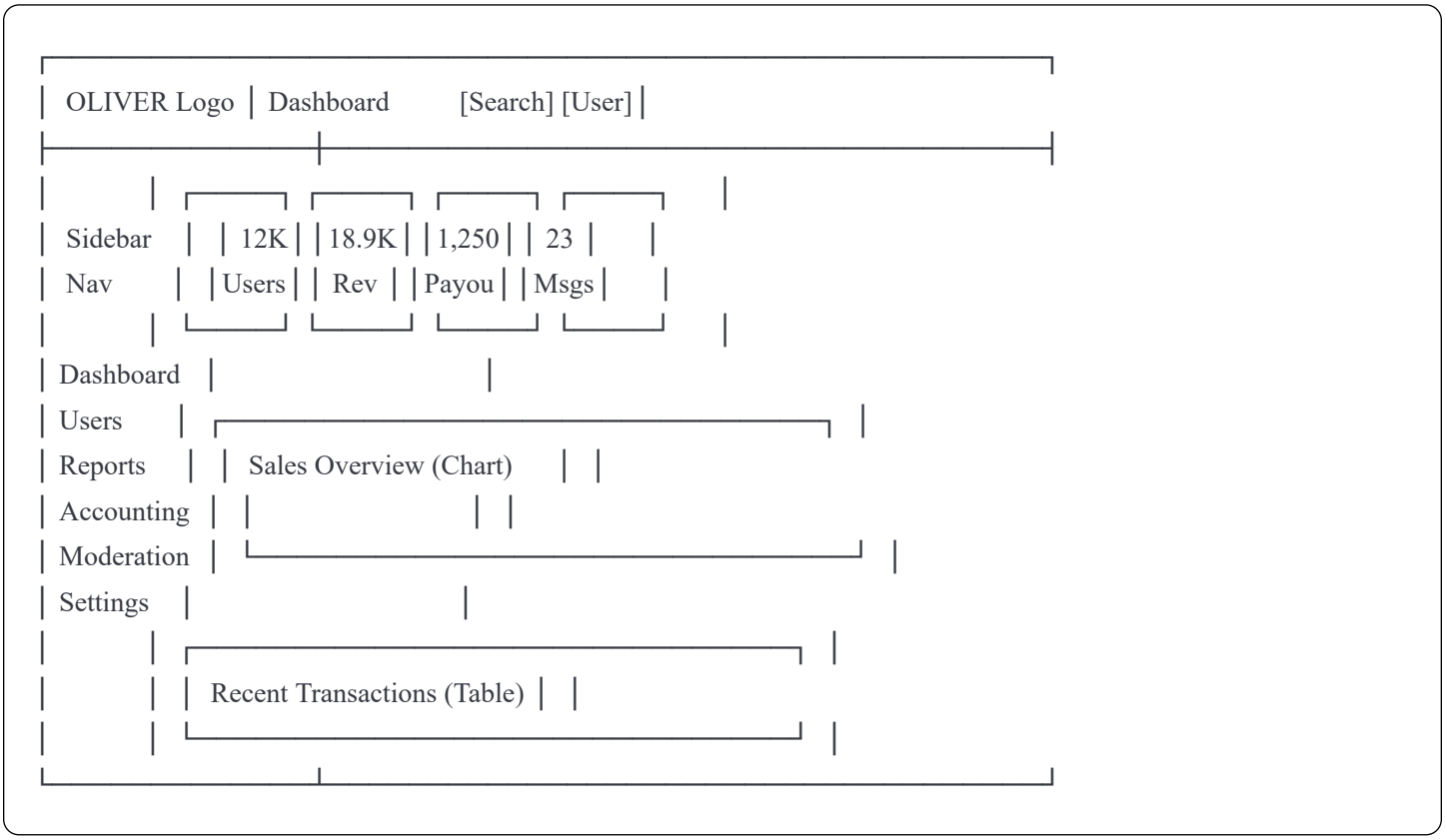
typescript

```typescript
// Spacing system (utiliser uniquement ces valeurs)
const spacing = {
  xs: '4px',
  sm: '8px',
  md: '16px',
  lg: '24px',
  xl: '32px',
  xxl: '48px'
}

// Border radius
const borderRadius = {
  button: '10px',
  card: '16px',
  input: '8px'
}

// Typography
const typography = {
  fontFamily: '-apple-system, Inter, system-ui',
  heading: {
    h1: { size: '32px', weight: 600 },
    h2: { size: '24px', weight: 600 },
    h3: { size: '20px', weight: 500 },
    h4: { size: '16px', weight: 500 }
  },
  body: {
    regular: { size: '14px', lineHeight: 1.5 },
    small: { size: '13px', lineHeight: 1.4 },
    tiny: { size: '12px', lineHeight: 1.3 }
  }
}

// Shadows (subtle uniquement)
const shadows = {
  card: '0 1px 3px rgba(0,0,0,0.05)',
  hover: '0 4px 6px rgba(0,0,0,0.07)',
  modal: '0 10px 25px rgba(0,0,0,0.15)'
}
```

1.1 Dashboard Admin (`/admin/dashboard`)

Référence Visuelle: Image 1

Layout:



Fichier: `apps/web/app/(admin)/dashboard/page.tsx`

Features à implémenter:

1. **Metric Cards** (4 cards horizontales)
   - Total Users: Nombre + % de changement (vert si positif, rouge si négatif)
   - Revenue: Montant en EUR + % changement
   - Pending Payouts: Nombre + % changement
   - Messages: Nombre non lus (badge rouge si > 0)
   - Source données: API `/api/admin/dashboard/metrics`
   - Refresh: toutes les 30 secondes (React Query)
   - Animation: Counter animé sur changement de valeur
2. **Sales Overview Chart**
   - Type: Area Chart (Recharts)
   - Données: Revenus quotidiens sur période sélectionnée
   - Période selector: [7 jours, 30 jours, 3 mois, 1 an]
   - Couleur: gradient turquoise (`#00B8A9` to transparent)
   - Axes: X = dates, Y = montant en K€
   - Tooltip: Date + montant exact
   - Responsive: adaptatif mobile
3. **Recent Transactions Table**
   - Colonnes: Date | Description | Amount
   - 5 dernières transactions
   - Amount: vert si positif, rouge si négatif
   - Action: Clic sur ligne → modal détails transaction
   - Button "View Reports" → redirect vers `/admin/accounting`

API Endpoints à créer:

typescript

```
GET /api/admin/dashboard/metrics
Response: {
  totalUsers: number
  totalUsersChange: number  // %
  revenue: number           // en centimes
  revenueChange: number     // %
  pendingPayouts: number
  pendingPayoutsChange: number
  unreadMessages: number
}

GET /api/admin/dashboard/sales?period=30d
Response: {
  date: string          // ISO format
  value: number         // en centimes
}[]

GET /api/admin/dashboard/transactions?limit=5
Response: {
  id: string
  date: string
  description: string
  amount: number        // en centimes (négatif = dépense)
  type: 'CREDIT' | 'DEBIT'
}[]
```

State Management:

typescript

```
// Zustand store pour admin dashboard
interface AdminDashboardStore {
  metrics: Metrics | null
  salesData: SalesData[]
  transactions: Transaction[]
  selectedPeriod: '7d' | '30d' | '3m' | '1y'
  setSelectedPeriod: (period: string) => void
  refreshMetrics: () => Promise<void>
}
```

Validations & Error Handling:

- Si API down: afficher skeleton loaders
- Si données incohérentes: logger erreur + fallback à 0
- Si timeout (>5s): afficher message retry
- Toasts pour actions (succès/erreur)

1.2 Users Management (`/admin/users`)

Référence Visuelle: Image 2

Features à implémenter:

1. **Filtres & Recherche**
   - Tabs: All Users | Admins | Creators | Pending KYCs | Suspended
   - Search bar: recherche par nom, email, username (debounce 300ms)
   - Button "Add New User" → Modal création
2. **Bulk Actions**
   - Checkboxes sur chaque ligne
   - Actions: "Suspend Selected" | "Reset Password" | "Export CSV"
   - Confirmation modal avant action bulk
   - Progress indicator pour actions longues
3. **Table Utilisateurs**
   - Colonnes: [Checkbox] | Avatar+Name | Email | Role (badge coloré) | Account Age | Last Login | Actions
   - Tri: toutes colonnes cliquables
   - Pagination: 50 users par page
   - Actions par ligne: Suspend | Reset | View (→ modal détails)
4. **User Detail Modal**
   - Infos complètes user
   - Historique transactions
   - Posts publiés
   - Warnings/bans
   - Boutons: Edit Profile | Suspend | Delete (avec confirmation)

API Endpoints:

```typescript
GET /api/admin/users?role=CREATOR&status=VERIFIED&search=john&page=1&limit=50&sortBy=createdAt&sortOrder
Response: {
  users: User[]
  total: number
  page: number
  totalPages: number
}

POST /api/admin/users/:id/suspend
Body: { reason: string, duration?: number }
Response: { success: boolean, user: User }

POST /api/admin/users/bulk-suspend
Body: { userIds: string[], reason: string }
Response: { success: boolean, suspended: number }

POST /api/admin/users/:id/reset-password
Response: { success: boolean, temporaryPassword: string }

GET /api/admin/users/export.csv?filters=...
Response: CSV file download
```

Permissions Check:

```typescript
// Middleware pour vérifier role admin
if (user.role !== 'ADMIN' && user.role !== 'MODERATOR') {
  throw new UnauthorizedException()
}
```

1.3 Reports & Flags (`/admin/reports`)

Référence Visuelle: Image 3

Features à implémenter:

1. **Filtres**
   - Dropdowns: Type | Severity | Status
   - Button "Filter" + "Clear"
   - Compteur de résultats
2. **Reports List**
   - Colonnes: Report ID | User (avatar) | Target (badge coloré) | Timestamp
   - Click sur ligne → Panel détails à droite
   - Badge sévérité (LOW=vert, MEDIUM=orange, HIGH=rouge)
3. **Report Detail Panel**
   - Content: Preview du contenu rapporté
   - Tabs: Content | Context | High (historique)
   - Reporter info
   - Timeline des actions
   - Actions: Dismiss | Warn | Ban | Approve

API Endpoints:

```typescript
GET /api/admin/reports?status=PENDING&severity=HIGH&type=CONTENT&page=1
Response: {
  reports: Report[]
  total: number
}

PATCH /api/admin/reports/:id
Body: {
  status: 'DISMISSED' | 'RESOLVED' | 'ESCALATED',
  action: 'WARN' | 'BAN' | 'DELETE_CONTENT',
  notes: string
}
Response: { success: boolean, report: Report }

POST /api/admin/reports/:id/assign
Body: { moderatorId: string }
Response: { success: boolean }
```

Real-time Updates:

```typescript
// WebSocket pour nouveaux reports
socket.on('admin:new-report', (report) => {
  // Toast notification
  // Incrémenter compteur
  // Ajouter à la liste si filtres matchent
})
```

1.4 Moderation Queue (`/admin/moderation`)

Référence Visuelle: Image 4

Features CRITIQUES:

1. **Queue List (gauche)**
   - Posts en attente de modération
   - Avatar créateur + username
   - Badge priorité (LOW, MEDIUM, HIGH)
   - Timestamp relatif (2h ago)
   - Click → charge détails à droite
2. **Content Review (droite)**
   - **IMPORTANT**: Image/Video avec blur par défaut
   - Bouton "Unblur to review" (protection moderator)
   - Infos: Creator, Reported by, Reports count
   - **AI Analysis** (CRUCIAL):
     - Violence: 12% (barre turquoise)
     - Adult: 2% (barre turquoise)
     - Hate: 56% (barre turquoise)
     - Spam: 8% (barre turquoise)
   - Note: "AI risk analysis generated automatically for review assistance"
3. **Actions**
   - Buttons: Approve (vert) | Reject (rouge) | Request (orange)
   - Shortcuts clavier: A=Approve, R=Reject, E=Escalate
   - Confirmation avant action irréversible
4. **Moderator Protection**
   - Timer: Maximum 4h de modération continue
   - Breaks obligatoires: 15min toutes les heures
   - Compteur de contenus reviewés
   - Alert si seuil atteint

API Endpoints:

```typescript
GET /api/admin/moderation/queue?priority=HIGH&status=PENDING
Response: {
  items: {
    id: string
    contentId: string
    contentType: 'POST' | 'MESSAGE' | 'PROFILE'
    creatorId: string
    creatorName: string
    creatorAvatar: string
    reportCount: number
    priority: 'LOW' | 'MEDIUM' | 'HIGH'
    aiAnalysis: {
      violence: number
      adult: number
      hate: number
      spam: number
    }
    createdAt: string
  }[]
}

POST /api/admin/moderation/decision
Body: {
  contentId: string
  decision: 'APPROVE' | 'REJECT' | 'ESCALATE'
  reason: string
  actions?: ('DELETE_CONTENT' | 'WARN_USER' | 'BAN_USER')[]
}
Response: { success: boolean }
```

AI Integration:

```typescript
// Service pour analyse AI (Google Vision + AWS Rekognition)
async analyzeContent(contentUrl: string) {
  const results = await Promise.all([
    googleVision.analyze(contentUrl),
    awsRekognition.analyze(contentUrl),
    customModel.analyze(contentUrl)
  ])

  return {
    violence: average([results[0].violence, results[1].violence]),
    adult: average([results[0].adult, results[1].adult]),
    hate: results[2].hate, // Custom model
    spam: results[2].spam
  }
}
```

1.5 Transactions Overview (`/admin/transactions`)

Référence Visuelle: Image 5

Features:

1. **Filtres**
   - Date Range picker
   - Transaction Type dropdown (Subscription, PPV, Tip, etc.)
   - Status dropdown (All, Completed, Pending, Failed)
   - Button "Apply Filter"
   - Button "Export CSV"
2. **Revenue Trend Chart**
   - Area chart (Recharts)
   - Période sélectionnée par filtres
   - Total visible en haut: "Cibes 25K" (format automatique K/M)
3. **Payouts Trend Chart**
   - Même format que Revenue
   - Avec bars overlay pour montants discrets
4. **Transactions Table**
   - Colonnes: Date | Type (badge) | Amount | Status (badge)
   - Status badges: Completed (vert), Pending (orange), Failed (rouge)
   - Click ligne → Modal détails transaction
5. **Payouts Status List**
   - Mini-table avec Status | Status | Actions
   - Checkmarks pour Completed
   - Warning icon pour Failed
   - Actions: Fev, Fav, Fail (icons)

API Endpoints:

```typescript
GET /api/admin/transactions?dateFrom=2025-01-01&dateTo=2025-10-20&type=SUBSCRIPTION&status=COMPLETED&
Response: {
  transactions: Transaction[]
  total: number
  summary: {
    totalRevenue: number
    totalPayouts: number
    avgTransaction: number
  }
}

GET /api/admin/transactions/trends?period=30d&metric=revenue
Response: {
  date: string
  value: number
}[]

POST /api/admin/transactions/:id/refund
Body: { amount: number, reason: string }
Response: { success: boolean }
```

1.6 Accounting & Export (`/admin/accounting`)

Référence Visuelle: Image 6

Features:

1. **Export Buttons**
   - "Export CSV" | "Export PDF" (actif, turquoise) | "Plan Export"
   - Click PDF → génère rapport comptable
   - Format PDF: Header avec logo, tableau revenus/dépenses, footer avec signatures
2. **Metric Cards**
   - Revenue: €1,950 (icône euro turquoise)
   - Fees: €8,75 (icône pourcentage orange)
   - Commission: €8,750 (icône chart orange)
   - Net Profit: €14,500 (icône wallet turquoise)
3. **Export History Table**
   - Colonnes: Date | Type | Format | Initiated By | Status
   - Status badges: Completed (vert), Pending (orange), Failed (rouge), Rending (orange)
   - Note en bas: "Exports stored 7 days before purge"
   - Click ligne → Download file

Calculs Comptables:

```typescript
// Formules EXACTES
const calculations = {
  revenue: sum(transactions.filter(t => t.type === 'CREDIT')),
  fees: sum(transactions.map(t => t.processingFee)),
  commission: revenue * platformTakeRate, // ex: 15%
  netProfit: revenue - fees - commission - operatingCosts
}
```

API Endpoints:

```typescript
GET /api/admin/accounting/summary?period=month&year=2025
Response: {
  revenue: number
  fees: number
  commission: number
  netProfit: number
  breakdown: {
    subscriptions: number
    ppv: number
    tips: number
  }
}

POST /api/admin/accounting/export
Body: {
  type: 'accounting' | 'transactions' | 'payouts'
  format: 'csv' | 'pdf' | 'xlsx'
  dateFrom: string
  dateTo: string
}
Response: {
  exportId: string
  status: 'PROCESSING'
}

GET /api/admin/accounting/exports/:id/download
Response: File download
```

PDF Generation:

```typescript
// Utiliser pdfkit ou puppeteer
async generateAccountingPDF(data: AccountingSummary) {
  const pdf = new PDFDocument()

  // Header
  pdf.image('logo.png', 50, 45, { width: 50 })
  pdf.fontSize(20).text('Oliver Platform', 120, 50)
  pdf.fontSize(10).text(`Accounting Report - ${data.period}`, 120, 70)

  // Summary table
  pdf.fontSize(12).text('Financial Summary', 50, 120)
  // ... tables avec données

  // Footer
  pdf.fontSize(8).text('Generated by Oliver Admin', 50, 700)

  return pdf
}
```

1.7 Audit Log (`/admin/audit-log`)

Référence Visuelle: Image 7

Features:

1. **Filtres**
    - Date dropdown
    - Actor dropdown (filter par admin user)
    - Action Type dropdown
    - Button "Export" (CSV)
2. **Log Entries Table**
    - Colonnes: TYPE (avatar) | ACTION (texte action) | IP | DEVICE
    - Actions colorées: Created/Added (turquoise), Updated (bleu), Deleted (gris), Changed (turquoise), Suspended (rouge)
    - Grouping par date: "Yesterday", dates spécifiques
    - Pagination infinie (load more on scroll)
3. **Immutabilité**
    - Logs append-only
    - Impossible de modifier ou supprimer
    - Hashing pour intégrité

API Endpoints:

typescript

```
GET /api/admin/audit-log?actor=userId&action=user.suspend&dateFrom=&dateTo=&page=1
Response: {
  logs: {
    id: string
    timestamp: string
    actorId: string
    actorName: string
    actorAvatar: string
    action: string      // ex: "Updated payout", "Changed John Doe role: admin"
    type: string        // ex: "Robert Fox", "Dianne Russell"
    ip: string
    device: string      // ex: "MacBook Pro", "iPhone 15"
    metadata: object    // Données additionnelles
  }[]
  total: number
}

GET /api/admin/audit-log/export.csv?filters=...
Response: CSV file
```

Schema Prisma:

prisma

```
model AuditLog {
  id        String   @id @default(cuid())
  timestamp DateTime @default(now())
  actorId   String
  actor     User     @relation(fields: [actorId], references: [id])
  action    String   // Verb: "created", "updated", "deleted", "suspended"
  targetType String  // "user", "post", "transaction", "setting"
  targetId  String?
  metadata  Json     // Full details de l'action
  ip        String
  userAgent String
  device    String   // Parsed from userAgent

  @@index([actorId, timestamp])
  @@index([action, timestamp])
  @@index([timestamp])
}
```

Logging Service:

typescript

```typescript
// Service centralisé pour tous les logs
class AuditLogService {
  async log(params: {
    actorId: string
    action: string
    targetType: string
    targetId?: string
    metadata: object
    request: Request
  }) {
    const ip = getClientIp(request)
    const device = parseUserAgent(request.headers['user-agent'])

    await prisma.auditLog.create({
      data: {
        ...params,
        ip,
        device,
        timestamp: new Date()
      }
    })
  }
}

// Utilisation
await auditLog.log({
  actorId: admin.id,
  action: 'user.suspend',
  targetType: 'user',
  targetId: suspendedUser.id,
  metadata: { reason: 'TOS violation', duration: '7d' },
  request
})
```

1.8 Settings (`/admin/settings`)

Référence Visuelle: Image 8

Features:

1. **Tabs Navigation**
    - General (actif) | Security | Notifications | Roles & Permissions
2. **General Tab**
    - Platform Info card:
        - Platform Name: input (OLIVER)
        - Contact Email: input
    - Preferences card:
        - Enable Dark Mode: toggle
        - Auto Backup: toggle
        - Show tooltips for new users: toggle (actif)
3. **Security Tab**
    - Password Length: number input (8)
    - 2FA Requirement: toggle (actif)
    - Allowed IPs: textarea
    - Button "Reset API Key" (rouge outline)
4. **Buttons**
    - Cancel (outline) | Save (primary turquoise)

API Endpoints:

typescript

```
GET /api/admin/settings
Response: {
  platformName: string
  contactEmail: string
  darkMode: boolean
  autoBackup: boolean
  showTooltips: boolean
  security: {
    passwordLength: number
    require2FA: boolean
    allowedIPs: string[]
    apiKey: string  // Masked
  }
}

PATCH /api/admin/settings
Body: Partial<Settings>
Response: { success: boolean, settings: Settings }

POST /api/admin/settings/api-key/reset
Response: {
  success: boolean,
  newApiKey: string  // Affiché une seule fois
}
```

Validation:

typescript

```
// Zod schema pour validation
const settingsSchema = z.object({
  platformName: z.string().min(3).max(50),
  contactEmail: z.string().email(),
  darkMode: z.boolean(),
  autoBackup: z.boolean(),
  showTooltips: z.boolean(),
  security: z.object({
    passwordLength: z.number().min(8).max(32),
    require2FA: z.boolean(),
    allowedIPs: z.array(z.string().ip())
  })
})
```

1.9 KYC Validation (`/admin/kyc`)

Référence Visuelle: Image 9

Features CRITIQUES:

1. **Breadcrumb**
   - Dashboard / Users / KYC Validation / Revview
   - Dropdown status: Pending (actif)
2. **Submitted Information (gauche)**
   - Full Name: Jane Smith
   - Date of Birth: March 13, 1984
   - Nationality: United States, Springfield, IL62701
   - ID Number: 123456789
   - Expiration Date: September 12, 2028
   - Submission Date: October 22, 2025
   - KYC ID: KYC-20210222.00124
3. **Documents Section**
   - 3 thumbnails: Front ID | Back ID | Proof of Address
   - Click thumbnail → Lightbox agrandit
   - Zoom in/out sur documents
4. **Identity Card Preview (droite)**
   - Reconstruction visuelle du document
   - Photo + Nom + ID + Expiration
   - Icons zoom in/out et fullscreen
5. **AI Confidence Analysis**
   - Identity Match: 82% (barre turquoise)
   - Liveness: 65% (barre orange) ⚠️
   - Document Authenticity: 95% (barre turquoise)
   - Seuil LOW: <60%, MEDIUM: 60-80%, HIGH: >80%
6. **Actions**
   - Approve (turquoise) | Reject (rouge) | Request Review (outline rouge)
   - Approve → User devient VERIFIED
   - Reject → Modal avec raison (required)
   - Request Review → Escalate à senior moderator

API Endpoints:

```typescript
GET /api/admin/kyc/pending?page=1
Response: {
  kycs: {
    id: string
    userId: string
    user: { name, email, avatar }
    status: 'PENDING' | 'VERIFIED' | 'REJECTED'
    submittedAt: string
    documents: {
      front: string    // S3 URL
      back: string
      proof: string
    }
    data: {
      fullName: string
      dateOfBirth: string
      nationality: string
      idNumber: string
      expirationDate: string
    }
    aiScores: {
      identityMatch: number
      liveness: number
      documentAuthenticity: number
    }
  }[]
}

POST /api/admin/kyc/:id/approve
Body: { notes?: string }
Response: { success: boolean }

POST /api/admin/kyc/:id/reject
Body: { reason: string, notes?: string }
Response: { success: boolean }

POST /api/admin/kyc/:id/request-review
Body: { notes: string }
Response: { success: boolean }
```

Intégration Yoti/Jumio:

typescript

```typescript
// Webhook handler pour résultats AI
async handleYotiWebhook(payload: YotiWebhookPayload) {
  const kyc = await prisma.kycVerification.findUnique({
    where: { externalId: payload.session_id }
  })

  await prisma.kycVerification.update({
    where: { id: kyc.id },
    data: {
      aiScores: {
        identityMatch: payload.checks.document_match.score,
        liveness: payload.checks.liveness.score,
        documentAuthenticity: payload.checks.document_authenticity.score
      },
      status: payload.result === 'PASS' ? 'PENDING' : 'REJECTED'
    }
  })

  // Si tous scores >80%, auto-approve
  if (allScoresAbove80(payload.checks)) {
    await this.approveKyc(kyc.id)
  }
}
```

Document Storage:

```typescript
// Encryption des documents sensibles
async storeKycDocument(file: File, userId: string) {
  // 1. Encrypt file
  const encrypted = await encrypt(file.buffer, process.env.KYC_ENCRYPTION_KEY)

  // 2. Upload to S3 avec server-side encryption
  const key = `kyc/${userId}/${uuid()}`
  await s3.upload({
    Bucket: 'oliver-kyc-documents',
    Key: key,
    Body: encrypted,
    ServerSideEncryption: 'AES256',
    Metadata: {
      userId,
      uploadedAt: new Date().toISOString()
    }
  })

  // 3. Store only reference in DB, never raw URL
  return key
}

// Accès temporaire pour review
async getKycDocumentUrl(key: string, adminId: string) {
  // Log access
  await auditLog.log({
    actorId: adminId,
    action: 'kyc.document.viewed',
    metadata: { documentKey: key }
  })

  // Generate signed URL (expire 5 minutes)
  return s3.getSignedUrl('getObject', {
    Bucket: 'oliver-kyc-documents',
    Key: key,
    Expires: 300
  })
}
```

🔧 SECTION 2: BACKEND API COMPLET

Structure NestJS

```
apps/api/src/modules/admin/
├── admin.module.ts
├── controllers/
│   ├── dashboard.controller.ts
│   ├── users.controller.ts
│   ├── reports.controller.ts
│   ├── moderation.controller.ts
│   ├── transactions.controller.ts
│   ├── accounting.controller.ts
│   ├── audit-log.controller.ts
│   ├── kyc.controller.ts
│   └── settings.controller.ts
├── services/
│   ├── dashboard.service.ts
│   ├── users.service.ts
│   ├── reports.service.ts
│   ├── moderation.service.ts
│   ├── transactions.service.ts
│   ├── accounting.service.ts
│   ├── audit-log.service.ts
│   ├── kyc.service.ts
│   └── settings.service.ts
├── dto/
│   └── [tous les DTOs avec validation Zod]
├── guards/
│   ├── admin-role.guard.ts
│   └── permissions.guard.ts
└── decorators/
    ├── admin.decorator.ts
    └── audit-log.decorator.ts
```

