ETUDE ARCHITECTURALE COMPLETE

MICROSERVICES EQUIPMENT ET BUILDING - IMMO360

TABLE DES MATIERES

1. ANALYSE DU DOMAINE METIER

1.1 CONTEXTE ACTUEL

Problematique Identifiee

L'Institut Universitaire Saint Jean (IUSJC) dispose de 3 sites (Eyang, Douala, Etokoss) avec un focus critique sur le site d'Eyang comprenant:

- 02 batiments pedagogiques/administratifs
- 01 batiment cite universitaire
- 01 batiment pour les peres

Le systeme actuel presente plusieurs deficiences majeures:

1. Gestion manuelle des incidents (papier, Excel)
2. Communication inefficace entre occupants et agents
3. Pas de tracabilite des interventions
4. Difficultde priorisation des taches
5. Absence de donnees analytiques
6. Impossibilite de prediction des pannes

Documents Analysés

D'apres les documents fournis:

stock_materiel_2025-2026.pdf:

- Inventaire de 29 categories de materiel de menage
- Quantites variables (balais, produits nettoyage, equipements)
- Gestion basique des stocks sans tracabilite

enregistrement_et_retrait.pdf:

- Suivi manuel des equipements (ordinateurs, projecteurs, materiel bureau)
- Colonnes: Reception, Designation, Livreur, Service, References, Date retrait
- Problemes: pas de lien avec les espaces, pas de suivi etat

Constats:

- Aucun lien entre equipements et espaces (chambres/bureaux)
- Pas de workflow de maintenance
- Pas de categorisation des incidents
- Absence de SLA (Service Level Agreement)

1.2 DOMAINES METIERS IDENTIFIES

Domaine 1: Gestion des Batiments

Entites principales:

- Site (Campus)
- Batiment
- Etage
- Espace (Chambre/Bureau/Salle)

Regles metier:

- Un site contient plusieurs batiments
- Un batiment contient plusieurs etages
- Un etage contient plusieurs espaces
- Chaque espace a un type (CHAMBRE, BUREAU, SALLE_CLASSE, SALLE_REUNION, etc.)
- Chaque espace a une capacite maximale d'occupants
- Certains espaces peuvent etre loues/reserves

Domaine 2: Gestion des Equipements

Entites principales:

- Categorie d'equipement
- Equipement
- Etat d'equipement

Regles metier:

- Un equipement appartient a UNE categorie
- Un equipement est assigne a UN espace specifique
- Un equipement a un etat (BON_ETAT, A_REPARER, A_REMPLACER, EN_MAINTENANCE, HORS_SERVICE)
- Les transitions d'etat sont controlees (workflow)
- Un equipement a une duree de vie estimee
- Un equipement peut avoir un historique de maintenances

Domaine 3: Gestion des Incidents

Entites principales:

- Incident
- Type d'incident
- Priorite
- Statut

Regles metier:

- Un incident est lie a UN equipement et UN espace
- Un incident a une priorite (BASSE, MOYENNE, HAUTE, CRITIQUE)
- Un incident passe par des etats (OUVERT, EN_COURS, RESOLU, FERME, REJETE)
- Un incident doit etre assigne a un agent dans un delai SLA
- La resolution d'un incident peut changer l'etat de l'equipement
- Un incident peut avoir des pieces jointes (photos)

Domaine 4: Gestion des Occupants

Entites principales:

- Occupant
- Session d'occupation
- Type d'occupant

Regles metier:

- Un occupant a une session temporelle (date debut, date fin)
- Une session expiree empeche l'acces
- Un occupant peut occuper plusieurs espaces (rare mais possible)
- Seuls les occupants actifs peuvent declarer des incidents

Domaine 5: Gestion des Interventions

Entites principales:

- Intervention
- Agent
- Rapport d'intervention

Regles metier:

- Une intervention est liee a UN incident
- Un agent peut avoir plusieurs interventions simultanees
- Une intervention a une duree estimee vs duree reelle
- La cloture d'intervention necessite un rapport
- Un agent peut etre specialise par type d'equipement

---

2. ARCHITECTURE GLOBALE DES MICROSERVICES

2.1 DECOMPOSITION EN MICROSERVICES

```
IMMO360 BACKEND
├── auth-service (4001) - EXISTANT
├── user-service (4002) - EXISTANT
├── sync-service (4003) - EXISTANT
├── api-gateway (4000) - EXISTANT
├── building-service (4004) - NOUVEAU
├── equipment-service (4005) - NOUVEAU
├── incident-service (4006) - NOUVEAU (optionnel)
└── notification-service (4007) - NOUVEAU (optionnel)
```

2.2 RESPONSABILITES DES SERVICES

building-service (Port 4004)

Responsabilites:

- Gestion des sites, batiments, etages, espaces
- Cartographie et geolocalisation indoor
- Assignation d'occupants aux espaces
- Gestion des sessions d'occupation
- Import/Export Excel des batiments
- API de recherche spatiale
- Generation de plans interactifs

Ne gere PAS:

- Les equipements (delegu a equipment-service)
- Les incidents (delegu a incident-service ou equipment-service)
- Les utilisateurs (delegu a user-service)

equipment-service (Port 4005)

Responsabilites:

- Gestion du catalogue d'equipements
- Assignation equipements <-> espaces

- Gestion des etats et transitions d'etat
- Gestion des incidents (OPTION A: integre ici)
- Historique des maintenances
- Import/Export Excel des equipements
- Predictions de pannes
- Statistiques et KPI equipements

Ne gere PAS:

- La structure des batiments (delegu a building-service)
- Les interventions agents (delegu a user-service ou service dedie)
- L'authentification (delegu a auth-service)

2.3 CHOIX ARCHITECTURAL: INCIDENT MANAGEMENT

Option A: Incidents dans equipment-service

Avantages:

- Cohesion forte (incident lie a equipement)
- Moins de microservices a maintenir
- Transactions plus simples

Inconvenients:

- Service potentiellement trop gros
- Violation du Single Responsibility Principle
- Difficile a scaler independamment

Option B: incident-service dedie

Avantages:

- Separation des concerns
- Scalabilite independante
- Workflows incidents complexes possibles
- Peut gerer incidents batiment (fuite, electricite)

Inconvenients:

- Plus de services a maintenir
- Communication inter-services plus complexe
- Transactions distribuees

RECOMMANDATION: Option A pour MVP, migration vers Option B si croissance

2.4 ARCHITECTURE HEXAGONALE

Chaque microservice suit l'architecture hexagonale:

```
/building-service
├── src/
│   ├── domain/                    # Coeur metier
│   │   ├── entities/              # Building, Floor, Space, OccupantSession
│   │   ├── value-objects/         # SpaceType, BuildingStatus, Coordinates
│   │   ├── repositories/          # Interfaces
│   │   └── ports/                 # Ports entrants/sortants
│   │
│   ├── application/               # Cas d'usage
│   │   ├── use-cases/
│   │   │   ├── create-building.use-case.ts
│   │   │   ├── assign-occupant.use-case.ts
│   │   │   ├── import-buildings.use-case.ts
│   │   │   └── generate-indoor-map.use-case.ts
│   │   ├── dto/
│   │   └── services/              # Services applicatifs
│   │
│   └── infrastructure/            # Adapters
│       ├── database/
│       │   ├── entities/          # TypeORM entities
│       │   └── repositories/      # Implementations
│       ├── http/
│       │   └── controllers/
│       ├── messaging/             # RabbitMQ
│       └── external/              # APIs externes
```

---

3. BUILDING-SERVICE - ANALYSE DETAILLEE

3.1 ENTITES DU DOMAINE

3.1.1 Site Entity

```
class SiteEntity {
  id: string;              // UUID
  name: string;            // "Eyang", "Douala", "Etokoss"
  address: string;
  city: string;
  country: string;
  coordinates: Coordinates; // Latitude, Longitude
  status: SiteStatus;      // ACTIVE, INACTIVE, MAINTENANCE
  createdAt: Date;
  updatedAt: Date;
}

enum SiteStatus {
  ACTIVE = 'ACTIVE',
  INACTIVE = 'INACTIVE',
  MAINTENANCE = 'MAINTENANCE'
}
```

3.1.2 Building Entity

```
class BuildingEntity {
  id: string;
  siteId: string;          // FK vers Site
  name: string;            // "Batiment A", "Cite U", "Batiment Peres"
  code: string;            // "BAT-A", "CITE-U", "BAT-P"
  type: BuildingType;
  floorsCount: number;
  totalCapacity: number;
  status: BuildingStatus;
  coordinates: Coordinates; // Position GPS ou indoor
  metadata: BuildingMetadata; // JSON flexible
  createdAt: Date;
  updatedAt: Date;
}

enum BuildingType {
  PEDAGOGICAL = 'PEDAGOGICAL',
  ADMINISTRATIVE = 'ADMINISTRATIVE',
  RESIDENTIAL = 'RESIDENTIAL',    // Cite U
  RELIGIOUS = 'RELIGIOUS',        // Batiment Peres
  MIXED = 'MIXED'
}

enum BuildingStatus {
  ACTIVE = 'ACTIVE',
  INACTIVE = 'INACTIVE',
  RENOVATION = 'RENOVATION',
  CONDEMNED = 'CONDEMNED'
}

interface BuildingMetadata {
  constructionYear?: number;
  lastRenovation?: Date;
  architect?: string;
  surfaceArea?: number;      // m²
  energyClass?: string;
  accessibility?: boolean;
  [key: string]: any;        // Extensible
}
```

3.1.3 Floor Entity

```
class FloorEntity {
  id: string;
  buildingId: string;
  number: number;          // 0 (RDC), 1, 2, etc.
```

```
  name: string;            // "Rez-de-chaussée", "1er étage"
  totalSpaces: number;
  status: FloorStatus;
  floorPlan?: string;      // URL vers plan SVG/PNG
  createdAt: Date;
  updatedAt: Date;
}

enum FloorStatus {
  ACTIVE = 'ACTIVE',
  INACTIVE = 'INACTIVE',
  MAINTENANCE = 'MAINTENANCE'
}
```