Guards & Middleware

admin-role.guard.ts:

typescript

```typescript
@Injectable()
export class AdminRoleGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean {
    const request = context.switchToHttp().getRequest()
    const user = request.user

    if (!user || !['ADMIN', 'MODERATOR'].includes(user.role)) {
      throw new UnauthorizedException('Admin access required')
    }

    return true
  }
}
```

permissions.guard.ts:

typescript

```typescript
// Granular permissions par action
const PERMISSIONS = {
  'users.read': ['ADMIN', 'MODERATOR'],
  'users.suspend': ['ADMIN'],
  'kyc.approve': ['ADMIN', 'SENIOR_MODERATOR'],
  'settings.write': ['ADMIN']
}

@Injectable()
export class PermissionsGuard implements CanActivate {
  constructor(private reflector: Reflector) {}

  canActivate(context: ExecutionContext): boolean {
    const permission = this.reflector.get<string>('permission', context.getHandler())
    const user = context.switchToHttp().getRequest().user

    return PERMISSIONS[permission]?.includes(user.role) ?? false
  }
}

// Decorator
export const RequirePermission = (permission: string) => SetMetadata('permission', permission)
```

audit-log.decorator.ts:

typescript

```typescript
// Decorator pour logger automatiquement les actions admin
export const AuditLog = (action: string) => {
  return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {
    const originalMethod = descriptor.value

    descriptor.value = async function (...args: any[]) {
      const result = await originalMethod.apply(this, args)

      // Log après succès de l'action
      const request = args.find(arg => arg.user)
      await this.auditLogService.log({
        actorId: request.user.id,
        action,
        metadata: { args, result },
        request
      })

      return result
    }

    return descriptor
  }
}

// Usage
@AuditLog('user.suspend')
async suspendUser(userId: string, reason: string) {
  // ...
}
```

Controllers Complets

dashboard.controller.ts

typescript

```typescript
@Controller('admin/dashboard')
@UseGuards(AdminRoleGuard)
export class DashboardController {
  constructor(private dashboardService: DashboardService) {}

  @Get('metrics')
  async getMetrics() {
    return this.dashboardService.getMetrics()
  }

  @Get('sales')
  async getSalesOverview(@Query('period') period: string) {
    return this.dashboardService.getSalesOverview(period)
  }

  @Get('transactions')
  async getRecentTransactions(@Query('limit') limit: number = 5) {
    return this.dashboardService.getRecentTransactions(limit)
  }
}
```

users.controller.ts

typescript

```typescript
@Controller('admin/users')
@UseGuards(AdminRoleGuard)
export class UsersController {
  constructor(
    private usersService: UsersService,
    private auditLogService: AuditLogService
  ) {}

  @Get()
  async getUsers(@Query() filters: UsersFilterDto) {
    return this.usersService.findAll(filters)
  }

  @Post(':id/suspend')
  @RequirePermission('users.suspend')
  @AuditLog('user.suspend')
  async suspendUser(
    @Param('id') userId: string,
    @Body() dto: SuspendUserDto,
    @Req() req
  ) {
    return this.usersService.suspendUser(userId, dto.reason, req.user.id)
  }

  @Post('bulk-suspend')
  @RequirePermission('users.suspend')
  @AuditLog('users.bulk-suspend')
  async bulkSuspend(@Body() dto: BulkSuspendDto) {
    return this.usersService.bulkSuspend(dto.userIds, dto.reason)
  }

  @Post(':id/reset-password')
  @RequirePermission('users.write')
  @AuditLog('user.password-reset')
  async resetPassword(@Param('id') userId: string) {
    return this.usersService.resetPassword(userId)
  }

  @Get('export.csv')
  async exportUsers(@Query() filters: UsersFilterDto, @Res() res) {
    const csv = await this.usersService.exportToCSV(filters)
    res.header('Content-Type', 'text/csv')
    res.attachment('users-export.csv')
    return res.send(csv)
  }
```

```
  }
}
```

[Continuer pour tous les autres controllers...]

🗄️ SECTION 3: DATABASE SCHEMAS COMPLETS

Modèles Prisma à Ajouter

Fichier: `packages/database/prisma/schema.prisma`

Ajouter ces modèles:

prisma

```
// ==========================================
// ADMIN MODELS
// ==========================================

// Audit Log pour traçabilité
model AuditLog {
  id        String   @id @default(cuid())
  timestamp DateTime @default(now())
  actorId   String
  actor     User     @relation("AuditLogActor", fields: [actorId], references: [id])
  action    String   // Format: "resource.action" ex: "user.suspend"
  targetType String  // "user", "post", "transaction", "setting"
  targetId   String?
  metadata  Json     // Détails complets de l'action
  ip        String
  userAgent String
  device    String   // Parsed device info

  @@index([actorId, timestamp])
  @@index([action, timestamp])
  @@index([targetType, targetId])
  @@index([timestamp])
  @@map("audit_logs")
}

// Reports System
model Report {
  id             String       @id @default(cuid())
  reporterId     String
  reporter       User         @relation("ReportsMade", fields: [reporterId], references: [id])
  targetId       String       // ID du contenu/user rapporté
  targetType     TargetType
  reason         String
  description    String?
  priority       Priority     @default(MEDIUM)
  status         ReportStatus @default(PENDING)
  assignedToId   String?
  assignedTo     User?        @relation("ReportsAssigned", fields: [assignedToId], references: [id])
  reviewedById   String?
  reviewedBy     User?        @relation("ReportsReviewed", fields: [reviewedById], references: [id])
  reviewedAt     DateTime?
  aiAnalysis     Json?        // Scores AI moderation
  escalationLog  Json[]       @default([])
  resolution     String?
```

```
  createdAt       DateTime     @default(now())
  updatedAt       DateTime     @updatedAt

  @@index([status, priority])
  @@index([assignedToId])
  @@index([targetId, targetType])
  @@index([createdAt])
  @@map("reports")
}

enum TargetType {
  USER
  POST
  MESSAGE
  COMMENT
  PROFILE
}

enum Priority {
  LOW
  MEDIUM
  HIGH
  CRITICAL
}

enum ReportStatus {
  PENDING
  UNDER_REVIEW
  RESOLVED
  DISMISSED
  ESCALATED
}

// KYC Verifications
model KycVerification {
  id              String       @id @default(cuid())
  userId          String       @unique
  user            User         @relation(fields: [userId], references: [id])
  provider        String       // "yoti", "jumio"
  externalId      String?      // ID chez le provider
  status          KycStatus
  documents       Json         // { front: url, back: url, selfie: url, proof: url }
  documentKeys    Json         // S3 keys chiffrées
  personalData    Json         // fullName, dob, nationality, idNumber, etc.
  aiScores        Json         // { identityMatch, liveness, documentAuthenticity }
  reviewedById    String?
  reviewedBy      User?        @relation("KycReviewedBy", fields: [reviewedById], references: [id])
```

```
  reviewedAt      DateTime?
  rejectionReason String?
  expiresAt       DateTime     // Re-verification needed after
  createdAt       DateTime     @default(now())
  updatedAt       DateTime     @updatedAt

  @@index([userId])
  @@index([status])
  @@index([expiresAt])
  @@map("kyc_verifications")
}

// Moderation Decisions
model ModerationDecision {
  id          String   @id @default(cuid())
  contentId   String
  contentType String   // POST, MESSAGE, PROFILE, COMMENT
  moderatorId String
  moderator   User     @relation("ModerationDecisions", fields: [moderatorId], references: [id])
  decision    Decision
  reason      String
  aiScorePre  Json?    // Scores AI avant modération
  notes       String?
  createdAt   DateTime @default(now())

  @@index([contentId, contentType])
  @@index([moderatorId])
  @@index([createdAt])
  @@map("moderation_decisions")
}

enum Decision {
  APPROVE
  REJECT
  ESCALATE
  REQUEST_CHANGES
}

// Export History
model ExportHistory {
  id          String       @id @default(cuid())
  type        String       // "accounting", "transactions", "users", "audit-log"
  format      String       // "csv", "pdf", "xlsx"
  initiatedById String
  initiatedBy  User        @relation("ExportsInitiated", fields: [initiatedById], references: [id])
  status      ExportStatus @default(PROCESSING)
```

```
  fileUrl      String?      // S3 URL (signed, expires 7 days)
  fileKey      String?      // S3 key pour régénération
  fileSize     Int?         // bytes
  rowCount     Int?
  errorMessage String?
  filters      Json?        // Filtres appliqués
  createdAt    DateTime     @default(now())
  completedAt  DateTime?
  expiresAt    DateTime?    // Auto-delete après 7 jours

  @@index([initiatedById])
  @@index([status])
  @@index([createdAt])
  @@index([expiresAt])
  @@map("export_history")
}

enum ExportStatus {
  PROCESSING
  COMPLETED
  FAILED
  EXPIRED
}

// Settings (singleton)
model Settings {
  id              String  @id @default("singleton")
  platformName    String  @default("OLIVER")
  contactEmail    String
  darkMode        Boolean @default(false)
  autoBackup      Boolean @default(true)
  showTooltips    Boolean @default(true)
  passwordLength  Int     @default(8)
  require2FA      Boolean @default(false)
  allowedIPs      String[]
  apiKey          String  @unique
  updatedAt       DateTime @updatedAt
  updatedBy       String?

  @@map("settings")
}

// Moderator Sessions (pour protection)
model ModeratorSession {
  id              String   @id @default(cuid())
  moderatorId     String
  moderator       User     @relation("ModeratorSessions", fields: [moderatorId], references: [id])
```

```
  moderator      User   @relation("ModeratorSessions", fields: [moderatorId], references: [id])
  startedAt      DateTime @default(now())
  endedAt        DateTime?
  itemsReviewed  Int     @default(0)
  breaksT taken  Int     @default(0)
  lastBreakAt    DateTime?
  status         String  // ACTIVE, BREAK, ENDED

  @@index([moderatorId, startedAt])
  @@map("moderator_sessions")
}

// ============================================
// UPDATE EXISTING USER MODEL
// ============================================
model User {
  // ... existing fields ...

  // Admin relations
  auditLogsActor      AuditLog[]           @relation("AuditLogActor")
  reportsMade         Report[]             @relation("ReportsMade")
  reportsAssigned     Report[]             @relation("ReportsAssigned")
  reportsReviewed     Report[]             @relation("ReportsReviewed")
  kycVerification     KycVerification?
  kycReviewed         KycVerification[]    @relation("KycReviewedBy")
  moderationDecisions ModerationDecision[] @relation("ModerationDecisions")
  exportsInitiated    ExportHistory[]      @relation("ExportsInitiated")
  moderatorSessions   ModeratorSession[]   @relation("ModeratorSessions")
}
```

Migration Commands

```bash
# 1. Créer la migration
cd packages/database
npx prisma migrate dev --name add-admin-models

# 2. Générer le client Prisma
npx prisma generate

# 3. Seed initial data (settings)
npx prisma db seed
```

seed.ts:

typescript

```typescript
async function main() {
  // Create default settings
  await prisma.settings.upsert({
    where: { id: 'singleton' },
    update: {},
    create: {
      id: 'singleton',
      platformName: 'OLIVER',
      contactEmail: 'contact@oliver.com',
      darkMode: false,
      autoBackup: true,
      showTooltips: true,
      passwordLength: 8,
      require2FA: false,
      allowedIPs: [],
      apiKey: generateApiKey()
    }
  })

  // Create first admin user
  await prisma.user.upsert({
    where: { email: 'admin@oliver.com' },
    update: {},
    create: {
      email: 'admin@oliver.com',
      username: 'admin',
      passwordHash: await hash('Admin123!', 10),
      role: 'ADMIN',
      emailVerified: true,
      ageVerified: true,
      kycStatus: 'VERIFIED'
    }
  })
}
```

🔒 SECTION 4: SÉCURITÉ & PERMISSIONS

Rate Limiting

typescript

```typescript
// apps/api/src/common/guards/rate-limit.guard.ts
@Injectable()
export class RateLimitGuard implements CanActivate {
  constructor(private redis: Redis) {}

  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest()
    const key = `ratelimit:admin:${request.user.id}:${request.path}`

    const requests = await this.redis.incr(key)
    if (requests === 1) {
      await this.redis.expire(key, 60) // 1 minute window
    }

    // 100 requests par minute pour admins
    if (requests > 100) {
      throw new ThrottlerException('Too many requests')
    }

    return true
  }
}
```

Input Validation

typescript

```typescript
// DTOs avec validation Zod stricte
import { z } from 'zod'

export const SuspendUserSchema = z.object({
  reason: z.string().min(10).max(500),
  duration: z.number().min(1).max(365).optional(), // jours
  notifyUser: z.boolean().default(true)
})

export type SuspendUserDto = z.infer<typeof SuspendUserSchema>

// Pipe de validation global
@Injectable()
export class ZodValidationPipe implements PipeTransform {
  constructor(private schema: z.ZodSchema) {}

  transform(value: any) {
    try {
      return this.schema.parse(value)
    } catch (error) {
      throw new BadRequestException('Validation failed', error.errors)
    }
  }
}

// Usage dans controller
@Post(':id/suspend')
async suspendUser(
  @Param('id') userId: string,
  @Body(new ZodValidationPipe(SuspendUserSchema)) dto: SuspendUserDto
) {
  // ...
}
```

SQL Injection Prevention

```typescript
// TOUJOURS utiliser Prisma (parameterized queries)
// JAMAIS de raw queries sauf absolument nécessaire

// ✅ CORRECT
await prisma.user.findMany({
  where: {
    email: { contains: searchTerm }
  }
})

// ❌ INCORRECT
await prisma.$queryRaw`SELECT * FROM users WHERE email LIKE '%${searchTerm}%'`

// Si raw query nécessaire, utiliser Prisma.sql
await prisma.$queryRaw(
  Prisma.sql`SELECT * FROM users WHERE email LIKE ${`%${searchTerm}%`}`
)
```

XSS Prevention

```typescript
// Sanitize tous les inputs utilisateur
import DOMPurify from 'isomorphic-dompurify'

function sanitizeInput(input: string): string {
  return DOMPurify.sanitize(input, {
    ALLOWED_TAGS: [], // Strip all HTML
    ALLOWED_ATTR: []
  })
}

// Apply dans DTOs
export class CreateReportDto {
  @Transform(({ value }) => sanitizeInput(value))
  reason: string

  @Transform(({ value }) => sanitizeInput(value))
  description?: string
}
```

CORS Configuration

typescript

```typescript
// apps/api/src/main.ts
app.enableCors({
  origin: [
    'https://oliver.com',
    'https://admin.oliver.com',
    process.env.NODE_ENV === 'development' && 'http://localhost:3000'
  ].filter(Boolean),
  credentials: true,
  methods: ['GET', 'POST', 'PATCH', 'DELETE'],
  allowedHeaders: ['Content-Type', 'Authorization']
})
```

📊 SECTION 5: PERFORMANCE & OPTIMISATION

Caching Strategy

typescript

```typescript
// Redis caching pour données fréquemment consultées
@Injectable()
export class DashboardService {
  constructor(
    private prisma: PrismaService,
    private redis: Redis
  ) {}

  async getMetrics() {
    const cacheKey = 'admin:dashboard:metrics'

    // Try cache first
    const cached = await this.redis.get(cacheKey)
    if (cached) {
      return JSON.parse(cached)
    }

    // Compute metrics
    const [totalUsers, revenue, pendingPayouts] = await Promise.all([
      this.prisma.user.count(),
      this.calculateRevenue(),
      this.calculatePendingPayouts()
    ])

    const metrics = { totalUsers, revenue, pendingPayouts }

    // Cache for 30 seconds
    await this.redis.set(cacheKey, JSON.stringify(metrics), 'EX', 30)

    return metrics
  }
}
```

Database Indexes

prisma

```
// Optimisation requêtes fréquentes
model User {
  // ...

  @@index([role, createdAt])      // Filter by role + sort
  @@index([email, username])      // Search
  @@index([kycStatus, updatedAt]) // KYC queue
}

model Payment {
  // ...

  @@index([status, createdAt])    // Transaction filters
  @@index([userId, type])         // User transactions
}

model AuditLog {
  // ...

  @@index([actorId, timestamp(sort: Desc)])  // User activity
  @@index([action, timestamp(sort: Desc)])   // Action type filter
}
```

Pagination

typescript

```typescript
// Cursor-based pagination pour grandes datasets
async function getTransactions(
  cursor?: string,
  limit: number = 50
) {
  return prisma.payment.findMany({
    take: limit + 1, // +1 pour savoir si plus de résultats
    cursor: cursor ? { id: cursor } : undefined,
    orderBy: { createdAt: 'desc' }
  }).then(items => {
    const hasMore = items.length > limit
    const data = hasMore ? items.slice(0, -1) : items
    const nextCursor = hasMore ? items[limit].id : null

    return { data, nextCursor, hasMore }
  })
}
```

Query Optimization

typescript

```typescript
// Utiliser select et include judicieusement
// ❌ BAD - Charge toutes les relations
const users = await prisma.user.findMany()

// ✅ GOOD - Select uniquement les champs nécessaires
const users = await prisma.user.findMany({
  select: {
    id: true,
    email: true,
    username: true,
    role: true,
    createdAt: true,
    _count: {
      select: {
        posts: true,
        subscriptions: true
      }
    }
  }
})

// ✅ GOOD - Aggregate pour statistiques
const stats = await prisma.payment.aggregate({
  where: { status: 'SUCCESS' },
  _sum: { amount: true },
  _avg: { amount: true },
  _count: true
})
```

🧪 SECTION 6: TESTS

Structure Tests

```
apps/api/src/modules/admin/
└── __tests__/
    ├── unit/
    │   ├── dashboard.service.spec.ts
    │   ├── users.service.spec.ts
    │   └── ...
    ├── integration/
    │   ├── dashboard.controller.spec.ts
    │   ├── users.controller.spec.ts
    │   └── ...
    └── e2e/
        └── admin-flow.e2e.spec.ts
```

Example Unit Test

typescript

```typescript
// dashboard.service.spec.ts
describe('DashboardService', () => {
  let service: DashboardService
  let prisma: PrismaService

  beforeEach(async () => {
    const module = await Test.createTestingModule({
      providers: [
        DashboardService,
        {
          provide: PrismaService,
          useValue: {
            user: { count: jest.fn() },
            payment: { aggregate: jest.fn() }
          }
        }
      ]
    }).compile()

    service = module.get(DashboardService)
    prisma = module.get(PrismaService)
  })

  describe('getMetrics', () => {
    it('should return dashboard metrics', async () => {
      jest.spyOn(prisma.user, 'count').mockResolvedValue(12450)
      jest.spyOn(prisma.payment, 'aggregate').mockResolvedValue({
        _sum: { amount: 1892000 }
      })

      const result = await service.getMetrics()

      expect(result).toEqual({
        totalUsers: 12450,
        revenue: 1892000,
        pendingPayouts: expect.any(Number)
      })
    })

    it('should handle errors gracefully', async () => {
      jest.spyOn(prisma.user, 'count').mockRejectedValue(new Error('DB error'))

      await expect(service.getMetrics()).rejects.toThrow()
    })
  })
```

```
  })
})
```

Example E2E Test

typescript

```typescript
// admin-flow.e2e.spec.ts
describe('Admin Flow (E2E)', () => {
  let app: INestApplication
  let adminToken: string

  beforeAll(async () => {
    // Setup test app
    const moduleFixture = await Test.createTestingModule({
      imports: [AppModule]
    }).compile()

    app = moduleFixture.createNestApplication()
    await app.init()

    // Get admin token
    const response = await request(app.getHttpServer())
      .post('/auth/login')
      .send({ email: 'admin@test.com', password: 'Test123!' })

    adminToken = response.body.token
  })

  it('should get dashboard metrics', async () => {
    const response = await request(app.getHttpServer())
      .get('/admin/dashboard/metrics')
      .set('Authorization', `Bearer ${adminToken}`)
      .expect(200)

    expect(response.body).toHaveProperty('totalUsers')
    expect(response.body).toHaveProperty('revenue')
  })

  it('should suspend a user', async () => {
    const response = await request(app.getHttpServer())
      .post('/admin/users/test-user-id/suspend')
      .set('Authorization', `Bearer ${adminToken}`)
      .send({ reason: 'TOS violation', duration: 7 })
      .expect(200)

    expect(response.body.success).toBe(true)
  })

  it('should reject non-admin access', async () => {
    // Login as regular user
```

```
    const userResponse = await request(app.getHttpServer())
      .post('/auth/login')
      .send({ email: 'user@test.com', password: 'User123!' })

    // Try to access admin endpoint
    await request(app.getHttpServer())
      .get('/admin/dashboard/metrics')
      .set('Authorization', `Bearer ${userResponse.body.token}`)
      .expect(403)
  })
})
```

📦 SECTION 7: DÉPLOIEMENT & CI/CD

Environment Variables

bash

```
# apps/api/.env.production
# Database
DATABASE_URL="postgresql://..."
REDIS_URL="redis://..."

# Authentication
JWT_SECRET="..."
JWT_EXPIRY="7d"

# AWS
AWS_ACCESS_KEY_ID="..."
AWS_SECRET_ACCESS_KEY="..."
AWS_REGION="eu-west-1"
S3_BUCKET_KYC="oliver-kyc-prod"

# External Services
YOTI_CLIENT_SDK_ID="..."
YOTI_KEY_FILE_PATH="/secrets/yoti-key.pem"
GOOGLE_VISION_API_KEY="..."
AWS_REKOGNITION_REGION="eu-west-1"

# Email
SENDGRID_API_KEY="..."
FROM_EMAIL="no-reply@oliver.com"

# Monitoring
SENTRY_DSN="..."
DATADOG_API_KEY="..."

# Feature Flags
ENABLE_AUTO_KYC_APPROVAL=false
MAX_MODERATION_SESSION_HOURS=4
```

Docker Configuration

dockerfile

```
# apps/api/Dockerfile
FROM node:20-alpine AS builder

WORKDIR /app
COPY package*.json pnpm-lock.yaml ./
RUN npm install -g pnpm && pnpm install --frozen-lockfile

COPY . .
RUN pnpm prisma generate
RUN pnpm build

FROM node:20-alpine AS runner

WORKDIR /app
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/package.json ./

EXPOSE 4000
CMD ["node", "dist/main.js"]
```

GitHub Actions CI/CD

yaml

```yaml
# .github/workflows/deploy-admin.yml
name: Deploy Admin Section

on:
  push:
    branches: [main]
    paths:
      - 'apps/web/app/(admin)/**'
      - 'apps/api/src/modules/admin/**'

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: pnpm/action-setup@v2
      - uses: actions/setup-node@v3
        with:
          node-version: '20'
          cache: 'pnpm'

      - run: pnpm install
      - run: pnpm test:admin
      - run: pnpm test:e2e:admin

  build-and-deploy:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Build Docker images
        run: |
          docker build -t oliver-api:${{ github.sha }} ./apps/api
          docker build -t oliver-web:${{ github.sha }} ./apps/web

      - name: Push to ECR
        run: |
          aws ecr get-login-password | docker login --username AWS --password-stdin $ECR_URL
          docker push oliver-api:${{ github.sha }}
          docker push oliver-web:${{ github.sha }}

      - name: Deploy to Kubernetes
        run: |
```

```
kubectl set image deployment/api api=oliver-api:${{ github.sha }}
kubectl set image deployment/web web=oliver-web:${{ github.sha }}
kubectl rollout status deployment/api
kubectl rollout status deployment/web
```

🎯 SECTION 10: FONCTIONNALITÉS CRITIQUES ADDITIONNELLES

IMPORTANT: Phase de Développement

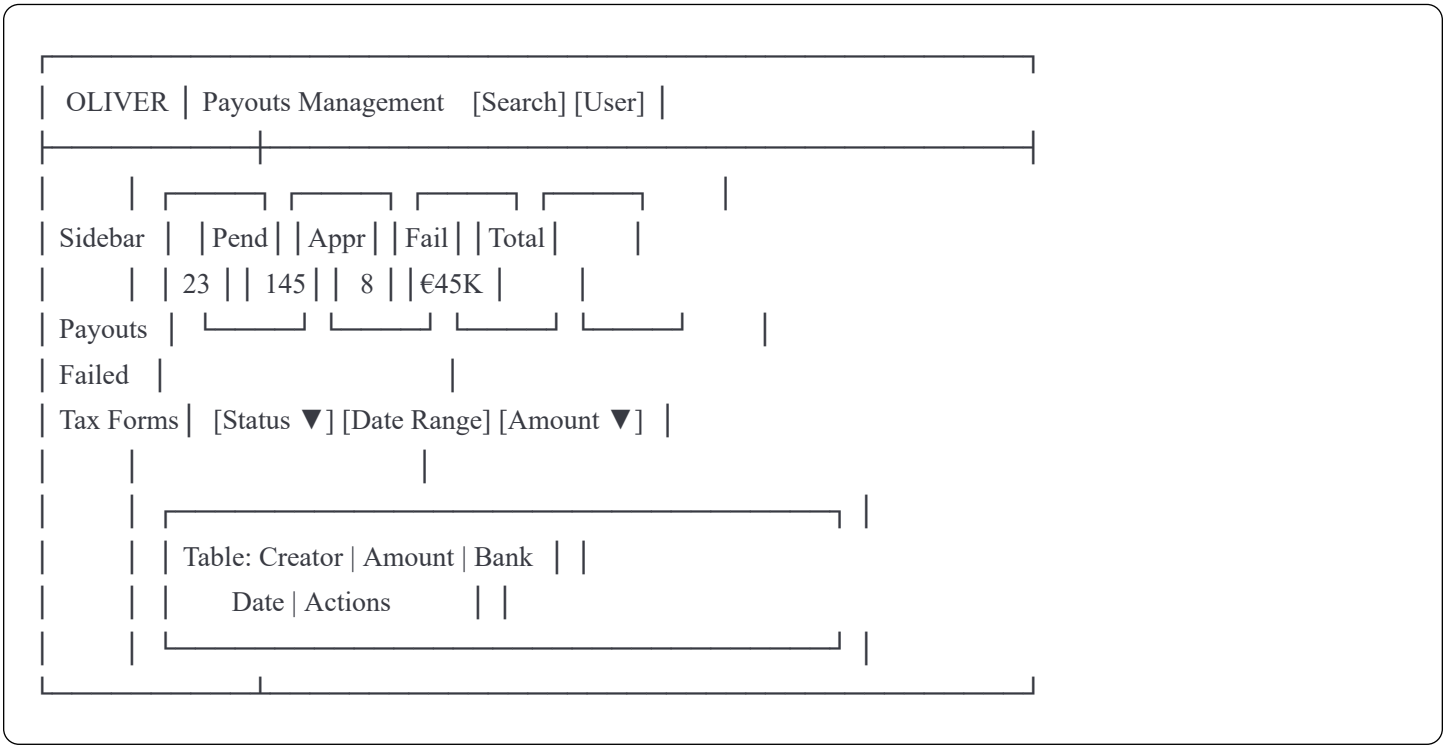
Les sections suivantes sont **CRITIQUES** pour une plateforme production-ready. Développe-les **APRÈS** les 9 écrans de base.