3.1.4 Space Entity (Chambre/Bureau/Salle)

```
class SpaceEntity {
  id: string;
  floorId: string;
  buildingId: string;      // Denormalized pour requetes rapides
  number: string;          // "101", "A12", etc.
  name: string;            // "Chambre 101", "Bureau Direction"
  type: SpaceType;
  capacity: number;        // Nombre max occupants
  surfaceArea: number;     // m²
  status: SpaceStatus;
  coordinates: IndoorCoordinates;
  features: SpaceFeatures;
  metadata: Record<string, any>;
  createdAt: Date;
  updatedAt: Date;
}

enum SpaceType {
  BEDROOM = 'BEDROOM',             // Chambre Cite U
  OFFICE = 'OFFICE',               // Bureau
  CLASSROOM = 'CLASSROOM',         // Salle de classe
  MEETING_ROOM = 'MEETING_ROOM',   // Salle de reunion
  LABORATORY = 'LABORATORY',       // Laboratoire
  LIBRARY = 'LIBRARY',             // Bibliotheque
  CAFETERIA = 'CAFETERIA',
  BATHROOM = 'BATHROOM',
  STORAGE = 'STORAGE',
  HALLWAY = 'HALLWAY',
  OTHER = 'OTHER'
}

enum SpaceStatus {
  AVAILABLE = 'AVAILABLE',
  OCCUPIED = 'OCCUPIED',
  RESERVED = 'RESERVED',
```

```
  MAINTENANCE = 'MAINTENANCE',
  OUT_OF_SERVICE = 'OUT_OF_SERVICE'
}

interface IndoorCoordinates {
  x: number;               // Position X sur le plan
  y: number;               // Position Y sur le plan
  polygon?: Point[];       // Forme de l'espace
}

interface Point {
  x: number;
  y: number;
}

interface SpaceFeatures {
  hasWindow: boolean;
  hasAC: boolean;          // Climatisation
  hasHeating: boolean;
  hasBathroom: boolean;
  hasKitchen: boolean;
  accessibility: boolean;
  internetAccess: boolean;
  [key: string]: boolean | number | string;
}
```

3.1.5 OccupantSession Entity

```
class OccupantSessionEntity {
  id: string;
  spaceId: string;
  occupantId: string;      // FK vers user-service
  startDate: Date;
  endDate: Date;
  status: SessionStatus;
  sessionType: SessionType;
  contractRef?: string;    // Reference contrat
  monthlyRent?: number;    // Loyer mensuel
  deposit?: number;        // Caution
  notes?: string;
  createdAt: Date;
  updatedAt: Date;
}

enum SessionStatus {
  ACTIVE = 'ACTIVE',
  EXPIRED = 'EXPIRED',
  CANCELLED = 'CANCELLED',
  SUSPENDED = 'SUSPENDED'
}
```

```
enum SessionType {
  STUDENT = 'STUDENT',          // Etudiant
  STAFF = 'STAFF',              // Personnel
  GUEST = 'GUEST',              // Invite
  TEMPORARY = 'TEMPORARY'
}
```

3.2 CAS D'USAGE BUILDING-SERVICE

3.2.1 Gestion des Batiments

CreateBuildingUseCase

```
Input: CreateBuildingDto {
  siteId: string;
  name: string;
  code: string;
  type: BuildingType;
  floorsCount: number;
  coordinates?: { lat: number; lng: number };
  metadata?: BuildingMetadata;
}

Output: BuildingEntity

Workflow:
1. Valider siteId existe
2. Valider code unique par site
3. Creer Building
4. Publier event: building.created
5. Retourner Building
```

ImportBuildingsFromExcelUseCase

```
Input: File (Excel)

Excel Format:
| Site | Batiment | Code | Type | Etages | Adresse |
|------|----------|------|------|--------|---------|
| Eyang| Bat A    | BAT-A| PEDAGOGICAL | 3 | ... |

Output: {
  success: boolean;
  imported: number;
  errors: string[];
}

Workflow:
1. Parser fichier Excel
```

```
2. Valider chaque ligne
3. Verifier doublons
4. Transaction: creer tous batiments
5. Publier events building.created (batch)
6. Retourner rapport
```

3.2.2 Gestion des Espaces

CreateSpaceUseCase

```
Input: CreateSpaceDto {
  floorId: string;
  number: string;
  name?: string;
  type: SpaceType;
  capacity: number;
  surfaceArea?: number;
  features?: SpaceFeatures;
  coordinates?: IndoorCoordinates;
}

Output: SpaceEntity

Workflow:
1. Valider floorId existe
2. Valider number unique dans floor
3. Calculer buildingId depuis floor
4. Creer Space avec status AVAILABLE
5. Publier event: space.created
6. Retourner Space
```

AssignEquipmentToSpaceUseCase (coordination avec equipment-service)

```
Input: {
  spaceId: string;
  equipmentId: string;
}

Output: Success/Failure

Workflow:
1. Valider spaceId existe
2. Appeler equipment-service.assignToSpace(equipmentId, spaceId)
3. Si succes, publier event: space.equipment.assigned
4. Retourner resultat
```

3.2.3 Gestion des Occupants

AssignOccupantToSpaceUseCase

```
Input: AssignOccupantDto {
  spaceId: string;
  occupantId: string;
  startDate: Date;
  endDate: Date;
  sessionType: SessionType;
  monthlyRent?: number;
  deposit?: number;
}

Output: OccupantSessionEntity

Validations:
- Space existe et est AVAILABLE
- Occupant existe (appel user-service)
- Pas de chevauchement sessions
- endDate > startDate
- Capacite space non depassee

Workflow:
1. Valider toutes les regles
2. Creer OccupantSession avec status ACTIVE
3. Mettre a jour Space.status = OCCUPIED
4. Publier event: occupant.assigned
5. Envoyer notification a l'occupant (via notification-service)
6. Retourner session
```

ExpireOccupantSessionJob (Cron Job)

```
Frequence: Quotidienne a minuit

Workflow:
1. Trouver sessions avec endDate < now() et status ACTIVE
2. Pour chaque session:
   a. Mettre status = EXPIRED
   b. Mettre Space.status = AVAILABLE
   c. Publier event: session.expired
   d. Notifier occupant
3. Logger resultats
```

3.2.4 Geolocalisation Indoor

GenerateIndoorMapUseCase

```
Input: {
  buildingId: string;
```

```
  floorNumber: number;
}

Output: {
  svg: string;          // Plan SVG genere
  spaces: SpaceMapData[];
}

interface SpaceMapData {
  id: string;
  number: string;
  type: SpaceType;
  status: SpaceStatus;
  coordinates: IndoorCoordinates;
  occupancyRate: number;
  equipmentCount: number;
  incidentCount: number;
}

Workflow:
1. Charger Floor et tous Spaces
2. Appeler equipment-service.getEquipmentCountsBySpace()
3. Appeler equipment-service.getIncidentCountsBySpace()
4. Generer SVG avec librairie (d3.js, svg.js)
5. Colorer espaces selon status
6. Ajouter legende
7. Retourner SVG + metadata
```

SearchNearbySpacesUseCase

```
Input: {
  spaceId: string;
  radius: number;       // metres
  filters?: {
    type?: SpaceType[];
    status?: SpaceStatus[];
    hasFeatures?: string[];
  }
}

Output: SpaceEntity[]

Workflow:
1. Charger Space de reference
2. Calculer espaces dans rayon (geospatial query)
3. Appliquer filtres
4. Trier par distance
5. Retourner liste
```

3.3 REPOSITORIES BUILDING-SERVICE

```
interface IBuildingRepository {
  save(building: BuildingEntity): Promise<BuildingEntity>;
  findById(id: string): Promise<BuildingEntity | null>;
  findBySiteId(siteId: string): Promise<BuildingEntity[]>;
  findByCode(code: string, siteId: string): Promise<BuildingEntity | null>;
  findAll(filters?: BuildingFilters): Promise<BuildingEntity[]>;
  update(id: string, data: Partial<BuildingEntity>): Promise<BuildingEntity>;
  delete(id: string): Promise<void>;
  count(filters?: BuildingFilters): Promise<number>;
}

interface ISpaceRepository {
  save(space: SpaceEntity): Promise<SpaceEntity>;
  findById(id: string): Promise<SpaceEntity | null>;
  findByFloorId(floorId: string): Promise<SpaceEntity[]>;
  findByBuildingId(buildingId: string): Promise<SpaceEntity[]>;
  findByNumber(number: string, floorId: string): Promise<SpaceEntity | null>;
  findAvailable(filters?: SpaceFilters): Promise<SpaceEntity[]>;
  findByOccupant(occupantId: string): Promise<SpaceEntity[]>;
  updateStatus(id: string, status: SpaceStatus): Promise<void>;
  findNearby(coordinates: IndoorCoordinates, radius: number):
Promise<SpaceEntity[]>;
}

interface IOccupantSessionRepository {
  save(session: OccupantSessionEntity): Promise<OccupantSessionEntity>;
  findById(id: string): Promise<OccupantSessionEntity | null>;
  findActiveByOccupant(occupantId: string): Promise<OccupantSessionEntity[]>;
  findActiveBySpace(spaceId: string): Promise<OccupantSessionEntity[]>;
  findExpired(): Promise<OccupantSessionEntity[]>;
  expireSession(id: string): Promise<void>;
  findOverlapping(spaceId: string, startDate: Date, endDate: Date):
Promise<OccupantSessionEntity[]>;
}
```

3.4 EVENTS BUILDING-SERVICE

```
// Events publies
building.created
building.updated
building.deleted
floor.created
floor.updated
space.created
space.updated
space.status.changed
occupant.assigned
occupant.removed
session.expired
session.cancelled
```

```
// Events ecoutes
user.deleted -> Supprimer sessions occupant
equipment.assigned -> Mettre a jour metadata space
incident.created -> Incrementer compteur incidents space
```

---

4. EQUIPMENT-SERVICE - ANALYSE DETAILLEE

4.1 ENTITES DU DOMAINE

4.1.1 EquipmentCategory Entity

```typescript
class EquipmentCategoryEntity {
  id: string;
  name: string;            // "Mobilier", "Electronique", "Climatisation"
  code: string;            // "MOB", "ELEC", "CLIM"
  description?: string;
  parentCategoryId?: string; // Hierarchie categories
  icon?: string;           // URL icone
  defaultLifespan?: number; // Duree vie en mois
  maintenanceInterval?: number; // Intervalle maintenance (jours)
  criticality: Criticality;
  createdAt: Date;
  updatedAt: Date;
}

enum Criticality {
  LOW = 'LOW',
  MEDIUM = 'MEDIUM',
  HIGH = 'HIGH',
  CRITICAL = 'CRITICAL'
}
```

4.1.2 Equipment Entity

```typescript
class EquipmentEntity {
  id: string;
  categoryId: string;
  spaceId?: string;        // Peut etre null (en stock)
  buildingId?: string;     // Denormalized
  name: string;            // "Climatiseur Samsung 12000 BTU"
  code: string;            // "CLIM-001"
  serialNumber?: string;
  brand?: string;
  model?: string;
  purchaseDate?: Date;
  purchasePrice?: number;
```

```
  warrantyEndDate?: Date;
  lifespan?: number;       // Mois
  status: EquipmentStatus;
  condition: EquipmentCondition;
  location: EquipmentLocation;
  lastMaintenanceDate?: Date;
  nextMaintenanceDate?: Date;
  metadata: EquipmentMetadata;
  createdAt: Date;
  updatedAt: Date;
}

enum EquipmentStatus {
  IN_STOCK = 'IN_STOCK',          // En stock
  ASSIGNED = 'ASSIGNED',          // Assigne a un espace
  IN_USE = 'IN_USE',              // En utilisation
  MAINTENANCE = 'MAINTENANCE',    // En maintenance
  OUT_OF_SERVICE = 'OUT_OF_SERVICE', // Hors service
  DISPOSED = 'DISPOSED'           // Mis au rebut
}

enum EquipmentCondition {
  EXCELLENT = 'EXCELLENT',     // Excellent etat
  GOOD = 'GOOD',              // Bon etat
  FAIR = 'FAIR',              // Etat moyen
  POOR = 'POOR',              // Mauvais etat
  TO_REPAIR = 'TO_REPAIR',    // A reparer
  TO_REPLACE = 'TO_REPLACE',  // A remplacer
  BROKEN = 'BROKEN'           // Casse
}

enum EquipmentLocation {
  SPACE = 'SPACE',            // Dans un espace
  STORAGE = 'STORAGE',        // En stockage
  WORKSHOP = 'WORKSHOP',      // Atelier reparation
  EXTERNAL = 'EXTERNAL'       // Chez fournisseur
}

interface EquipmentMetadata {
  specifications?: Record<string, any>;
  manuals?: string[];         // URLs manuels
  photos?: string[];
  qrCode?: string;
  dimensions?: { w: number; h: number; d: number };
  weight?: number;
  powerConsumption?: number;  // Watts
  [key: string]: any;
}
```

4.1.3 Incident Entity

```
class IncidentEntity {
  id: string;
  equipmentId: string;
  spaceId: string;
  buildingId: string;      // Denormalized
  reportedBy: string;      // User ID (occupant ou agent)
  assignedTo?: string;     // Agent ID
  title: string;
  description: string;
  priority: IncidentPriority;
  status: IncidentStatus;
  category: IncidentCategory;
  photos: string[];        // URLs photos
  reportedAt: Date;
  acknowledgedAt?: Date;
  startedAt?: Date;
  resolvedAt?: Date;
  closedAt?: Date;
  resolution?: string;
  estimatedDuration?: number; // minutes
  actualDuration?: number;
  slaDeadline?: Date;
  metadata: IncidentMetadata;
  createdAt: Date;
  updatedAt: Date;
}

enum IncidentPriority {
  LOW = 'LOW',
  MEDIUM = 'MEDIUM',
  HIGH = 'HIGH',
  CRITICAL = 'CRITICAL'
}

enum IncidentStatus {
  OPEN = 'OPEN',              // Ouvert
  ACKNOWLEDGED = 'ACKNOWLEDGED', // Pris en compte
  ASSIGNED = 'ASSIGNED',      // Assigne
  IN_PROGRESS = 'IN_PROGRESS', // En cours
  RESOLVED = 'RESOLVED',      // Resolu
  CLOSED = 'CLOSED',          // Clos
  REJECTED = 'REJECTED',      // Rejete
  CANCELLED = 'CANCELLED'     // Annule
}

enum IncidentCategory {
  BREAKDOWN = 'BREAKDOWN',        // Panne
  MALFUNCTION = 'MALFUNCTION',    // Dysfonctionnement
  DAMAGE = 'DAMAGE',              // Degradation
  MAINTENANCE = 'MAINTENANCE',    // Maintenance preventive
  INSTALLATION = 'INSTALLATION', // Installation
  REMOVAL = 'REMOVAL',            // Retrait
  OTHER = 'OTHER'
```

```
}

interface IncidentMetadata {
  affectedUsers?: number;
  workaround?: string;
  spareParts?: string[];
  cost?: number;
  vendor?: string;
  [key: string]: any;
}
```

4.1.4 MaintenanceHistory Entity

```
class MaintenanceHistoryEntity {
  id: string;
  equipmentId: string;
  incidentId?: string;     // Si lie a un incident
  type: MaintenanceType;
  description: string;
  performedBy: string;     // Agent ID
  performedAt: Date;
  duration: number;        // minutes
  cost?: number;
  spareParts?: SparePart[];
  beforeCondition: EquipmentCondition;
  afterCondition: EquipmentCondition;
  notes?: string;
  photos?: string[];
  createdAt: Date;
}

enum MaintenanceType {
  PREVENTIVE = 'PREVENTIVE',
  CORRECTIVE = 'CORRECTIVE',
  PREDICTIVE = 'PREDICTIVE',
  EMERGENCY = 'EMERGENCY'
}

interface SparePart {
  name: string;
  reference?: string;
  quantity: number;
  unitPrice?: number;
}
```

4.2 CAS D'USAGE EQUIPMENT-SERVICE

4.2.1 Gestion des Equipements

CreateEquipmentUseCase

```
Input: CreateEquipmentDto {
  categoryId: string;
  spaceId?: string;
  name: string;
  code: string;
  serialNumber?: string;
  brand?: string;
  model?: string;
  purchaseDate?: Date;
  metadata?: EquipmentMetadata;
}

Output: EquipmentEntity

Workflow:
1. Valider categoryId existe
2. Valider code unique
3. Si spaceId fourni, valider space existe
4. Creer Equipment avec status IN_STOCK ou ASSIGNED
5. Calculer nextMaintenanceDate
6. Publier event: equipment.created
7. Si assigned, publier equipment.assigned
8. Retourner Equipment
```

ImportEquipmentsFromExcelUseCase

```
Input: File (Excel)

Excel Format:
| Categorie | Nom | Code | Marque | Modele | Espace | Etat |
|-----------|-----|------|--------|--------|--------|------|
| Mobilier  | Lit | LIT-001 | IKEA | MALM | 101 | Bon |

Output: {
  success: boolean;
  imported: number;
  errors: string[];
  warnings: string[];
}

Workflow:
1. Parser fichier Excel
2. Valider categories existent (creer si absent?)
3. Valider codes uniques
4. Resoudre espaces par numero
5. Transaction: creer tous equipements
6. Publier events equipment.created (batch)
```

```
7. Generer mot de passe temporaire pour chaque
8. Retourner rapport detaille
```

AssignEquipmentToSpaceUseCase

```
Input: {
  equipmentId: string;
  spaceId: string;
}

Output: EquipmentEntity

Validations:
- Equipment existe et status != DISPOSED
- Space existe et status = AVAILABLE ou OCCUPIED
- Equipment pas deja assigne

Workflow:
1. Charger Equipment
2. Valider Space existe (appel building-service)
3. Mettre a jour Equipment:
   - spaceId = spaceId
   - buildingId = space.buildingId
   - status = ASSIGNED
   - location = SPACE
4. Publier event: equipment.assigned
5. Retourner Equipment
```

4.2.2 Gestion des Incidents

CreateIncidentUseCase

```
Input: CreateIncidentDto {
  equipmentId: string;
  spaceId: string;
  title: string;
  description: string;
  priority?: IncidentPriority;
  category: IncidentCategory;
  photos?: File[];
}
reportedBy: string (from JWT)

Output: IncidentEntity

Workflow:
1. Valider equipmentId existe
2. Valider spaceId existe
3. Valider reportedBy a acces (session active)
```

```
4. Upload photos vers S3/storage
5. Calculer priority auto si non fournie (selon category + equipment criticality)
6. Calculer SLA deadline selon priority
7. Creer Incident avec status OPEN
8. Mettre Equipment.condition = TO_REPAIR (si pertinent)
9. Publier event: incident.created
10. Notifier agents disponibles (push notification)
11. Retourner Incident
```

AssignIncidentToAgentUseCase

```
Input: {
  incidentId: string;
  agentId: string;
}

Output: IncidentEntity

Validations:
- Incident existe et status = OPEN ou ACKNOWLEDGED
- Agent existe et actif (appel user-service)
- Agent pas surcharg (max 5 incidents EN_COURS)

Workflow:
1. Charger Incident
2. Valider Agent disponible
3. Mettre a jour Incident:
   - assignedTo = agentId
   - status = ASSIGNED
   - acknowledgedAt = now()
4. Publier event: incident.assigned
5. Notifier agent (push + email)
6. Notifier reporter
7. Retourner Incident
```

ResolveIncidentUseCase

```
Input: {
  incidentId: string;
  resolution: string;
  newEquipmentCondition: EquipmentCondition;
  spareParts?: SparePart[];
  cost?: number;
  photos?: File[];
}
agentId: string (from JWT)

Output: IncidentEntity
```

```
Validations:
- Incident existe et status = IN_PROGRESS
- Agent = assignedTo

Workflow:
1. Charger Incident
2. Upload photos
3. Calculer actualDuration = now() - startedAt
4. Mettre a jour Incident:
   - status = RESOLVED
   - resolution = resolution
   - resolvedAt = now()
   - actualDuration = calculatedDuration
5. Mettre a jour Equipment.condition
6. Creer MaintenanceHistory
7. Si newCondition = GOOD, mettre Equipment.status = IN_USE
8. Publier event: incident.resolved
9. Notifier reporter (push + email)
10. Retourner Incident
```

CloseIncidentUseCase (par reporter)

```
Input: {
  incidentId: string;
  satisfied: boolean;
  feedback?: string;
  rating?: number; // 1-5
}
userId: string (from JWT)

Output: IncidentEntity

Validations:
- Incident.status = RESOLVED
- userId = reportedBy

Workflow:
1. Charger Incident
2. Mettre a jour:
   - status = CLOSED
   - closedAt = now()
   - metadata.satisfied = satisfied
   - metadata.feedback = feedback
   - metadata.rating = rating
3. Publier event: incident.closed
4. Mettre a jour stats agent (si rating fourni)
5. Retourner Incident
```

4.2.3 Analytics et Predictions

GetEquipmentStatisticsUseCase

```
Input: {
  period: Period;          // DAILY, WEEKLY, MONTHLY, YEARLY
  startDate?: Date;
  endDate?: Date;
  buildingId?: string;
  categoryId?: string;
}

Output: {
  totalEquipments: number;
  byCondition: Record<EquipmentCondition, number>;
  byStatus: Record<EquipmentStatus, number>;
  averageAge: number;      // mois
  upcomingMaintenances: number;
  overdueMaintenances: number;
  totalIncidents: number;
  resolvedIncidents: number;
  averageResolutionTime: number; // heures
  mostProblematicSpaces: SpaceIncidentStats[];
  mostProblematicCategories: CategoryIncidentStats[];
}

Workflow:
1. Construire requetes aggregation
2. Appliquer filtres
3. Calculer metriques
4. Trier top listes
5. Retourner statistiques
```