10.1 Payout Management (ÉCRANS 10-12)

PRIORITÉ: CRITIQUE - Sans ça, créateurs ne peuvent pas être payés

10.1.1 Payouts Queue (`/admin/payouts`)

Layout:



Features à implémenter:

1. **Metric Cards** (4 cards)
   - Pending: Nombre + montant total
   - Approved Today: Nombre + montant
   - Failed: Nombre (rouge si > 0)
   - Total This Month: Montant total payé
2. **Filtres**
   - Status: All | Pending | Approved | Processing | Failed | Rejected
   - Date Range: Last 7 days | Last 30 days | Custom
   - Amount Range: slider (min-max)
   - Payment Method: Bank Transfer | PayPal | Crypto
3. **Table Payouts**
   - Colonnes: [Checkbox] | Creator (avatar+name) | Amount | Method | Bank Details (masked) | Requested Date | Status | Actions
   - Status badges:
     - Pending (orange)
     - Approved (blue)
     - Processing (purple)
     - Completed (green)
     - Failed (red)
     - Rejected (gray)
   - Actions dropdown: Approve | Reject | Hold | View Details
4. **Bulk Actions**
   - Checkbox "Select All"
   - Bulk Approve button (avec confirmation)
   - Vérifications avant bulk approve:
     - Bank details valides
     - Tax form complété si montant > seuil
     - Pas de flags fraud
     - Solde suffisant
5. **Payout Detail Modal**

- Creator info complète
- Montant breakdown (revenus - fees - commission)
- Bank details (full, décryptés)
- Transaction history
- Tax form status
- Fraud score (si disponible)
- Boutons: Approve (vert) | Reject (rouge) | Request Info (bleu)

Fichier: `apps/web/app/(admin)/payouts/page.tsx`

API Endpoints:

typescript

```
GET /api/admin/payouts?status=PENDING&method=BANK&dateFrom=&dateTo=&page=1
Response: {
  payouts: {
    id: string
    creatorId: string
    creator: {
      id: string
      name: string
      avatar: string
      email: string
    }
    amount: number          // centimes
    currency: string
    method: 'BANK_TRANSFER' | 'PAYPAL' | 'CRYPTO'
    bankDetails: {
      iban: string          // Encrypted, decrypt pour admin
      bic: string
      accountName: string
    }
    requestedAt: string
    status: PayoutStatus
    taxFormStatus: TaxFormStatus
    fraudScore?: number
  }[]
  total: number
  summary: {
    pending: { count: number, amount: number }
    approved: { count: number, amount: number }
    failed: { count: number, amount: number }
  }
}

POST /api/admin/payouts/:id/approve
Body: { notes?: string }
Response: { success: boolean, payout: Payout }

POST /api/admin/payouts/:id/reject
Body: { reason: string, notes?: string }
Response: { success: boolean }

POST /api/admin/payouts/bulk-approve
Body: { payoutIds: string[], notes?: string }
Response: {
  success: boolean
```

```
  approved: number
  failed: { id: string, reason: string }[]
}

POST /api/admin/payouts/:id/hold
Body: { reason: string, duration: number }
Response: { success: boolean }
```

10.1.2 Failed Payouts (`/admin/payouts/failed`)

Features:

1. **Failed Payouts Table**
   - Colonnes: Creator | Amount | Method | Failure Reason | Failed At | Retry Count | Actions
   - Failure reasons typiques:
     - "Invalid bank details"
     - "Account closed"
     - "Insufficient funds"
     - "Fraud detected"
     - "Processor error"
2. **Retry Mechanism**
   - Manual retry button
   - Auto-retry configuré (3 tentatives, 24h intervalle)
   - Retry count visible
   - Max retries: 3
3. **Contact Creator Workflow**
   - Button "Contact Creator"
   - Template email pré-rempli
   - Request updated bank details
   - Track communication

API Endpoints:

```typescript
GET /api/admin/payouts/failed
Response: {
  payouts: FailedPayout[]
  retrySchedule: { payoutId: string, nextRetry: string }[]
}

POST /api/admin/payouts/:id/retry
Response: { success: boolean, status: string }

POST /api/admin/payouts/:id/contact-creator
Body: { message: string, requestBankUpdate: boolean }
Response: { success: boolean, emailSent: boolean }
```

10.1.3 Tax Forms Management (`/admin/payouts/tax-forms`)

Features:

1. **Tax Forms Dashboard**
   - Metric cards:
     - Forms Required: nombre de créateurs au-dessus du seuil
     - Forms Completed: pourcentage
     - Forms Pending Review: nombre
     - 1099 Generation Ready: nombre (fin d'année)
2. **Creators Requiring Tax Forms**
   - Table: Creator | Country | YTD Earnings | Form Type | Status | Actions
   - Form types:
     - W-9 (US citizens/residents)
     - W-8BEN (Non-US individuals)
     - W-8BEN-E (Non-US entities)
   - Status: Not Started | Submitted | Approved | Rejected
3. **Tax Form Review**
   - Document viewer
   - Validation checklist:
     - Name matches account
     - TIN/SSN format valide
     - Signature présente
     - Date récente
   - Actions: Approve | Reject | Request Correction
4. **1099 Generation (End of Year)**
   - Automated 1099-NEC generation
   - Seuil: $600+ per year
   - Export pour comptable
   - E-file avec IRS (si configuré)

API Endpoints:

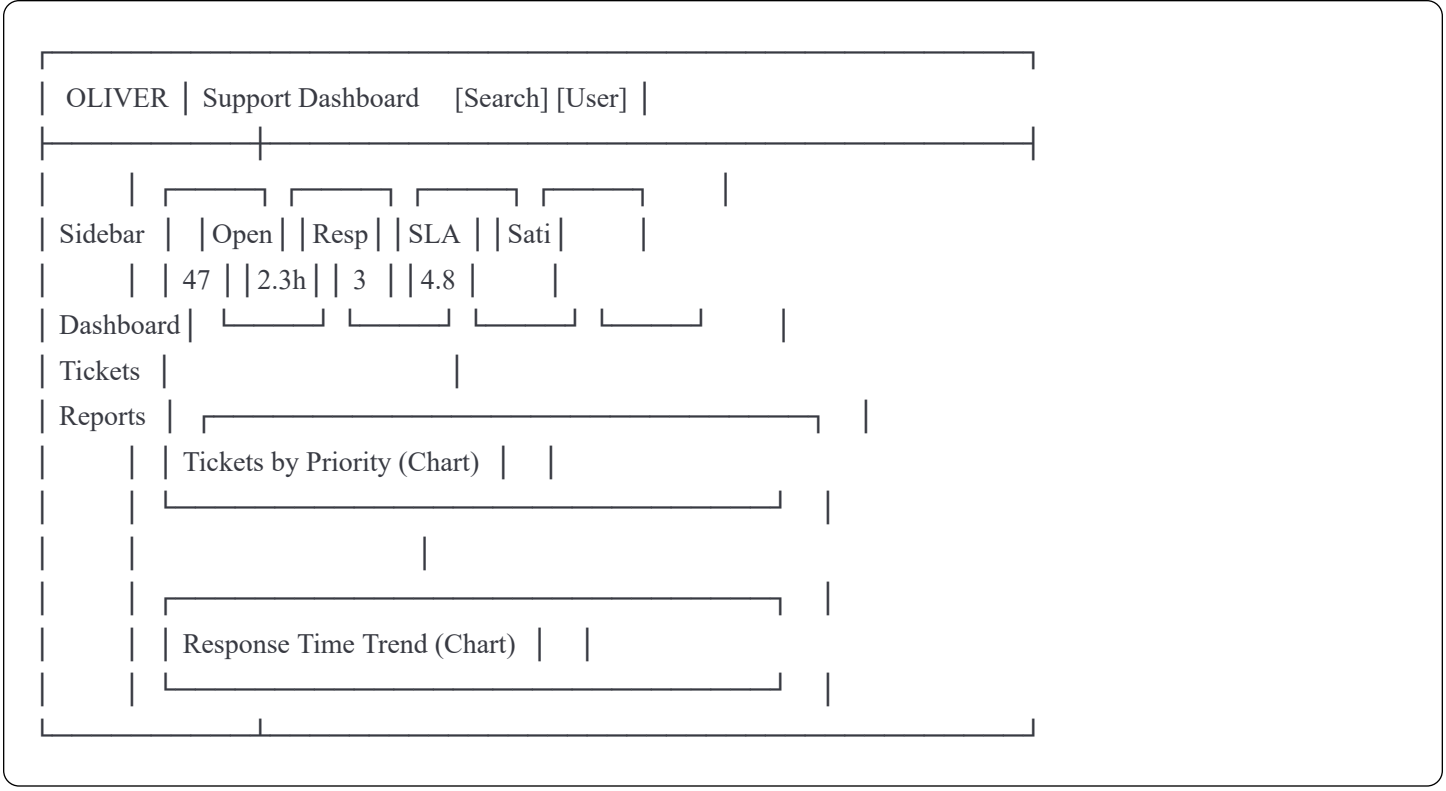
```typescript
GET /api/admin/tax-forms?status=PENDING&country=US
Response: {
  forms: {
    id: string
    creatorId: string
    creator: { name, email, avatar }
    country: string
    ytdEarnings: number
    formType: string
    status: TaxFormStatus
    documentUrl?: string
    submittedAt?: string
  }[]
}

POST /api/admin/tax-forms/:id/approve
Body: { notes?: string }
Response: { success: boolean }

POST /api/admin/tax-forms/:id/reject
Body: { reason: string, corrections: string[] }
Response: { success: boolean }

POST /api/admin/tax-forms/generate-1099
Body: { year: number, creatorIds?: string[] }
Response: {
  success: boolean
  generated: number
  downloadUrl: string
}
```

10.2 Support Tickets System (ÉCRANS 13-15)

PRIORITÉ: CRITIQUE - Sans support, frustration → churn

10.2.1 Tickets Dashboard (`/admin/support`)

Layout:



Features:

1. **Metric Cards**
   - Open Tickets: nombre total
   - Avg Response Time: en heures (vert si <2h, orange si <6h, rouge si >6h)
   - SLA Breaches: nombre (rouge si >0)
   - Satisfaction: score moyen /5 (derniers 30 jours)
2. **Charts**
   - Tickets by Priority: Pie chart (LOW, MEDIUM, HIGH, URGENT)
   - Response Time Trend: Line chart (derniers 30 jours)
   - Tickets by Category: Bar chart
   - Tickets by Source: Donut chart (Email, In-app, Chat)
3. **Quick Stats**
   - Assigned to Me: nombre
   - Unassigned: nombre
   - Waiting for User: nombre
   - Closed Today: nombre

Fichier: `apps/web/app/(admin)/support/page.tsx`

10.2.2 Tickets Queue (`/admin/support/tickets`)

Features:

1. **Filtres & Recherche**
   - Search bar: recherche dans subject + description
   - Filtres:
     - Status: Open | In Progress | Waiting User | Resolved | Closed
     - Priority: All | Low | Medium | High | Urgent
     - Category: Account | Payment | Technical | Content | Verification | Other
     - Assigned to: Me | Unassigned | Specific user
     - Date range
2. **Table Tickets**
   - Colonnes: [Checkbox] | ID | User (avatar) | Subject | Priority | Category | Status | Age | Assigned | SLA | Actions
   - SLA indicator:
     - Vert: >50% temps restant
     - Orange: 20-50% temps restant
     - Rouge: <20% temps restant
     - Rouge foncé + icon: SLA breach
   - Click sur ligne → ouvre ticket detail
3. **Bulk Actions**
   - Assign to me
   - Assign to...
   - Change priority
   - Change status
   - Close selected
4. **Sorting**
   - Par défaut: Priority DESC, Age DESC
   - Click colonnes pour changer tri

API Endpoints:

```typescript
GET /api/admin/support/tickets?status=OPEN&priority=HIGH&assignedTo=me&page=1
Response: {
  tickets: {
    id: string
    ticketNumber: string    // #12345
    userId: string
    user: { name, avatar, email }
    subject: string
    category: TicketCategory
    priority: TicketPriority
    status: TicketStatus
    source: string
    assignedTo?: string
    assignee?: { name, avatar }
    createdAt: string
    updatedAt: string
    slaDeadline?: string
    slaStatus: 'OK' | 'WARNING' | 'BREACH'
    unreadCount: number     // Messages non lus par admin
  }[]
  total: number
}

PATCH /api/admin/support/tickets/bulk-update
Body: {
  ticketIds: string[]
  updates: {
    status?: TicketStatus
    priority?: TicketPriority
    assignedTo?: string
  }
}
Response: { success: boolean, updated: number }
```

10.2.3 Ticket Detail (`/admin/support/tickets/:id`)

Layout: Split view

- Gauche: Conversation thread
- Droite: User info panel + Actions

Features:

1. **Conversation Thread**
   - Messages chronologiques
   - Avatar + nom pour chaque message
   - Timestamp
   - Distinction user vs admin messages
   - Internal notes (fond jaune, visible seulement admins)
   - Attachments preview
2. **Reply Editor**
   - Rich text editor (bold, italic, lists, links)
   - Canned responses dropdown
   - Attach files (images, PDFs)
   - Internal note checkbox
   - Buttons: Send | Save Draft
   - Auto-save draft toutes les 30s
3. **Canned Responses Library**
   - Categories: Account Issues | Payment Help | Technical Support | Verification
   - Search canned responses
   - Insert with variables: {{user.name}}, {{ticket.id}}
   - Exemples:
     - "Hello {{user.name}}, thank you for contacting us..."
     - "Your verification is currently being reviewed..."
     - "We've processed your refund of {{amount}}..."
4. **User Info Panel**
   - User details: name, email, joined date
   - Account type: Creator | Fan
   - Total spending/earnings
   - Previous tickets count
   - Quick actions: View Profile | Suspend | Reset Password
5. **Ticket Actions**
   - Change status dropdown

- Change priority dropdown
- Assign to dropdown
- Add tags
- Merge with another ticket
- Escalate to senior support
- Close ticket (avec satisfaction survey)

6. **Satisfaction Survey**
   - Envoyé automatiquement à la fermeture
   - Question: "How satisfied are you with the support?"
   - Stars 1-5
   - Optional comment