PredictEquipmentFailureUseCase (Machine Learning)

```
Input: {
  equipmentId?: string;
  categoryId?: string;
  buildingId?: string;
  threshold?: number;      // Probabilite minimum (0.7 par defaut)
}

Output: {
  predictions: EquipmentPrediction[];
}

interface EquipmentPrediction {
  equipmentId: string;
  equipmentName: string;
  spaceId: string;
  spaceNumber: string;
  probability: number;     // 0-1
```

```
  estimatedFailureDate: Date;
  factors: PredictionFactor[];
  recommendation: string;
}

interface PredictionFactor {
  factor: string;          // "age", "incident_frequency", "last_maintenance"
  weight: number;          // 0-1
  value: any;
}

Workflow:
1. Charger historique maintenances
2. Charger historique incidents
3. Calculer features:
   - Age equipement
   - Frequence incidents
   - Temps depuis derniere maintenance
   - Condition actuelle
   - Pattern saisonnier
4. Appliquer modele ML (regression logistique ou RF)
5. Filtrer predictions > threshold
6. Generer recommandations
7. Publier event: prediction.generated
8. Retourner predictions
```

GenerateMaintenanceScheduleUseCase

```
Input: {
  buildingId?: string;
  categoryId?: string;
  startDate: Date;
  endDate: Date;
}

Output: {
  schedule: MaintenanceTask[];
}

interface MaintenanceTask {
  equipmentId: string;
  equipmentName: string;
  spaceId: string;
  type: MaintenanceType;
  scheduledDate: Date;
  estimatedDuration: number;
  priority: number;
  notes: string;
}

Workflow:
1. Charger equipements avec nextMaintenanceDate dans periode
```

```
2. Charger predictions pannes
3. Combiner maintenances preventives + predictives
4. Optimiser planning (eviter conflits espaces)
5. Assigner priorites
6. Generer taches
7. Publier event: schedule.generated
8. Retourner schedule
```

4.3 REPOSITORIES EQUIPMENT-SERVICE

```
interface IEquipmentRepository {
  save(equipment: EquipmentEntity): Promise<EquipmentEntity>;
  findById(id: string): Promise<EquipmentEntity | null>;
  findByCode(code: string): Promise<EquipmentEntity | null>;
  findBySpaceId(spaceId: string): Promise<EquipmentEntity[]>;
  findByBuildingId(buildingId: string): Promise<EquipmentEntity[]>;
  findByCategoryId(categoryId: string): Promise<EquipmentEntity[]>;
  findByCondition(condition: EquipmentCondition): Promise<EquipmentEntity[]>;
  findUpcomingMaintenance(days: number): Promise<EquipmentEntity[]>;
  findOverdueMaintenance(): Promise<EquipmentEntity[]>;
  updateCondition(id: string, condition: EquipmentCondition): Promise<void>;
  updateStatus(id: string, status: EquipmentStatus): Promise<void>;
  count(filters?: EquipmentFilters): Promise<number>;
}

interface IIncidentRepository {
  save(incident: IncidentEntity): Promise<IncidentEntity>;
  findById(id: string): Promise<IncidentEntity | null>;
  findByEquipmentId(equipmentId: string): Promise<IncidentEntity[]>;
  findBySpaceId(spaceId: string): Promise<IncidentEntity[]>;
  findByReporter(userId: string): Promise<IncidentEntity[]>;
  findByAgent(agentId: string): Promise<IncidentEntity[]>;
  findByStatus(status: IncidentStatus): Promise<IncidentEntity[]>;
  findByPriority(priority: IncidentPriority): Promise<IncidentEntity[]>;
  findOverdue(): Promise<IncidentEntity[]>;
  findOpenIncidents(): Promise<IncidentEntity[]>;
  updateStatus(id: string, status: IncidentStatus): Promise<void>;
  assignToAgent(id: string, agentId: string): Promise<void>;
  count(filters?: IncidentFilters): Promise<number>;
  getStatistics(filters: StatisticsFilters): Promise<IncidentStatistics>;
}

interface IMaintenanceHistoryRepository {
  save(history: MaintenanceHistoryEntity): Promise<MaintenanceHistoryEntity>;
  findByEquipmentId(equipmentId: string): Promise<MaintenanceHistoryEntity[]>;
  findByAgent(agentId: string): Promise<MaintenanceHistoryEntity[]>;
  findByPeriod(startDate: Date, endDate: Date):
Promise<MaintenanceHistoryEntity[]>;
  getEquipmentMaintenanceStats(equipmentId: string): Promise<MaintenanceStats>;
}
```

4.4 EVENTS EQUIPMENT-SERVICE

```
// Events publies
equipment.created
equipment.updated
equipment.assigned
equipment.removed
equipment.condition.changed
equipment.maintenance.due
incident.created
incident.assigned
incident.status.changed
incident.resolved
incident.closed
maintenance.scheduled
maintenance.completed
prediction.generated

// Events ecoutes
space.deleted -> Mettre equipments.spaceId = null
user.deleted -> Reassigner incidents
building.deleted -> Supprimer equipements
```

---

5. MODELE DE DONNEES COMPLET

5.1 SCHEMA PostgreSQL - BUILDING-SERVICE

```
-- Database: immo360_building

CREATE TABLE sites (
  id VARCHAR(255) PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  address TEXT,
  city VARCHAR(100),
  country VARCHAR(100),
  latitude DECIMAL(10, 8),
  longitude DECIMAL(11, 8),
  status VARCHAR(50) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE buildings (
  id VARCHAR(255) PRIMARY KEY,
  site_id VARCHAR(255) NOT NULL REFERENCES sites(id) ON DELETE CASCADE,
  name VARCHAR(255) NOT NULL,
  code VARCHAR(50) NOT NULL,
  type VARCHAR(50) NOT NULL,
  floors_count INTEGER NOT NULL,
```

```
  total_capacity INTEGER,
  status VARCHAR(50) NOT NULL,
  latitude DECIMAL(10, 8),
  longitude DECIMAL(11, 8),
  metadata JSONB,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE(code, site_id)
);

CREATE INDEX idx_buildings_site_id ON buildings(site_id);
CREATE INDEX idx_buildings_status ON buildings(status);
CREATE INDEX idx_buildings_type ON buildings(type);

CREATE TABLE floors (
  id VARCHAR(255) PRIMARY KEY,
  building_id VARCHAR(255) NOT NULL REFERENCES buildings(id) ON DELETE CASCADE,
  number INTEGER NOT NULL,
  name VARCHAR(255),
  total_spaces INTEGER DEFAULT 0,
  status VARCHAR(50) NOT NULL,
  floor_plan TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE(building_id, number)
);

CREATE INDEX idx_floors_building_id ON floors(building_id);

CREATE TABLE spaces (
  id VARCHAR(255) PRIMARY KEY,
  floor_id VARCHAR(255) NOT NULL REFERENCES floors(id) ON DELETE CASCADE,
  building_id VARCHAR(255) NOT NULL REFERENCES buildings(id) ON DELETE CASCADE,
  number VARCHAR(50) NOT NULL,
  name VARCHAR(255),
  type VARCHAR(50) NOT NULL,
  capacity INTEGER NOT NULL,
  surface_area DECIMAL(10, 2),
  status VARCHAR(50) NOT NULL,
  coordinates JSONB,
  features JSONB,
  metadata JSONB,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE(floor_id, number)
);

CREATE INDEX idx_spaces_floor_id ON spaces(floor_id);
CREATE INDEX idx_spaces_building_id ON spaces(building_id);
CREATE INDEX idx_spaces_type ON spaces(type);
CREATE INDEX idx_spaces_status ON spaces(status);
CREATE INDEX idx_spaces_coordinates ON spaces USING GIN (coordinates);

CREATE TABLE occupant_sessions (
```

```
  id VARCHAR(255) PRIMARY KEY,
  space_id VARCHAR(255) NOT NULL REFERENCES spaces(id) ON DELETE CASCADE,
  occupant_id VARCHAR(255) NOT NULL, -- FK vers user-service (soft)
  start_date DATE NOT NULL,
  end_date DATE NOT NULL,
  status VARCHAR(50) NOT NULL,
  session_type VARCHAR(50) NOT NULL,
  contract_ref VARCHAR(100),
  monthly_rent DECIMAL(10, 2),
  deposit DECIMAL(10, 2),
  notes TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  CHECK (end_date > start_date)
);

CREATE INDEX idx_sessions_space_id ON occupant_sessions(space_id);
CREATE INDEX idx_sessions_occupant_id ON occupant_sessions(occupant_id);
CREATE INDEX idx_sessions_status ON occupant_sessions(status);
CREATE INDEX idx_sessions_dates ON occupant_sessions(start_date, end_date);
```

5.2 SCHEMA PostgreSQL - EQUIPMENT-SERVICE

```
-- Database: immo360_equipment

CREATE TABLE equipment_categories (
  id VARCHAR(255) PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  code VARCHAR(50) NOT NULL UNIQUE,
  description TEXT,
  parent_category_id VARCHAR(255) REFERENCES equipment_categories(id),
  icon TEXT,
  default_lifespan INTEGER,
  maintenance_interval INTEGER,
  criticality VARCHAR(50) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_categories_parent ON equipment_categories(parent_category_id);

CREATE TABLE equipments (
  id VARCHAR(255) PRIMARY KEY,
  category_id VARCHAR(255) NOT NULL REFERENCES equipment_categories(id),
  space_id VARCHAR(255), -- Soft FK vers building-service
  building_id VARCHAR(255), -- Denormalized
  name VARCHAR(255) NOT NULL,
  code VARCHAR(100) NOT NULL UNIQUE,
  serial_number VARCHAR(100),
  brand VARCHAR(100),
  model VARCHAR(100),
```

```
  purchase_date DATE,
  purchase_price DECIMAL(10, 2),
  warranty_end_date DATE,
  lifespan INTEGER,
  status VARCHAR(50) NOT NULL,
  condition VARCHAR(50) NOT NULL,
  location VARCHAR(50) NOT NULL,
  last_maintenance_date DATE,
  next_maintenance_date DATE,
  metadata JSONB,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_equipments_category_id ON equipments(category_id);
CREATE INDEX idx_equipments_space_id ON equipments(space_id);
CREATE INDEX idx_equipments_building_id ON equipments(building_id);
CREATE INDEX idx_equipments_status ON equipments(status);
CREATE INDEX idx_equipments_condition ON equipments(condition);
CREATE INDEX idx_equipments_next_maintenance ON equipments(next_maintenance_date);

CREATE TABLE incidents (
  id VARCHAR(255) PRIMARY KEY,
  equipment_id VARCHAR(255) NOT NULL REFERENCES equipments(id),
  space_id VARCHAR(255) NOT NULL, -- Soft FK
  building_id VARCHAR(255) NOT NULL, -- Soft FK
  reported_by VARCHAR(255) NOT NULL, -- Soft FK vers user
  assigned_to VARCHAR(255), -- Soft FK vers user
  title VARCHAR(255) NOT NULL,
  description TEXT NOT NULL,
  priority VARCHAR(50) NOT NULL,
  status VARCHAR(50) NOT NULL,
  category VARCHAR(50) NOT NULL,
  photos JSONB,
  reported_at TIMESTAMP NOT NULL,
  acknowledged_at TIMESTAMP,
  started_at TIMESTAMP,
  resolved_at TIMESTAMP,
  closed_at TIMESTAMP,
  resolution TEXT,
  estimated_duration INTEGER,
  actual_duration INTEGER,
  sla_deadline TIMESTAMP,
  metadata JSONB,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_incidents_equipment_id ON incidents(equipment_id);
CREATE INDEX idx_incidents_space_id ON incidents(space_id);
CREATE INDEX idx_incidents_building_id ON incidents(building_id);
CREATE INDEX idx_incidents_reported_by ON incidents(reported_by);
CREATE INDEX idx_incidents_assigned_to ON incidents(assigned_to);
CREATE INDEX idx_incidents_status ON incidents(status);
```

```sql
CREATE INDEX idx_incidents_priority ON incidents(priority);
CREATE INDEX idx_incidents_sla ON incidents(sla_deadline);
CREATE INDEX idx_incidents_dates ON incidents(reported_at, resolved_at);

CREATE TABLE maintenance_history (
  id VARCHAR(255) PRIMARY KEY,
  equipment_id VARCHAR(255) NOT NULL REFERENCES equipments(id),
  incident_id VARCHAR(255) REFERENCES incidents(id),
  type VARCHAR(50) NOT NULL,
  description TEXT NOT NULL,
  performed_by VARCHAR(255) NOT NULL, -- Soft FK
  performed_at TIMESTAMP NOT NULL,
  duration INTEGER NOT NULL,
  cost DECIMAL(10, 2),
  spare_parts JSONB,
  before_condition VARCHAR(50) NOT NULL,
  after_condition VARCHAR(50) NOT NULL,
  notes TEXT,
  photos JSONB,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_maintenance_equipment_id ON maintenance_history(equipment_id);
CREATE INDEX idx_maintenance_incident_id ON maintenance_history(incident_id);
CREATE INDEX idx_maintenance_performed_by ON maintenance_history(performed_by);
CREATE INDEX idx_maintenance_date ON maintenance_history(performed_at);
```

---

6. DESIGN PATTERNS APPLICABLES

6.1 DOMAIN-DRIVEN DESIGN (DDD)

Bounded Contexts

```
Building Context:
- Aggregates: Building (root), Floor, Space
- Value Objects: Coordinates, Address, BuildingMetadata
- Domain Events: BuildingCreated, SpaceOccupied, SessionExpired

Equipment Context:
- Aggregates: Equipment (root), IncidentWorkflow
- Value Objects: EquipmentCondition, IncidentPriority
- Domain Events: EquipmentAssigned, IncidentCreated, IncidentResolved

Shared Kernel:
- User (reference)
- SpaceId (reference)
```

Entities vs Value Objects

Entities (ont une identite):

- Building, Floor, Space, Equipment, Incident

Value Objects (sans identite, immuables):

- Coordinates, Address, Money, Duration, SLAPolicy

Domain Services

```
// building-service
class SpaceAvailabilityService {
  isSpaceAvailable(spaceId: string, startDate: Date, endDate: Date): boolean {
    // Logique complexe verification disponibilite
  }
}

// equipment-service
class SLACalculationService {
  calculateDeadline(priority: IncidentPriority, reportedAt: Date): Date {
    const slaHours = {
      CRITICAL: 2,
      HIGH: 8,
      MEDIUM: 24,
      LOW: 72
    };
    return addHours(reportedAt, slaHours[priority]);
  }
}

class IncidentPriorityCalculator {
  calculate(equipment: Equipment, category: IncidentCategory): IncidentPriority {
    // Logique: equipement critique + panne = CRITICAL
    // equipement normal + dysfonctionnement = MEDIUM
  }
}
```

6.2 REPOSITORY PATTERN

Deja implemente dans l'architecture hexagonale.

Avantages:

- Abstraction de la persistence
- Testabilite (mock repositories)
- Changement DB transparent
- Separation concerns

6.3 SPECIFICATION PATTERN

Pour les requetes complexes avec filtres dynamiques:

```
// building-service
interface ISpecification<T> {
  isSatisfiedBy(candidate: T): boolean;
  and(other: ISpecification<T>): ISpecification<T>;
  or(other: ISpecification<T>): ISpecification<T>;
}

class AvailableSpaceSpecification implements ISpecification<SpaceEntity> {
  isSatisfiedBy(space: SpaceEntity): boolean {
    return space.status === SpaceStatus.AVAILABLE;
  }
}

class SpaceTypeSpecification implements ISpecification<SpaceEntity> {
  constructor(private type: SpaceType) {}

  isSatisfiedBy(space: SpaceEntity): boolean {
    return space.type === this.type;
  }
}

// Utilisation:
const bedroomSpec = new SpaceTypeSpecification(SpaceType.BEDROOM);
const availableSpec = new AvailableSpaceSpecification();
const combinedSpec = bedroomSpec.and(availableSpec);

const spaces = await spaceRepo.findBySpecification(combinedSpec);
```

6.4 STRATEGY PATTERN

Pour les algorithmes de calcul variables:

```
// equipment-service
interface IMaintenanceStrategy {
  calculateNextMaintenance(equipment: Equipment): Date;
}

class TimeBasedMaintenanceStrategy implements IMaintenanceStrategy {
  calculateNextMaintenance(equipment: Equipment): Date {
    const interval = equipment.category.maintenanceInterval || 90;
    return addDays(equipment.lastMaintenanceDate || equipment.createdAt,
interval);
  }
}

class UsageBasedMaintenanceStrategy implements IMaintenanceStrategy {
  calculateNextMaintenance(equipment: Equipment): Date {
    // Basé sur nombre d'heures utilisation
    const usageHours = equipment.metadata.totalUsageHours || 0;
    const maxHours = 1000;
```

```
      if (usageHours >= maxHours) {
        return new Date(); // Maintenance immediate
      }
      // ...
    }
  }

  class ConditionBasedMaintenanceStrategy implements IMaintenanceStrategy {
    calculateNextMaintenance(equipment: Equipment): Date {
      // Basé sur etat actuel
      if (equipment.condition === EquipmentCondition.POOR) {
        return addDays(new Date(), 7);
      }
      // ...
    }
  }

  // Utilisation:
  class MaintenanceScheduler {
    constructor(private strategy: IMaintenanceStrategy) {}

    schedule(equipment: Equipment): Date {
      return this.strategy.calculateNextMaintenance(equipment);
    }
  }
```

6.5 OBSERVER PATTERN (Event-Driven)

Deja implemente via RabbitMQ.

Publisher-Subscriber:

```
  // equipment-service
  class IncidentEventPublisher {
    async publish(event: DomainEvent): Promise<void> {
      await this.messagePublisher.publish(event.type, event.data);
    }
  }

  // Subscribers dans notification-service
  class IncidentCreatedSubscriber {
    async handle(event: IncidentCreatedEvent): Promise<void> {
      // Notifier agents disponibles
      await this.notificationService.notifyAgents(event.data);
    }
  }
```

6.6 FACTORY PATTERN

Pour creation d'objets complexes:

```
// equipment-service
class IncidentFactory {
  create(dto: CreateIncidentDto, reportedBy: string): IncidentEntity {
    const incident = new IncidentEntity(
      uuidv4(),
      dto.equipmentId,
      dto.spaceId,
      dto.buildingId,
      reportedBy
    );

    incident.title = dto.title;
    incident.description = dto.description;
    incident.category = dto.category;
    incident.photos = dto.photos || [];
    incident.reportedAt = new Date();

    // Calculer priority auto
    incident.priority = this.priorityCalculator.calculate(
      dto.equipment,
      dto.category
    );

    // Calculer SLA
    incident.slaDeadline = this.slaService.calculateDeadline(
      incident.priority,
      incident.reportedAt
    );

    incident.status = IncidentStatus.OPEN;

    return incident;
  }
}
```

6.7 SAGA PATTERN (Transactions Distribuees)

Pour operations multi-services:

```
// Scenario: Assigner occupant a un espace
// Implique: building-service + user-service + notification-service

class AssignOccupantSaga {
  async execute(dto: AssignOccupantDto): Promise<void> {
    const sagaId = uuidv4();

    try {
      // Etape 1: Verifier espace disponible (building-service)
      const space = await this.buildingService.getSpace(dto.spaceId);
      if (space.status !== SpaceStatus.AVAILABLE) {
```

```
        throw new Error('Space not available');
      }

      // Etape 2: Verifier occupant existe (user-service)
      const occupant = await this.userService.getUser(dto.occupantId);

      // Etape 3: Creer session (building-service)
      const session = await this.sessionRepo.save({
        spaceId: dto.spaceId,
        occupantId: dto.occupantId,
        startDate: dto.startDate,
        endDate: dto.endDate,
        status: SessionStatus.ACTIVE
      });

      // Etape 4: Mettre a jour space (building-service)
      await this.spaceRepo.updateStatus(dto.spaceId, SpaceStatus.OCCUPIED);

      // Etape 5: Envoyer notification (notification-service)
      await this.notificationService.sendWelcome(occupant.email, space);

      // Publier event de succes
      await this.eventPublisher.publish('saga.assign_occupant.completed', {
        sagaId,
        sessionId: session.id
      });

    } catch (error) {
      // Compensation (rollback)
      await this.compensate(sagaId);
      throw error;
    }
  }

  private async compensate(sagaId: string): Promise<void> {
    // Annuler operations precedentes
    // Publier event saga.assign_occupant.failed
  }
}
```

6.8 CQRS PATTERN (Command Query Responsibility Segregation)

Separation lecture/ecriture pour performance:

```
// equipment-service

// WRITE SIDE (Commands)
class CreateEquipmentCommand {
  constructor(public dto: CreateEquipmentDto) {}
}
```

```
class CreateEquipmentHandler {
  async handle(command: CreateEquipmentCommand): Promise<string> {
    const equipment = this.factory.create(command.dto);
    await this.writeRepo.save(equipment);
    await this.eventBus.publish(new EquipmentCreatedEvent(equipment.id));
    return equipment.id;
  }
}

// READ SIDE (Queries)
class GetEquipmentStatisticsQuery {
  constructor(public filters: StatisticsFilters) {}
}

class GetEquipmentStatisticsHandler {
  async handle(query: GetEquipmentStatisticsQuery): Promise<Statistics> {
    // Lire depuis read model optimise (vue materialisee)
    return await this.readRepo.getStatistics(query.filters);
  }
}

// Read Model (Vue materialisee SQL)
CREATE MATERIALIZED VIEW equipment_statistics AS
SELECT
  building_id,
  COUNT(*) as total,
  COUNT(*) FILTER (WHERE condition = 'GOOD') as good_condition,
  COUNT(*) FILTER (WHERE condition = 'TO_REPAIR') as to_repair,
  AVG(EXTRACT(YEAR FROM AGE(NOW(), purchase_date))) as avg_age
FROM equipments
GROUP BY building_id;

-- Refresh periodique (cron job)
REFRESH MATERIALIZED VIEW equipment_statistics;
```