API Endpoints:

typescript

```
GET /api/admin/support/tickets/:id
Response: {
  ticket: TicketDetail
  messages: {
    id: string
    senderId: string
    sender: { name, avatar, role }
    content: string
    isInternal: boolean
    attachments: string[]
    createdAt: string
  }[]
  user: UserProfile
}

POST /api/admin/support/tickets/:id/reply
Body: {
  content: string
  isInternal: boolean
  attachments?: string[]
}
Response: { success: boolean, message: Message }

GET /api/admin/support/canned-responses?category=ACCOUNT
Response: {
  responses: {
    id: string
    title: string
    content: string
    category: string
    variables: string[]
  }[]
}

PATCH /api/admin/support/tickets/:id
Body: {
  status?: TicketStatus
  priority?: TicketPriority
  assignedTo?: string
  tags?: string[]
}
Response: { success: boolean }

POST /api/admin/support/tickets/:id/close
```

Body: { resolution: string, sendSurvey: boolean }

Response: { success: boolean }

10.3 DMCA & Legal Compliance (ÉCRANS 16-18)

PRIORITÉ: CRITIQUE - Protection légale obligatoire

10.3.1 DMCA Takedown Requests (`/admin/legal/dmca`)

Features:

1. **Public DMCA Form** (accessible sans login)
   - URL: `/dmca-takedown`
   - Champs requis (17 USC §512(c)(3)):
     - Copyright holder name
     - Contact email & address
     - Description of copyrighted work
     - Infringing URL (on Oliver platform)
     - Good faith statement checkbox
     - Accuracy statement checkbox
     - Electronic signature
     - Date

2. **DMCA Queue (Admin)**
   - Table: Request ID | Holder | Infringing URL | Received | Status | Actions
   - Status flow:
     - RECEIVED → Under Review
     - TAKEDOWN → Content Disabled
     - COUNTER_NOTICE_PERIOD (10-14 days)
     - RESOLVED / RESTORED

3. **DMCA Processing**
   - Review request validity
   - Disable content immediately (si valide)
   - Notify creator (email)
   - Start counter-notice period (14 days)
   - Track counter-notice
   - Restore content si counter-notice valide

4. **Counter-Notice Form**
   - Creator peut soumettre counter-notice
   - Requiert:
     - Identification
     - Consent to jurisdiction

- Statement under penalty of perjury
- Signature

API Endpoints:

```typescript
POST /api/dmca/submit
Body: {
  holderName: string
  contactEmail: string
  contactAddress: string
  workDescription: string
  infringingUrl: string
  goodFaithStatement: boolean
  accuracyStatement: boolean
  signature: string
}
Response: { success: boolean, requestId: string }

GET /api/admin/legal/dmca?status=RECEIVED
Response: {
  requests: {
    id: string
    holderName: string
    contactEmail: string
    workDescription: string
    infringingUrl: string
    contentId: string
    status: DmcaStatus
    receivedAt: string
    processedAt?: string
    counterNotice: boolean
    counterNoticeAt?: string
  }[]
}

POST /api/admin/legal/dmca/:id/takedown
Body: { notes?: string }
Response: { success: boolean }

POST /api/admin/legal/dmca/:id/restore
Body: { reason: string }
Response: { success: boolean }
```

10.3.2 Legal Holds (`/admin/legal/holds`)

Features:

1. **Create Legal Hold**
   - Form:
     - Case number (required, unique)
     - Jurisdiction
     - Reason/Description
     - User ID
     - Data types to freeze: Posts | Messages | Transactions | Profile | All
     - Expiration date
     - Notes
2. **Active Holds Table**
   - Colonnes: Case # | User | Jurisdiction | Created | Expires | Status | Actions
   - Status: Active | Expired | Released
   - Actions: Extend | Release | Export Data
3. **Data Freeze Implementation**
   - Mark user data as "frozen"
   - Prevent deletion by user
   - Prevent modification
   - Log all access attempts
   - Alert si user tente de supprimer
4. **Data Export for Legal**
   - Generate comprehensive package
   - Includes: All posts, messages, transactions, profile changes
   - Format: PDF report + JSON data + Media files (ZIP)
   - Password protected
   - Download link expires after 7 days

API Endpoints:

```typescript
POST /api/admin/legal/holds
Body: {
  caseNumber: string
  jurisdiction: string
  reason: string
  userId: string
  dataTypes: string[]
  expiresAt: string
  notes?: string
}
Response: { success: boolean, hold: LegalHold }

GET /api/admin/legal/holds?status=ACTIVE
Response: { holds: LegalHold[] }

POST /api/admin/legal/holds/:id/extend
Body: { newExpirationDate: string }
Response: { success: boolean }

POST /api/admin/legal/holds/:id/release
Body: { reason: string }
Response: { success: boolean }

POST /api/admin/legal/holds/:id/export
Response: {
  success: boolean
  exportId: string
  downloadUrl: string  // Signed, expires 7 days
  password: string
}
```

10.3.3 Subpoena Management (`/admin/legal/subpoenas`)

Features:

1. **Subpoena Upload**
   - Upload PDF of subpoena
   - Extract details:
     - Case number
     - Issuing authority
     - Jurisdiction
     - User(s) requested
     - Data scope
     - Response deadline
   - Assign to legal team
2. **Subpoenas Queue**
   - Table: Case # | Authority | Users | Deadline | Status | Actions
   - Deadline countdown (rouge si <7 days)
   - Status: Received | Under Review | Data Collected | Response Sent
3. **Data Collection**
   - Similar to Legal Hold export
   - Specific to subpoena scope
   - Certification of completeness
   - Chain of custody log
4. **Response Tracking**
   - Upload response document
   - Mark as sent
   - Delivery confirmation
   - Archive

API Endpoints:

```typescript
POST /api/admin/legal/subpoenas
Body: {
  caseNumber: string
  authority: string
  jurisdiction: string
  documentUrl: string
  userIds: string[]
  dataScope: string[]
  deadline: string
}
Response: { success: boolean, subpoena: Subpoena }

GET /api/admin/legal/subpoenas
Response: { subpoenas: Subpoena[] }

POST /api/admin/legal/subpoenas/:id/collect-data
Response: { success: boolean, collectionId: string }

POST /api/admin/legal/subpoenas/:id/send-response
Body: { responseDocumentUrl: string }
Response: { success: boolean }
```

10.4 Chargebacks & Disputes (ÉCRAN 19)

PRIORITÉ: HIGH - Impact financier direct

Chargeback Management (`/admin/finance/chargebacks`)

Features:

1. **Chargeback Metrics**
    - Total Chargebacks This Month
    - Win Rate: pourcentage
    - Average Chargeback Amount
    - Total Cost (chargebacks + fees)
2. **Chargebacks Queue**
    - Table: Transaction ID | User | Amount | Reason | Deadline | Status | Actions
    - Deadline countdown (rouge si <3 days)
    - Reason codes:
        - Fraudulent
        - Unrecognized
        - Duplicate
        - Not Received
        - Canceled
        - Other
3. **Evidence Collection (Automatique)**
    - Transaction details
    - User IP address
    - Device fingerprint
    - Content access logs
    - Communication history
    - Previous chargebacks by user
4. **Evidence Submission**
    - Pre-filled evidence package
    - Add additional documents
    - Submit to payment processor
    - Track submission status
5. **Chargeback Analytics**
    - Win rate by reason
    - Chargebacks by user (identify repeat offenders)

- Cost over time
- Recommendations pour prevention

API Endpoints:

```typescript
GET /api/admin/finance/chargebacks?status=PENDING
Response: {
  chargebacks: {
    id: string
    paymentId: string
    transactionId: string
    userId: string
    user: { name, email }
    amount: number
    reason: ChargebackReason
    status: ChargebackStatus
    receivedAt: string
    dueDate: string
    evidenceUrl?: string
    outcome?: string
  }[]
  metrics: {
    total: number
    winRate: number
    avgAmount: number
    totalCost: number
  }
}

POST /api/admin/finance/chargebacks/:id/submit-evidence
Body: {
  additionalDocuments?: string[]
  notes?: string
}
Response: { success: boolean }

GET /api/admin/finance/chargebacks/analytics
Response: {
  winRateByReason: { reason: string, winRate: number }[]
  topOffenders: { userId: string, count: number }[]
  costOverTime: { date: string, amount: number }[]
}
```

10.5 System Health Monitoring (ÉCRAN 20)

PRIORITÉ: HIGH - Détection proactive

System Dashboard (`/admin/system/health`)

Features:

1. **Services Status**
   - Grid de cards:
     - API (status: Operational | Degraded | Down)
     - Database (avec latency)
     - Redis Cache (avec hit rate)
     - CDN (avec bandwidth)
     - Queue Workers (avec job rate)
     - Email Service
     - Payment Processors (Stripe, CCBill)
   - Chaque card: status indicator (vert/orange/rouge) + key metric
2. **Performance Metrics**
   - Response Times:
     - p50: ligne avec valeur
     - p95: ligne avec valeur
     - p99: ligne avec valeur
   - Chart: Response time over last 24h
   - Error Rate: pourcentage (vert si <0.1%, rouge si >1%)
   - Request Volume: req/sec
3. **Infrastructure Metrics**
   - CPU Usage: gauge
   - Memory Usage: gauge
   - Disk Usage: gauge
   - Network: In/Out
4. **Active Alerts**
   - Table: Alert | Severity | Triggered | Status
   - Acknowledge button
   - Resolve button
5. **Slowest Endpoints**
   - Table: Endpoint | Avg Time | Calls | p95
   - Top 10 slowest dans dernières 24h

API Endpoints:

```typescript
GET /api/admin/system/health/status
Response: {
  services: {
    name: string
    status: 'operational' | 'degraded' | 'down'
    metrics: object
  }[]
  infrastructure: {
    cpu: number
    memory: number
    disk: number
  }
  activeAlerts: Alert[]
}

GET /api/admin/system/health/metrics?period=24h
Response: {
  responseTimes: {
    p50: number
    p95: number
    p99: number
    timeseries: { timestamp: string, value: number }[]
  }
  errorRate: number
  requestVolume: number
}

GET /api/admin/system/health/slow-endpoints
Response: {
  endpoints: {
    path: string
    method: string
    avgTime: number
    calls: number
    p95: number
  }[]
}
```

🗄️ SECTION 11: DATABASE SCHEMAS COMPLETS (ADDITIONS)

Ajouter ces modèles au fichier `packages/database/prisma/schema.prisma`:

prisma

```
// ============================================
// PAYOUT MANAGEMENT
// ============================================

model Payout {
  id              String       @id @default(cuid())
  creatorId       String
  creator         User         @relation("CreatorPayouts", fields: [creatorId], references: [id])
  amount          Decimal      @db.Decimal(10, 2)
  currency        String       @default("EUR")
  status          PayoutStatus @default(PENDING)
  method          PayoutMethod
  bankDetails     Json?        // Encrypted: { iban, bic, accountName }
  paypalEmail     String?
  cryptoAddress   String?
  requestedAt     DateTime     @default(now())
  approvedAt      DateTime?
  approvedById    String?
  approvedBy      User?        @relation("PayoutsApproved", fields: [approvedById], references: [id])
  rejectedAt      DateTime?
  rejectionReason String?
  processedAt     DateTime?
  completedAt     DateTime?
  failureReason   String?
  retryCount      Int          @default(0)
  maxRetries      Int          @default(3)
  nextRetryAt     DateTime?
  externalId      String?      // ID from payment processor
  taxFormStatus   TaxFormStatus @default(NOT_REQUIRED)
  notes           String?

  @@index([creatorId, status])
  @@index([status, requestedAt])
  @@index([nextRetryAt])
  @@map("payouts")
}

enum PayoutStatus {
  PENDING
  APPROVED
  PROCESSING
  COMPLETED
  FAILED
  REJECTED
  ON_HOLD
```

```
  ON_HOLD
}

enum PayoutMethod {
  BANK_TRANSFER
  PAYPAL
  CRYPTO
}

enum TaxFormStatus {
  NOT_REQUIRED
  REQUIRED
  SUBMITTED
  APPROVED
  REJECTED
}

model TaxForm {
  id              String       @id @default(cuid())
  creatorId       String       @unique
  creator         User         @relation("CreatorTaxForm", fields: [creatorId], references: [id])
  country         String
  formType        String       // W-9, W-8BEN, W-8BEN-E
  ytdEarnings     Decimal      @db.Decimal(10, 2)
  status          TaxFormStatus @default(NOT_REQUIRED)
  documentUrl     String?      // S3 URL
  documentKey     String?      // S3 key
  submittedAt     DateTime?
  reviewedById    String?
  reviewedBy      User?        @relation("TaxFormsReviewed", fields: [reviewedById], references: [id])
  reviewedAt      DateTime?
  approvedAt      DateTime?
  rejectedAt      DateTime?
  rejectionReason String?
  corrections     String[]     @default([])
  tin             String?      // Tax Identification Number (encrypted)
  createdAt       DateTime     @default(now())
  updatedAt       DateTime     @updatedAt

  @@index([status])
  @@index([country, status])
  @@map("tax_forms")
}

// ============================================
// SUPPORT TICKETS
// ============================================
```

```
model Ticket {
  id             String         @id @default(cuid())
  ticketNumber   String         @unique // #12345
  userId         String
  user           User           @relation("UserTickets", fields: [userId], references: [id])
  subject        String
  category       TicketCategory
  priority       TicketPriority @default(MEDIUM)
  status         TicketStatus   @default(OPEN)
  source         String         // email, in-app, chat
  assignedToId   String?
  assignedTo     User?          @relation("TicketsAssigned", fields: [assignedToId], references: [id])
  slaDeadline    DateTime?
  tags           String[]       @default([])
  createdAt      DateTime       @default(now())
  updatedAt      DateTime       @updatedAt
  closedAt       DateTime?
  closedById     String?
  closedBy       User?          @relation("TicketsClosed", fields: [closedById], references: [id])
  resolution     String?
  satisfactionScore Int?        // 1-5
  satisfactionComment String?
  messages       TicketMessage[]

  @@index([status, priority])
  @@index([assignedToId])
  @@index([userId])
  @@index([createdAt])
  @@map("tickets")
}

model TicketMessage {
  id          String   @id @default(cuid())
  ticketId    String
  ticket      Ticket   @relation(fields: [ticketId], references: [id], onDelete: Cascade)
  senderId    String
  sender      User     @relation("TicketMessages", fields: [senderId], references: [id])
  content     String   @db.Text
  isInternal  Boolean  @default(false) // Internal notes
  attachments String[] @default([])
  createdAt   DateTime @default(now())

  @@index([ticketId, createdAt])
  @@map("ticket_messages")
}
```

```
enum TicketCategory {
  ACCOUNT
  PAYMENT
  TECHNICAL
  CONTENT
  VERIFICATION
  LEGAL
  OTHER
}

enum TicketPriority {
  LOW
  MEDIUM
  HIGH
  URGENT
}

enum TicketStatus {
  OPEN
  IN_PROGRESS
  WAITING_USER
  RESOLVED
  CLOSED
}

model CannedResponse {
  id          String   @id @default(cuid())
  title       String
  content     String   @db.Text
  category    TicketCategory
  variables   String[] @default([]) // {{user.name}}, {{ticket.id}}
  usageCount  Int      @default(0)
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  @@index([category])
  @@map("canned_responses")
}

// ============================================
// LEGAL & COMPLIANCE
// ============================================

model DmcaRequest {
  id                String     @id @default(cuid())
  copyrightHolder   String
```

```
  contactEmail        String
  contactAddress      String
  workDescription     String     @db.Text
  infringingUrl       String
  goodFaithStatement  Boolean
  accuracyStatement   Boolean
  signature           String
  signatureDate       DateTime
  status              DmcaStatus @default(RECEIVED)
  contentId           String?
  contentType         String?    // POST, MESSAGE
  contentDisabledAt   DateTime?
  counterNotice       Boolean    @default(false)
  counterNoticeAt     DateTime?
  counterNoticeBy     String?
  counterNoticeReason String?
  restoredAt          DateTime?
  notes               String?
  createdAt           DateTime   @default(now())
  updatedAt           DateTime   @updatedAt

  @@index([status])
  @@index([contentId])
  @@map("dmca_requests")
}

enum DmcaStatus {
  RECEIVED
  UNDER_REVIEW
  TAKEDOWN
  COUNTER_NOTICE_PERIOD
  RESTORED
  RESOLVED
}

model LegalHold {
  id           String   @id @default(cuid())
  caseNumber   String   @unique
  jurisdiction String
  reason       String   @db.Text
  userId       String
  user         User     @relation("LegalHolds", fields: [userId], references: [id])
  dataTypes    String[] // posts, messages, transactions, profile, all
  dataSnapshot Json?    // Snapshot of data at time of hold
  createdAt    DateTime @default(now())
  createdById  String
```

```
  createdBy     User     @relation("LegalHoldsCreated", fields: [createdById], references: [id])
  expiresAt     DateTime
  extendedUntil DateTime?
  releasedAt    DateTime?
  releasedById  String?
  releasedBy    User?    @relation("LegalHoldsReleased", fields: [releasedById], references: [id])
  releaseReason String?
  exportUrl     String?  // Signed URL to data export
  notes         String?

  @@index([userId])
  @@index([expiresAt])
  @@map("legal_holds")
}

model Subpoena {
  id             String         @id @default(cuid())
  caseNumber     String         @unique
  authority      String         // FBI, Court, etc.
  jurisdiction   String
  documentUrl    String         // Uploaded subpoena PDF
  userIds        String[]
  dataScope      String[]       // What data is requested
  deadline       DateTime
  status         SubpoenaStatus @default(RECEIVED)
  assignedToId   String?
  assignedTo     User?          @relation("SubpoenasAssigned", fields: [assignedToId], references: [id])
  dataCollectedAt DateTime?
  collectionId   String?
  responseUrl    String?        // Response document
  sentAt         DateTime?
  notes          String?
  createdAt      DateTime       @default(now())

  @@index([status, deadline])
  @@index([assignedToId])
  @@map("subpoenas")
}

enum SubpoenaStatus {
  RECEIVED
  UNDER_REVIEW
  DATA_COLLECTED
  RESPONSE_SENT
  COMPLETED
}
```

```
// ==========================================
// CHARGEBACKS
// ==========================================

model Chargeback {
  id              String          @id @default(cuid())
  paymentId       String
  payment         Payment         @relation(fields: [paymentId], references: [id])
  transactionId   String          // From processor
  userId          String
  user            User            @relation("UserChargebacks", fields: [userId], references: [id])
  amount          Decimal         @db.Decimal(10, 2)
  currency        String          @default("EUR")
  reason          ChargebackReason
  reasonCode      String?
  status          ChargebackStatus @default(PENDING)
  receivedAt      DateTime        @default(now())
  dueDate         DateTime        // Deadline to respond
  evidenceUrl     String?         // Auto-collected evidence
  additionalDocs  String[]        @default([])
  submittedAt     DateTime?
  outcome         String?         // WON, LOST
  outcomeFee      Decimal?        @db.Decimal(10, 2)
  resolvedAt      DateTime?
  notes           String?

  @@index([status, dueDate])
  @@index([userId])
  @@index([paymentId])
  @@map("chargebacks")
}

enum ChargebackReason {
  FRAUDULENT
  UNRECOGNIZED
  DUPLICATE
  NOT_RECEIVED
  CANCELED
  NOT_AS_DESCRIBED
  OTHER
}

enum ChargebackStatus {
  PENDING
  EVIDENCE_REQUIRED
  SUBMITTED
```

```
  UNDER_REVIEW
  WON
  LOST
}

// ==========================================
// SYSTEM MONITORING
// ==========================================

model SystemAlert {
  id          String      @id @default(cuid())
  type        String      // cpu_high, error_rate, slow_response
  severity    AlertSeverity
  message     String
  metadata    Json
  status      AlertStatus @default(ACTIVE)
  triggeredAt DateTime    @default(now())
  acknowledgedAt DateTime?
  acknowledgedById String?
  acknowledgedBy User?    @relation("AlertsAcknowledged", fields: [acknowledgedById], references: [id])
  resolvedAt  DateTime?
  resolvedById String?
  resolvedBy  User?       @relation("AlertsResolved", fields: [resolvedById], references: [id])

  @@index([status, triggeredAt])
  @@index([severity, status])
  @@map("system_alerts")
}

enum AlertSeverity {
  INFO
  WARNING
  ERROR
  CRITICAL
}

enum AlertStatus {
  ACTIVE
  ACKNOWLEDGED
  RESOLVED
}

// ==========================================
// GDPR DATA REQUESTS
// ==========================================
```

```
model DataRequest {
  id          String            @id @default(cuid())
  userId      String
  user        User              @relation("DataRequests", fields: [userId], references: [id])
  type        DataRequestType
  status      DataRequestStatus @default(PENDING)
  requestedAt DateTime          @default(now())
  dueDate     DateTime          // 30 days from request
  processedAt DateTime?
  completedAt DateTime?
  downloadUrl String?           // Signed URL, expires 7 days
  deletedAt   DateTime?         // For deletion requests
  notes       String?

  @@index([userId])
  @@index([status, dueDate])
  @@map("data_requests")
}

// ============================================
// UPDATE EXISTING MODELS
// ============================================

model User {
  // ... existing fields ...

  // New relations
  payoutsCreator        Payout[]           @relation("CreatorPayouts")
  payoutsApprover       Payout[]           @relation("PayoutsApproved")
  taxForm               TaxForm?           @relation("CreatorTaxForm")
  taxFormsReviewed      TaxForm[]          @relation("TaxFormsReviewed")
  ticketsUser           Ticket[]           @relation("UserTickets")
  ticketsAssigned       Ticket[]           @relation("TicketsAssigned")
  ticketsClosed         Ticket[]           @relation("TicketsClosed")
  ticketMessages        TicketMessage[]    @relation("TicketMessages")
  legalHolds            LegalHold[]        @relation("LegalHolds")
  legalHoldsCreated     LegalHold[]        @relation("LegalHoldsCreated")
  legalHoldsReleased    LegalHold[]        @relation("LegalHoldsReleased")
  subpoenasAssigned     Subpoena[]         @relation("SubpoenasAssigned")
  chargebacks           Chargeback[]       @relation("UserChargebacks")
  alertsAcknowledged    SystemAlert[]      @relation("AlertsAcknowledged")
  alertsResolved        SystemAlert[]      @relation("AlertsResolved")
  dataRequests          DataRequest[]      @relation("DataRequests")
}

model Payment {
  // ... existing fields ...
```

```
  chargebacks  Chargeback[]
}
```

✅ CHECKLIST FINALE COMPLÈTE

Avant de considérer le développement terminé, **VALIDER TOUTES** ces sections:

Frontend (20 Écrans Total)

- [ ] **Base Admin (9 écrans)**: Dashboard, Users, Reports, Moderation, Transactions, Accounting, Audit Log, Settings, KYC
- [ ] **Payouts (3 écrans)**: Queue, Failed, Tax Forms
- [ ] **Support (3 écrans)**: Dashboard, Tickets Queue, Ticket Detail
- [ ] **Legal (3 écrans)**: DMCA, Legal Holds, Subpoenas
- [ ] **Finance (1 écran)**: Chargebacks
- [ ] **System (1 écran)**: Health Monitoring
- [ ] Tous pixel-perfect aux visuels fournis
- [ ] Design system cohérent partout
- [ ] Responsive parfait (mobile, tablet, desktop)
- [ ] Accessibilité WCAG AA minimum
- [ ] Loading states pour TOUTES les requêtes
- [ ] Error handling avec messages clairs français
- [ ] Toasts pour TOUTES les actions
- [ ] Confirmations pour actions destructives
- [ ] Animations fluides (transitions 150ms)
- [ ] Aucune console error ou warning
- [ ] Pagination fonctionnelle partout
- [ ] Filtres et recherche opérationnels

Backend (APIs Complètes)

- [ ] **Dashboard**: 3 endpoints (metrics, sales, transactions)
- [ ] **Users**: 5 endpoints (list, suspend, bulk-suspend, reset-password, export)
- [ ] **Reports**: 4 endpoints (list, update, assign, escalate)
- [ ] **Moderation**: 2 endpoints (queue, decision)
- [ ] **Transactions**: 3 endpoints (list, trends, detail)
- [ ] **Accounting**: 3 endpoints (summary, export, exports list)
- [ ] **Audit Log**: 2 endpoints (list, export)
- [ ] **KYC**: 5 endpoints (pending, detail, approve, reject, request-review)
- [ ] **Settings**: 3 endpoints (get, update, reset-api-key)
- [ ] **Payouts**: 6 endpoints (list, approve, reject, bulk-approve, retry, failed)
- [ ] **Tax Forms**: 4 endpoints (list, approve, reject, generate-1099)
- [ ] **Support**: 7 endpoints (dashboard, tickets, detail, reply, canned-responses, update, close)
- [ ] **DMCA**: 5 endpoints (submit, list, takedown, restore, counter-notice)
- [ ] **Legal Holds**: 5 endpoints (create, list, extend, release, export)
- [ ] **Subpoenas**: 4 endpoints (create, list, collect-data, send-response)
- [ ] **Chargebacks**: 3 endpoints (list, submit-evidence, analytics)
- [ ] **System Health**: 3 endpoints (status, metrics, slow-endpoints)
- [ ] Tous les endpoints documentés (Swagger)
- [ ] Validation stricte des inputs (Zod)
- [ ] Guards admin + permissions granulaires
- [ ] Audit log pour toutes les actions admin
- [ ] Rate limiting configuré (100 req/min par admin)
- [ ] Error handling centralisé
- [ ] Logging structuré

Database

- [ ] **Base models**: User, Post, Payment, etc. (déjà existants)
- [ ] **Admin models**: AuditLog, Report, KycVerification, ModerationDecision, ExportHistory, Settings, ModeratorSession
- [ ] **Payout models**: Payout, TaxForm
- [ ] **Support models**: Ticket, TicketMessage, CannedResponse
- [ ] **Legal models**: DmcaRequest, LegalHold, Subpoena
- [ ] **Finance models**: Chargeback
- [ ] **System models**: SystemAlert, DataRequest
- [ ] Tous les indexes créés
- [ ] Toutes les relations définies
- [ ] Migrations générées et testées
- [ ] Seed data pour testing
- [ ] Backup automatisé configuré

Sécurité

- [ ] JWT authentication robuste
- [ ] RBAC (Role-Based Access Control) implémenté
- [ ] Permissions granulaires par action
- [ ] CORS configuré strictement
- [ ] Rate limiting par endpoint et par admin
- [ ] Input sanitization PARTOUT (XSS prevention)
- [ ] SQL injection impossible (Prisma parameterized)
- [ ] CSRF protection
- [ ] Secrets dans variables d'environnement
- [ ] KYC documents chiffrés (AES-256)
- [ ] Bank details chiffrés
- [ ] Audit log immutable (append-only)
- [ ] Legal holds fonctionnels (data freeze)

Intégrations

- [ ] Yoti/Jumio webhook configuré et testé
- [ ] Google Vision API setup et testé
- [ ] AWS Rekognition setup et testé
- [ ] Email transactionnel (SendGrid) configuré
- [ ] S3 pour documents KYC configuré
- [ ] Redis pour cache et queues configuré
- [ ] Payment processors webhooks (pour chargebacks)
- [ ] Monitoring (Datadog/Sentry) configuré