---

7. FLUX DE DONNEES ET EVENEMENTS

7.1 FLUX: CREATION INCIDENT PAR OCCUPANT

```
Mobile App (Occupant)
  |
  | 1. POST /incidents
  | { equipmentId, description, photo }
  |
  v
API Gateway (4000)
  |
  | 2. Verify JWT
  | Check session active
  |
```

```
  v
Equipment-Service (4005)
  |
  | 3. CreateIncidentUseCase
  |    - Validate equipment exists
  |    - Upload photo to S3
  |    - Calculate priority
  |    - Calculate SLA deadline
  |    - Save incident
  |
  | 4. Publish RabbitMQ
  |    Event: incident.created
  |
  v
RabbitMQ
  |
  |----> Notification-Service
  |      |
  |      | 5. Send push to available agents
  |      | 6. Send email to supervisor
  |
  |----> Sync-Service
  |      |
  |      | 7. Log event in history
  |
  |----> Equipment-Service (self)
         |
         | 8. Update equipment condition
         | 9. Trigger ML prediction refresh
```

7.2 FLUX: ASSIGNATION INCIDENT A AGENT

```
Mobile App (Agent)
  |
  | 1. POST /incidents/{id}/assign
  |
  v
Equipment-Service
  |
  | 2. AssignIncidentToAgentUseCase
  |    - Validate agent capacity
  |    - Update incident
  |    - Set status = ASSIGNED
  |
  | 3. Publish: incident.assigned
  |
  v
RabbitMQ
  |
  |----> Notification-Service
  |      |
```

```
  |      | Push to agent: "Vous avez un incident"
  |      | Push to reporter: "Pris en charge par Agent X"
  |
  |----> User-Service
         |
         | Update agent stats
```

7.3 FLUX: EXPIRATION SESSION OCCUPANT (CRON)

```
Cron Job (daily 00:00)
  |
  v
Building-Service
  |
  | 1. ExpireOccupantSessionJob
  |    - Find sessions with endDate < now()
  |    - For each:
  |      * Set status = EXPIRED
  |      * Update space status = AVAILABLE
  |      * Publish: session.expired
  |
  v
RabbitMQ
  |
  |----> Notification-Service
  |      |
  |      | Email to occupant: "Session terminee"
  |
  |----> Equipment-Service
  |      |
  |      | Close open incidents for this occupant
  |
  |----> Auth-Service
         |
         | Revoke access tokens
```

7.4 FLUX: IMPORT EXCEL EQUIPEMENTS

```
Web App (Admin)
  |
  | 1. POST /equipments/import/upload
  | multipart/form-data
  |
  v
Equipment-Service
  |
  | 2. ImportEquipmentsFromExcelUseCase
  |    - Parse Excel
  |    - Validate all rows
```

```
    |    - Resolve space IDs
    |    |
    |    | 3. Call Building-Service
    |    |    GET /spaces?number={number}
    |    v
    | Building-Service returns spaceId
    |    |
    |    | 4. Transaction: Create all equipments
    |    | 5. Publish: equipment.created (batch)
    |
    v
  RabbitMQ
    |
    |----> Sync-Service
    |      |
    |      | Log import history
    |
    |----> Notification-Service
           |
           | Email report to admin
```

---

8. INTEGRATIONS INTER-SERVICES

8.1 COMMUNICATION SYNCHRONE (HTTP)

building-service appelle user-service:

```typescript
// building-service/src/infrastructure/external/user-service.client.ts

@Injectable()
export class UserServiceClient {
  constructor(private httpService: HttpService) {}

  async getUser(userId: string): Promise<UserDto> {
    const response = await this.httpService
      .get(`http://user-service:4002/users/${userId}`)
      .toPromise();

    if (!response.data) {
      throw new NotFoundException('User not found');
    }

    return response.data;
  }

  async validateOccupant(userId: string): Promise<boolean> {
    try {
      const user = await this.getUser(userId);
      return user.role === UserRole.OCCUPANT && user.status === UserStatus.ACTIVE;
    } catch {
```

```
      return false;
    }
  }
}
```

equipment-service appelle building-service:

```
// equipment-service/src/infrastructure/external/building-service.client.ts

@Injectable()
export class BuildingServiceClient {
  async getSpace(spaceId: string): Promise<SpaceDto> {
    const response = await this.httpService
      .get(`http://building-service:4004/spaces/${spaceId}`)
      .toPromise();

    return response.data;
  }

  async getSpaceByNumber(buildingId: string, number: string): Promise<SpaceDto> {
    const response = await this.httpService
      .get(`http://building-service:4004/spaces`, {
        params: { buildingId, number }
      })
      .toPromise();

    return response.data[0];
  }
}
```

8.2 COMMUNICATION ASYNCHRONE (RabbitMQ)

building-service publie events:

```
// building-service/src/infrastructure/messaging/rabbitmq.publisher.ts

@Injectable()
export class RabbitMQPublisher implements IMessagePublisher {
  async publish(eventType: string, data: any): Promise<void> {
    await this.channel.publish('building.events', eventType, {
      type: eventType,
      data,
      timestamp: new Date().toISOString(),
      source: 'building-service'
    });
  }
}

// Usage dans use-case
```

```
  await this.messagePublisher.publish('space.occupied', {
    spaceId: session.spaceId,
    occupantId: session.occupantId,
    startDate: session.startDate,
    endDate: session.endDate
  });
```

equipment-service ecoute events:

```
// equipment-service/src/infrastructure/messaging/rabbitmq.consumer.ts

@Injectable()
export class RabbitMQConsumer {
  async consumeEvents() {
    this.channel.consume('building.events', async (msg) => {
      const event = JSON.parse(msg.content.toString());

      switch (event.type) {
        case 'space.deleted':
          await this.handleSpaceDeleted(event.data);
          break;
        case 'occupant.removed':
          await this.handleOccupantRemoved(event.data);
          break;
      }

      this.channel.ack(msg);
    });
  }

  private async handleSpaceDeleted(data: any) {
    // Mettre equipments.spaceId = null
    await this.equipmentRepo.unassignFromSpace(data.spaceId);
  }
}
```

8.3 RESILIENCE ET PATTERNS

Circuit Breaker Pattern

```
// equipment-service
import * as CircuitBreaker from 'opossum';

@Injectable()
export class ResilientBuildingServiceClient {
  private breaker: CircuitBreaker;

  constructor(private buildingClient: BuildingServiceClient) {
    this.breaker = new CircuitBreaker(
```

```
      this.buildingClient.getSpace.bind(this.buildingClient),
      {
        timeout: 3000,           // 3s timeout
        errorThresholdPercentage: 50,
        resetTimeout: 30000      // 30s avant retry
      }
    );

    this.breaker.fallback(() => {
      return { id: 'unknown', number: 'N/A', type: 'UNKNOWN' };
    });
  }

  async getSpace(spaceId: string): Promise<SpaceDto> {
    return await this.breaker.fire(spaceId);
  }
}
```

Retry Pattern

```
import { retry } from 'rxjs/operators';

async getSpaceWithRetry(spaceId: string): Promise<SpaceDto> {
  return this.httpService
    .get(`http://building-service:4004/spaces/${spaceId}`)
    .pipe(
      retry({
        count: 3,
        delay: (error, retryCount) => {
          return timer(retryCount * 1000); // Backoff exponentiel
        }
      })
    )
    .toPromise();
}
```

---

9. ASPECTS TECHNIQUES AVANCES

9.1 CACHING STRATEGY

Redis Caching dans API Gateway

```
// api-gateway/src/cache/cache.service.ts

@Injectable()
export class CacheService {
  constructor(@Inject('REDIS') private redis: Redis) {}
```

```
  async get<T>(key: string): Promise<T | null> {
    const cached = await this.redis.get(key);
    return cached ? JSON.parse(cached) : null;
  }

  async set(key: string, value: any, ttl: number): Promise<void> {
    await this.redis.setex(key, ttl, JSON.stringify(value));
  }

  async invalidatePattern(pattern: string): Promise<void> {
    const keys = await this.redis.keys(pattern);
    if (keys.length > 0) {
      await this.redis.del(...keys);
    }
  }
}

// Cache Strategy par endpoint
const CACHE_TTL = {
  'GET /buildings': 600,      // 10 min
  'GET /spaces': 300,         // 5 min
  'GET /equipments': 300,     // 5 min
  'GET /incidents': 60,       // 1 min
  'GET /statistics': 900      // 15 min
};

// Invalidation sur mutations
@Post('/equipments')
async createEquipment(@Body() dto: CreateEquipmentDto) {
  const equipment = await this.equipmentService.create(dto);

  // Invalider caches associes
  await this.cacheService.invalidatePattern('GET:/equipments*');
  await this.cacheService.invalidatePattern('GET:/statistics*');

  return equipment;
}
```

Cache Local (In-Memory) pour References

```
// building-service
import { LRUCache } from 'lru-cache';

@Injectable()
export class SpaceTypeCache {
  private cache = new LRUCache<string, SpaceType>({
    max: 1000,
    ttl: 1000 * 60 * 60 // 1 hour
  });
```

```
  get(spaceId: string): SpaceType | undefined {
    return this.cache.get(spaceId);
  }

  set(spaceId: string, type: SpaceType): void {
    this.cache.set(spaceId, type);
  }
}
```

9.2 UPLOAD ET STOCKAGE FICHIERS

S3/MinIO pour Photos

```
// equipment-service/src/infrastructure/storage/s3.service.ts

@Injectable()
export class S3StorageService {
  private s3: S3Client;

  constructor() {
    this.s3 = new S3Client({
      endpoint: process.env.S3_ENDPOINT,
      region: process.env.S3_REGION,
      credentials: {
        accessKeyId: process.env.S3_ACCESS_KEY,
        secretAccessKey: process.env.S3_SECRET_KEY
      }
    });
  }

  async uploadIncidentPhoto(
    incidentId: string,
    file: Express.Multer.File
  ): Promise<string> {
    const key = `incidents/${incidentId}/${Date.now()}-${file.originalname}`;

    await this.s3.send(new PutObjectCommand({
      Bucket: process.env.S3_BUCKET,
      Key: key,
      Body: file.buffer,
      ContentType: file.mimetype,
      ACL: 'public-read'
    }));

    return `${process.env.S3_PUBLIC_URL}/${key}`;
  }

  async deletePhoto(url: string): Promise<void> {
    const key = url.replace(`${process.env.S3_PUBLIC_URL}/`, '');
    await this.s3.send(new DeleteObjectCommand({
      Bucket: process.env.S3_BUCKET,
```

```
      Key: key
    }));
  }
}
```

9.3 INDEXATION ET RECHERCHE

Elasticsearch pour Recherche Full-Text

```
// equipment-service/src/infrastructure/search/elasticsearch.service.ts

@Injectable()
export class EquipmentSearchService {
  private client: Client;

  constructor() {
    this.client = new Client({
      node: process.env.ELASTICSEARCH_URL
    });
  }

  async indexEquipment(equipment: EquipmentEntity): Promise<void> {
    await this.client.index({
      index: 'equipments',
      id: equipment.id,
      body: {
        name: equipment.name,
        code: equipment.code,
        brand: equipment.brand,
        model: equipment.model,
        categoryName: equipment.category.name,
        spaceNumber: equipment.space?.number,
        buildingName: equipment.building?.name,
        status: equipment.status,
        condition: equipment.condition,
        createdAt: equipment.createdAt
      }
    });
  }

  async search(query: string, filters?: any): Promise<EquipmentEntity[]> {
    const result = await this.client.search({
      index: 'equipments',
      body: {
        query: {
          bool: {
            must: [
              {
                multi_match: {
                  query: query,
                  fields: ['name^2', 'code^2', 'brand', 'model', 'categoryName']
```

```
                }
              }
            ],
            filter: this.buildFilters(filters)
          }
        }
      }
    });

    return result.hits.hits.map(hit => hit._source as EquipmentEntity);
  }
}
```

9.4 PAGINATION OPTIMISEE

Cursor-Based Pagination

```
// Plus performant que offset-based pour grandes tables

interface CursorPaginationParams {
  limit: number;
  cursor?: string;  // Base64 encoded
}

interface CursorPaginationResult<T> {
  data: T[];
  nextCursor?: string;
  hasMore: boolean;
}

@Injectable()
export class EquipmentRepository {
  async findWithCursor(
    params: CursorPaginationParams
  ): Promise<CursorPaginationResult<EquipmentEntity>> {
    const limit = params.limit + 1; // +1 pour detecter hasMore

    let query = this.repo.createQueryBuilder('equipment');

    if (params.cursor) {
      const decoded = this.decodeCursor(params.cursor);
      query = query.where('equipment.createdAt < :cursor', { cursor: decoded });
    }

    query = query
      .orderBy('equipment.createdAt', 'DESC')
      .limit(limit);

    const results = await query.getMany();
    const hasMore = results.length > params.limit;
```

```
    if (hasMore) {
      results.pop(); // Retirer le dernier
    }

    const nextCursor = hasMore && results.length > 0
      ? this.encodeCursor(results[results.length - 1].createdAt)
      : undefined;

    return {
      data: results,
      nextCursor,
      hasMore
    };
  }

  private encodeCursor(date: Date): string {
    return Buffer.from(date.toISOString()).toString('base64');
  }

  private decodeCursor(cursor: string): Date {
    return new Date(Buffer.from(cursor, 'base64').toString());
  }
}
```

---

10. SCALABILITE ET PERFORMANCE

10.1 HORIZONTAL SCALING

Load Balancing avec Nginx

```
# nginx.conf
upstream building_service {
  least_conn;
  server building-service-1:4004;
  server building-service-2:4004;
  server building-service-3:4004;
}

upstream equipment_service {
  least_conn;
  server equipment-service-1:4005;
  server equipment-service-2:4005;
  server equipment-service-3:4005;
}

server {
  listen 80;

  location /buildings {
    proxy_pass http://building_service;
```

```
    proxy_set_header X-Real-IP $remote_addr;
  }

  location /equipments {
    proxy_pass http://equipment_service;
    proxy_set_header X-Real-IP $remote_addr;
  }
}
```

Docker Compose avec replicas

```yaml
# docker-compose.yml
version: '3.8'

services:
  building-service:
    image: immo360/building-service:latest
    deploy:
      replicas: 3
      resources:
        limits:
          cpus: '0.5'
          memory: 512M
    environment:
      - DATABASE_URL=postgresql://user:pass@postgres:5432/immo360_building
      - REDIS_URL=redis://redis:6379
      - RABBITMQ_URL=amqp://rabbitmq:5672
    networks:
      - immo360-network

  equipment-service:
    image: immo360/equipment-service:latest
    deploy:
      replicas: 3
      resources:
        limits:
          cpus: '0.5'
          memory: 512M
    environment:
      - DATABASE_URL=postgresql://user:pass@postgres:5432/immo360_equipment
    networks:
      - immo360-network
```

10.2 DATABASE OPTIMIZATION

Indexation Strategique

```
-- building-service
CREATE INDEX CONCURRENTLY idx_spaces_compound
ON spaces(building_id, status, type);

CREATE INDEX CONCURRENTLY idx_sessions_active
ON occupant_sessions(status, end_date)
WHERE status = 'ACTIVE';

-- equipment-service
CREATE INDEX CONCURRENTLY idx_incidents_open
ON incidents(status, priority, sla_deadline)
WHERE status IN ('OPEN', 'ASSIGNED', 'IN_PROGRESS');

CREATE INDEX CONCURRENTLY idx_equipments_maintenance
ON equipments(next_maintenance_date)
WHERE next_maintenance_date IS NOT NULL;
```

Partitionnement Tables

```
-- Partitionner incidents par mois
CREATE TABLE incidents (
  id VARCHAR(255),
  reported_at TIMESTAMP NOT NULL,
  -- autres colonnes
) PARTITION BY RANGE (reported_at);

CREATE TABLE incidents_2025_01 PARTITION OF incidents
  FOR VALUES FROM ('2025-01-01') TO ('2025-02-01');

CREATE TABLE incidents_2025_02 PARTITION OF incidents
  FOR VALUES FROM ('2025-02-01') TO ('2025-03-01');

-- Auto-creation partitions futures (cron job)
```

Connection Pooling

```
// database.config.ts
export const databaseConfig = {
  type: 'postgres',
  host: process.env.DB_HOST,
  port: parseInt(process.env.DB_PORT),
  username: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  entities: [__dirname + '/../**/*.entity{.ts,.js}'],
  synchronize: false,
  logging: false,
  extra: {
```

```
    max: 20,              // Max connections
    min: 5,               // Min connections
    idleTimeoutMillis: 30000,
    connectionTimeoutMillis: 2000,
  }
};
```

10.3 RATE LIMITING

```
// api-gateway/src/rate-limit/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 userId = request.user?.id || request.ip;
    const endpoint = `${request.method}:${request.route.path}`;

    const key = `rate-limit:${userId}:${endpoint}`;
    const limit = this.getLimitForEndpoint(endpoint);
    const window = 60; // 1 minute

    const current = await this.redis.incr(key);

    if (current === 1) {
      await this.redis.expire(key, window);
    }

    if (current > limit) {
      throw new TooManyRequestsException('Rate limit exceeded');
    }

    return true;
  }

  private getLimitForEndpoint(endpoint: string): number {
    const limits = {
      'POST:/incidents': 10,
      'POST:/equipments': 20,
      'GET:/equipments': 100,
      'GET:/statistics': 30,
    };
    return limits[endpoint] || 60;
  }
}
```

---

11. GESTION HORS-LIGNE (OFFLINE-FIRST)

11.1 ARCHITECTURE MOBILE OFFLINE

Strategy: Sync Queue

```
// Mobile App (React Native / Flutter)

class OfflineManager {
  private syncQueue: SyncOperation[] = [];
  private isOnline: boolean = true;

  constructor() {
    // Ecouter connectivite
    NetInfo.addEventListener(state => {
      this.isOnline = state.isConnected;
      if (this.isOnline) {
        this.processQueue();
      }
    });
  }

  async createIncident(data: CreateIncidentDto): Promise<string> {
    const localId = uuidv4();

    // Sauvegarder localement
    await this.localDB.incidents.add({
      id: localId,
      ...data,
      status: 'PENDING_SYNC',
      createdAt: new Date()
    });

    if (this.isOnline) {
      try {
        const result = await this.api.createIncident(data);
        await this.localDB.incidents.update(localId, {
          id: result.id,
          status: 'SYNCED'
        });
        return result.id;
      } catch (error) {
        this.addToSyncQueue({
          type: 'CREATE_INCIDENT',
          localId,
          data
        });
      }
    } else {
      this.addToSyncQueue({
        type: 'CREATE_INCIDENT',
        localId,
        data
      });
```

```
    }

    return localId;
  }

  private async processQueue() {
    while (this.syncQueue.length > 0 && this.isOnline) {
      const operation = this.syncQueue[0];

      try {
        await this.executeOperation(operation);
        this.syncQueue.shift();
      } catch (error) {
        // Retry plus tard
        await delay(5000);
      }
    }
  }

  private async executeOperation(op: SyncOperation) {
    switch (op.type) {
      case 'CREATE_INCIDENT':
        const result = await this.api.createIncident(op.data);
        await this.localDB.incidents.update(op.localId, {
          id: result.id,
          status: 'SYNCED'
        });
        break;
      // Autres operations
    }
  }
}
```

Local Storage avec IndexedDB/SQLite

```
// Mobile App

class LocalDatabase {
  private db: Dexie;

  constructor() {
    this.db = new Dexie('IMMO360');
    this.db.version(1).stores({
      incidents: 'id, equipmentId, spaceId, status, createdAt',
      equipments: 'id, spaceId, condition, status',
      spaces: 'id, buildingId, number, status'
    });
  }

  async getIncidentsBySpace(spaceId: string): Promise<Incident[]> {
    return await this.db.incidents
```

```
      .where('spaceId').equals(spaceId)
      .toArray();
  }

  async getOfflineChanges(): Promise<SyncOperation[]> {
    const incidents = await this.db.incidents
      .where('status').equals('PENDING_SYNC')
      .toArray();

    return incidents.map(i => ({
      type: 'CREATE_INCIDENT',
      localId: i.id,
      data: i
    }));
  }
}
```

11.2 CONFLICT RESOLUTION

Last-Write-Wins Strategy

```
// equipment-service

@Injectable()
export class ConflictResolver {
  async resolveIncidentConflict(
    serverVersion: IncidentEntity,
    clientVersion: IncidentEntity
  ): Promise<IncidentEntity> {
    // Strategy: Last write wins
    if (clientVersion.updatedAt > serverVersion.updatedAt) {
      return clientVersion;
    }
    return serverVersion;
  }
}
```

Operational Transformation pour Texte

```
// Pour description/commentaires simultanes

class TextOT {
  transform(op1: Operation, op2: Operation): Operation {
    // Algorithm OT
    // Gerer insertions/suppressions concurrentes
  }
}
```