Business Logic Critique

- [ ] **Payout approval flow**: Vérifications (bank details, tax form, fraud) avant approval
- [ ] **Tax form validation**: Automatic 1099 generation si $600+
- [ ] **DMCA takedown**: Immediate content disable + 14 days counter-notice period
- [ ] **Legal hold**: Data freeze fonctionnel, impossible de modifier/supprimer
- [ ] **Chargeback evidence**: Auto-collection + submission
- [ ] **Support SLA**: Countdown + alerts si breach imminent
- [ ] **Moderation protection**: Max 4h/day, breaks obligatoires
- [ ] **Ban system**: IP + device + email bans fonctionnels
- [ ] **Audit log**: Toutes actions admin loggées avec IP, device, metadata

Tests

- [ ] Unit tests pour tous les services (>80% coverage)
- [ ] Integration tests pour tous les controllers
- [ ] E2E tests pour flows critiques:
- [ ] Payout approval flow
- [ ] Support ticket lifecycle
- [ ] DMCA takedown flow
- [ ] Chargeback evidence submission
- [ ] KYC approval flow
- [ ] Load testing (>100 req/s)
- [ ] Security testing (OWASP Top 10)

Performance

- [ ] Response time <200ms (p95)
- [ ] Database queries optimisées (no N+1)
- [ ] Pagination sur toutes grandes listes
- [ ] Caching stratégique (Redis)
- [ ] CDN pour assets statiques
- [ ] Indexes database créés

Documentation

- [ ] README complet avec setup instructions
- [ ] API documentation (Swagger UI accessible)
- [ ] Architecture diagrams
- [ ] Runbook pour incidents
- [ ] Guide déploiement

DevOps

- [ ] Docker images optimisées
- [ ] CI/CD pipeline fonctionnel
- [ ] Monitoring (Datadog/Sentry) actif
- [ ] Alerting configuré (PagerDuty/Slack)
- [ ] Logs centralisés (CloudWatch/ELK)
- [ ] Health checks endpoints
- [ ] Graceful shutdown

🎯 ORDRE DE DÉVELOPPEMENT RECOMMANDÉ

Phase 1: Base Admin (Semaines 1-4)

1. Dashboard
2. Users Management
3. Reports & Flags
4. Moderation Queue
5. Transactions Overview
6. Accounting & Export
7. Audit Log
8. Settings
9. KYC Validation

Phase 2: Features Critiques (Semaines 5-7)

10. Payout Management (Queue + Failed + Tax Forms)
11. Support Tickets (Dashboard + Queue + Detail)
12. DMCA & Legal (DMCA + Holds + Subpoenas)

Phase 3: Features Importantes (Semaines 8-9)

13. Chargebacks Management
14. System Health Monitoring

Phase 4: Polish & Tests (Semaine 10)

- Tests E2E complets
- Performance optimization
- Security audit
- Documentation
- Bug fixes

TOTAL ESTIMÉ: 10 semaines pour 1-2 développeurs full-time

🚨 INSTRUCTIONS CRITIQUES POUR CLAUDE DEV

❌ CE QU'IL NE FAUT PAS FAIRE

1. **PAS D'EXEMPLES DE CODE**

```typescript
// ❌ INCORRECT
// Example implementation:
function getUsers() {
  // TODO: Implement this
  return []
}
```

2. **PAS DE PLACEHOLDERS**

```typescript
// ❌ INCORRECT
const data = mockData // Replace with real data
```

3. **PAS DE COMMENTAIRES "TODO"**

```typescript
// ❌ INCORRECT
// TODO: Add error handling
// TODO: Implement pagination
```

4. **PAS DE CODE INCOMPLET**

```typescript
// ❌ INCORRECT
async function approveKyc(id: string) {
  // Implement approval logic here
}
```

✅ CE QU'IL FAUT FAIRE

1. **CODE COMPLET ET FONCTIONNEL**

typescript

```typescript
// ✅ CORRECT
async function approveKyc(id: string, adminId: string): Promise<KycApprovalResult> {
  // Fetch KYC
  const kyc = await prisma.kycVerification.findUnique({
    where: { id },
    include: { user: true }
  })

  if (!kyc) {
    throw new NotFoundException(`KYC ${id} not found`)
  }

  if (kyc.status !== 'PENDING') {
    throw new BadRequestException(`KYC already ${kyc.status}`)
  }

  // Update KYC status
  const updated = await prisma.kycVerification.update({
    where: { id },
    data: {
      status: 'VERIFIED',
      reviewedById: adminId,
      reviewedAt: new Date(),
      approvedAt: new Date()
    }
  })

  // Update user verification status
  await prisma.user.update({
    where: { id: kyc.userId },
    data: {
      kycStatus: 'VERIFIED',
      ageVerified: true
    }
  })

  // Log action
  await auditLog.log({
    actorId: adminId,
    action: 'kyc.approved',
    targetType: 'kyc',
    targetId: id,
    metadata: { userId: kyc.userId }
  })
```

```
    // Send email to user
    await emailService.send({
      to: kyc.user.email,
      template: 'kyc-approved',
      data: { userName: kyc.user.displayName }
    })

    return {
      success: true,
      kyc: updated
    }
  }
```

2. **TOUT IMPLÉMENTER EN DÉTAIL**
   - Toutes les validations
   - Tout le error handling
   - Tous les edge cases
   - Toutes les intégrations
   - Tous les logs
   - Toutes les notifications
3. **SUIVRE LES CONVENTIONS DU PROJET**
   - Utiliser Prisma pour database
   - Utiliser Zod pour validation
   - Utiliser le design system existant
   - Suivre la structure de dossiers
4. **CODER PRODUCTION-READY**
   - Performance optimisée
   - Sécurité maximum
   - Code testé
   - Documentation inline pour code complexe

📝 FORMAT DE DÉVELOPPEMENT

Pour chaque écran/feature, développe dans cet ordre:

1. Database Schema

```prisma
// Modèle Prisma complet avec tous les champs et relations
```

2. DTOs & Validation

```typescript
// Schemas Zod pour validation des inputs
```

3. Service Layer

```typescript
// Logique métier complète avec error handling
```

4. Controller

```typescript
// Endpoints avec guards, validation, error handling
```

5. Frontend Component

```typescript
// Component React complet avec state, API calls, UI
```

6. API Integration

```typescript
// React Query hooks pour data fetching
```

🎬 COMMANDE DE LANCEMENT

Pour Claude Dev, lance cette commande:

bash

```
# Étape 1: Analyser le code existant
"Analyse le projet dans C:\dev et identifie tous les fichiers existants dont tu auras besoin"

# Étape 2: Créer l'arborescence complète
"Crée l'arborescence complète de fichiers pour toute la section admin (20 écrans + backend + database)"

# Étape 3: Développer Phase par Phase
"Développe la Phase 1 (9 écrans de base) en commençant par le Dashboard. Code COMPLET seulement, pas d'exemples."

# Puis Phase 2, 3, 4...
```

💰 BUDGET RÉALISTE PROJET COMPLET

Développement (10 semaines)

- 2 développeurs senior: **100,000€**
- Design UI/UX: **15,000€**
- Tests & QA: **20,000€**

Infrastructure

- Setup initial: **10,000€**
- Coûts mensuels: **5,000€**

Services & Intégrations

- Yoti/Jumio: **500€/mois**
- Google Vision: **200€/mois**
- SendGrid: **100€/mois**
- Monitoring: **500€/mois**

Legal & Compliance

- Audit sécurité: **15,000€**
- Conseil juridique: **10,000€**

TOTAL DÉVELOPPEMENT ADMIN: **170,000€ + 6,300€/mois opérations**

🎯 SUCCÈS = PLATEFORME PRODUCTION-READY

Une fois ce prompt complètement développé, la plateforme Oliver aura:

✅ **Interface admin complète** (20 écrans) ✅ **Gestion financière** (payouts, chargebacks, taxes) ✅ **Support utilisateurs** (tickets, SLA, satisfaction) ✅ **Conformité légale** (DMCA, legal holds, subpoenas) ✅ **Modération robuste** (AI + humain, protections) ✅ **KYC/AML** (Yoti, validation, 2257 records) ✅ **Monitoring** (système, performance, alertes) ✅ **Sécurité maximum** (audit log, bans, permissions) ✅ **Scalabilité** (architecture optimisée)

Ready pour lancement production avec vraie audience.

⚠️ RAPPEL FINAL

CLAUDE DEV: Tu dois coder **TOUT** en détail, **PAS D'EXEMPLES**.

Chaque fichier doit être:

- ✅ Complet et fonctionnel
- ✅ Production-ready
- ✅ Avec error handling
- ✅ Avec validation
- ✅ Avec logging
- ✅ Avec tests

Si tu ne peux pas coder quelque chose complètement, DEMANDE des clarifications plutôt que de mettre un placeholder.

Commence par analyser le code existant dans `C:\dev`, puis crée l'arborescence complète de fichiers, puis développe écran par écran en Phase 1, 2, 3, 4.

CODE COMPLET SEULEMENT. PAS D'EXEMPLES.

- [ ] Les 9 écrans sont identiques pixel-perfect aux visuels
- [ ] Design system cohérent (couleurs, spacing, typography)
- [ ] Responsive parfait (mobile, tablet, desktop)
- [ ] Accessibilité WCAG AA minimum
- [ ] Loading states pour toutes les requêtes
- [ ] Error handling avec messages clairs
- [ ] Toasts pour toutes les actions
- [ ] Confirmations pour actions destructives
- [ ] Animations fluides (transitions 150ms)
- [ ] Pas de console errors ou warnings

Backend

- [ ] Tous les endpoints API documentés (Swagger)
- [ ] Validation stricte des inputs (Zod)
- [ ] Guards admin + permissions granulaires
- [ ] Audit log pour toutes les actions admin
- [ ] Rate limiting configuré
- [ ] Caching Redis pour métriques
- [ ] Indexes database optimisés
- [ ] Queries optimisées (pas de N+1)
- [ ] Error handling centralisé
- [ ] Logging structuré (Winston/Pino)

Database

- [ ] Tous les modèles Prisma créés
- [ ] Migrations générées et appliquées
- [ ] Indexes créés
- [ ] Seed data pour testing
- [ ] Backup automatisé configuré

Sécurité

- [ ] JWT authentication robuste
- [ ] RBAC (Role-Based Access Control)
- [ ] CORS configuré strictement
- [ ] Rate limiting par endpoint
- [ ] Input sanitization partout
- [ ] SQL injection impossible (Prisma)
- [ ] XSS prevention
- [ ] CSRF protection
- [ ] Secrets dans variables d'environnement
- [ ] KYC documents chiffrés

Intégrations

- [ ] Yoti/Jumio webhook configuré
- [ ] Google Vision API setup
- [ ] AWS Rekognition setup
- [ ] Email transactionnel (SendGrid)
- [ ] S3 pour documents KYC
- [ ] Redis pour cache et queues

Tests

- [ ] Unit tests pour tous les services (>80% coverage)
- [ ] Integration tests pour tous les controllers
- [ ] E2E tests pour flows critiques
- [ ] Load testing (>100 req/s)
- [ ] Security testing (OWASP Top 10)

Performance

- [ ] Response time <200ms (p95)
- [ ] Database queries optimisées
- [ ] Pagination sur grandes listes
- [ ] Caching stratégique
- [ ] CDN pour assets statiques
- [ ] Image optimization

Documentation

- [ ] README complet avec setup instructions
- [ ] API documentation (Swagger)
- [ ] Architecture diagrams
- [ ] Runbook pour incidents
- [ ] Guide déploiement

DevOps

- [ ] Docker images optimisées
- [ ] CI/CD pipeline fonctionnel
- [ ] Monitoring (Datadog/Sentry)
- [ ] Alerting configuré
- [ ] Logs centralisés
- [ ] Health checks endpoints
- [ ] Graceful shutdown

🚀 COMMANDE DE LANCEMENT

Une fois **TOUT** développé, teste avec:

```bash
# 1. Setup base de données
cd packages/database
pnpm prisma migrate dev
pnpm prisma db seed

# 2. Démarrer services
docker-compose up -d  # PostgreSQL + Redis

# 3. Lancer l'API
cd apps/api
pnpm dev

# 4. Lancer le Frontend
cd apps/web
pnpm dev

# 5. Ouvrir admin
# http://localhost:3000/admin/dashboard
# Login: admin@oliver.com / Admin123!
```

📝 NOTES IMPORTANTES

1. **AUCUN placeholder ou exemple** - Tout doit être fonctionnel
2. **Production-ready** - Code prêt pour déploiement réel
3. **Error handling** - Gérer TOUS les cas d'erreur possibles
4. **Security first** - Chaque endpoint protégé et validé
5. **Performance** - Optimiser toutes les requêtes
6. **Tests** - Coverage minimum 80%
7. **Documentation** - Commenter le code complexe
8. **Consistency** - Suivre les conventions du projet existant

---

🎯 OBJECTIF FINAL

Après développement, l'admin doit permettre de:

- ✅ Gérer tous les utilisateurs de la plateforme
- ✅ Modérer les contenus avec AI assistance
- ✅ Valider les KYC avec confiance
- ✅ Suivre les revenus et transactions
- ✅ Gérer les rapports et flags
- ✅ Exporter les données comptables
- ✅ Auditer toutes les actions admin
- ✅ Configurer la plateforme

Le tout avec une interface identique aux 9 visuels fournis.

Commence par créer l'arborescence de fichiers, puis développe écran par écran, en commençant par le Dashboard. **Code complet seulement.**