12. PUSH NOTIFICATIONS

12.1 ARCHITECTURE NOTIFICATIONS

Firebase Cloud Messaging (FCM)

```
// notification-service/src/fcm/fcm.service.ts

@Injectable()
export class FCMService {
  private admin: admin.app.App;

  constructor() {
    this.admin = admin.initializeApp({
      credential: admin.credential.cert({
        projectId: process.env.FCM_PROJECT_ID,
        privateKey: process.env.FCM_PRIVATE_KEY,
        clientEmail: process.env.FCM_CLIENT_EMAIL
      })
    });
  }

  async sendToDevice(
    deviceToken: string,
    notification: NotificationPayload
  ): Promise<void> {
    await this.admin.messaging().send({
      token: deviceToken,
      notification: {
        title: notification.title,
        body: notification.body
      },
      data: notification.data,
      android: {
        priority: 'high',
        notification: {
          channelId: notification.channel
        }
      },
      apns: {
        payload: {
          aps: {
            sound: 'default',
            badge: 1
          }
        }
      }
    });
  }

  async sendToTopic(
```

```
    topic: string,
    notification: NotificationPayload
  ): Promise<void> {
    await this.admin.messaging().sendToTopic(topic, {
      notification: {
        title: notification.title,
        body: notification.body
      },
      data: notification.data
    });
  }
}
```

Notification Types

```
enum NotificationChannel {
  INCIDENTS = 'incidents',
  MAINTENANCE = 'maintenance',
  SESSIONS = 'sessions',
  ALERTS = 'alerts'
}

interface IncidentNotification {
  type: 'INCIDENT_CREATED' | 'INCIDENT_ASSIGNED' | 'INCIDENT_RESOLVED';
  incidentId: string;
  equipmentName: string;
  spaceNumber: string;
  priority: IncidentPriority;
}

// Envoi notification incident cree
await fcmService.sendToTopic('agents-available', {
  title: 'Nouvel incident',
  body: `${incident.equipmentName} - Chambre ${incident.spaceNumber}`,
  channel: NotificationChannel.INCIDENTS,
  data: {
    incidentId: incident.id,
    priority: incident.priority,
    action: 'VIEW_INCIDENT'
  }
});
```

12.2 GESTION ABONNEMENTS

```
// user-service/src/entities/device-token.entity.ts

class DeviceTokenEntity {
  id: string;
  userId: string;
```

```
  token: string;
  platform: 'IOS' | 'ANDROID';
  isActive: boolean;
  lastUsed: Date;
  createdAt: Date;
}

// Enregistrer token
@Post('/devices/register')
async registerDevice(@Body() dto: RegisterDeviceDto, @User() user: User) {
  await this.deviceTokenRepo.save({
    userId: user.id,
    token: dto.token,
    platform: dto.platform,
    isActive: true,
    lastUsed: new Date()
  });
}

// Topics par role
await fcmService.subscribeToTopic(deviceToken, 'agents');
await fcmService.subscribeToTopic(deviceToken, `building-${buildingId}`);
```

---

13. ANALYTICS ET PREDICTIONS

13.1 KPI DASHBOARD

Metriques Building-Service

```
interface BuildingAnalytics {
  totalSpaces: number;
  occupiedSpaces: number;
  availableSpaces: number;
  occupancyRate: number;        // %
  averageSessionDuration: number; // jours
  upcomingExpirations: number;   // 30 jours
  topOccupiedBuildings: BuildingOccupancy[];
}

interface BuildingOccupancy {
  buildingId: string;
  buildingName: string;
  totalSpaces: number;
  occupiedSpaces: number;
  occupancyRate: number;
}

@Injectable()
export class BuildingAnalyticsService {
  async getDashboard(filters: DateRange): Promise<BuildingAnalytics> {
```

```
    // Requetes SQL optimisees avec aggregations
    const stats = await this.db.query(`
      SELECT
        COUNT(*) as total_spaces,
        COUNT(*) FILTER (WHERE status = 'OCCUPIED') as occupied,
        COUNT(*) FILTER (WHERE status = 'AVAILABLE') as available,
        ROUND(
          COUNT(*) FILTER (WHERE status = 'OCCUPIED')::numeric /
          COUNT(*)::numeric * 100,
          2
        ) as occupancy_rate
      FROM spaces
      WHERE created_at >= $1
    `, [filters.startDate]);

    return stats;
  }
}
```

Metriques Equipment-Service

```
interface EquipmentAnalytics {
  totalEquipments: number;
  byCondition: Record<EquipmentCondition, number>;
  byCategory: CategoryStats[];
  totalIncidents: number;
  openIncidents: number;
  averageResolutionTime: number;  // heures
  slaCompliance: number;          // %
  topProblematicSpaces: SpaceStats[];
  maintenanceCosts: number;
  predictedFailures: number;
}

@Injectable()
export class EquipmentAnalyticsService {
  async getDashboard(filters: AnalyticsFilters): Promise<EquipmentAnalytics> {
    // Metriques equipements
    const equipmentStats = await this.equipmentRepo.getStatistics(filters);

    // Metriques incidents
    const incidentStats = await this.incidentRepo.getStatistics(filters);

    // Calcul SLA compliance
    const slaCompliance = this.calculateSLACompliance(incidentStats);

    // Top espaces problematiques
    const topSpaces = await this.getTopProblematicSpaces(filters);

    // Predictions
    const predictions = await this.mlService.predictFailures({
```

```
      threshold: 0.7,
      horizon: 30 // jours
    });

    return {
      totalEquipments: equipmentStats.total,
      byCondition: equipmentStats.byCondition,
      byCategory: equipmentStats.byCategory,
      totalIncidents: incidentStats.total,
      openIncidents: incidentStats.open,
      averageResolutionTime: incidentStats.avgResolutionTime,
      slaCompliance,
      topProblematicSpaces: topSpaces,
      maintenanceCosts: incidentStats.totalCosts,
      predictedFailures: predictions.length
    };
  }

  private calculateSLACompliance(stats: IncidentStatistics): number {
    const onTime = stats.resolvedWithinSLA;
    const total = stats.totalResolved;
    return total > 0 ? (onTime / total) * 100 : 100;
  }
}
```

13.2 MACHINE LEARNING PREDICTIONS

Feature Engineering

```
interface EquipmentFeatures {
  age: number;                    // mois
  incidentFrequency: number;      // incidents/mois
  daysSinceLastMaintenance: number;
  totalIncidents: number;
  averageResolutionTime: number;  // heures
  categoryRiskScore: number;      // 0-1
  currentCondition: number;       // encode: GOOD=1, FAIR=0.5, POOR=0
  seasonalFactor: number;         // 0-1
  usageIntensity: number;         // 0-1
}

@Injectable()
export class FeatureExtractor {
  async extract(equipment: EquipmentEntity): Promise<EquipmentFeatures> {
    const age = this.calculateAge(equipment.purchaseDate);
    const incidents = await this.incidentRepo.findByEquipmentId(equipment.id);

    const incidentFrequency = this.calculateIncidentFrequency(
      incidents,
      age
    );
```

```
    const lastMaintenance = await this.maintenanceRepo.findLastByEquipment(
      equipment.id
    );

    const daysSinceLastMaintenance = lastMaintenance
      ? daysBetween(lastMaintenance.performedAt, new Date())
      : 999;

    const categoryRiskScore = await this.getCategoryRiskScore(
      equipment.categoryId
    );

    return {
      age,
      incidentFrequency,
      daysSinceLastMaintenance,
      totalIncidents: incidents.length,
      averageResolutionTime: this.calculateAvgResolutionTime(incidents),
      categoryRiskScore,
      currentCondition: this.encodeCondition(equipment.condition),
      seasonalFactor: this.getSeasonalFactor(new Date()),
      usageIntensity: equipment.metadata.usageIntensity || 0.5
    };
  }

  private encodeCondition(condition: EquipmentCondition): number {
    const mapping = {
      [EquipmentCondition.EXCELLENT]: 1.0,
      [EquipmentCondition.GOOD]: 0.8,
      [EquipmentCondition.FAIR]: 0.5,
      [EquipmentCondition.POOR]: 0.3,
      [EquipmentCondition.TO_REPAIR]: 0.1,
      [EquipmentCondition.TO_REPLACE]: 0.0,
      [EquipmentCondition.BROKEN]: 0.0
    };
    return mapping[condition];
  }
}
```

Modele de Prediction (Scikit-learn via Python)

```
# ml-service/predict.py

import pandas as pd
from sklearn.ensemble import RandomForestClassifier
import joblib

class EquipmentFailurePredictor:
    def __init__(self):
        self.model = joblib.load('models/failure_predictor.pkl')
```

```
    def predict(self, features: dict) -> dict:
        """
        Predire probabilite de panne dans les 30 prochains jours
        """
        df = pd.DataFrame([features])

        # Prediction
        probability = self.model.predict_proba(df)[0][1]

        # Feature importance
        importance = dict(zip(
            ['age', 'incidentFrequency', 'daysSinceLastMaintenance', ...],
            self.model.feature_importances_
        ))

        # Top factors
        top_factors = sorted(
            importance.items(),
            key=lambda x: x[1],
            reverse=True
        )[:3]

        return {
            'probability': float(probability),
            'risk_level': self._get_risk_level(probability),
            'top_factors': top_factors,
            'recommendation': self._get_recommendation(probability, features)
        }

    def _get_risk_level(self, prob: float) -> str:
        if prob >= 0.8: return 'CRITICAL'
        if prob >= 0.6: return 'HIGH'
        if prob >= 0.4: return 'MEDIUM'
        return 'LOW'

    def _get_recommendation(self, prob: float, features: dict) -> str:
        if prob >= 0.7:
            return "Maintenance preventive urgente recommandee"
        if features['daysSinceLastMaintenance'] > 180:
            return "Programmer une maintenance preventive"
        return "Continuer surveillance normale"
```

Integration NestJS <-> Python

```
// equipment-service/src/ml/ml.service.ts

@Injectable()
export class MLService {
  private pythonProcess: ChildProcess;
```

```
  constructor() {
    // Lancer serveur Python Flask
    this.pythonProcess = spawn('python', ['ml-service/server.py']);
  }

  async predictFailure(
    equipmentId: string
  ): Promise<FailurePrediction> {
    const equipment = await this.equipmentRepo.findById(equipmentId);
    const features = await this.featureExtractor.extract(equipment);

    const response = await axios.post('http://ml-service:5000/predict', {
      features
    });

    return {
      equipmentId,
      equipmentName: equipment.name,
      spaceId: equipment.spaceId,
      probability: response.data.probability,
      riskLevel: response.data.risk_level,
      factors: response.data.top_factors,
      recommendation: response.data.recommendation,
      predictedAt: new Date()
    };
  }

  async batchPredict(
    filters?: PredictionFilters
  ): Promise<FailurePrediction[]> {
    const equipments = await this.equipmentRepo.findAll(filters);

    return await Promise.all(
      equipments.map(e => this.predictFailure(e.id))
    );
  }
}
```

---

14. GEOLOCALISATION INDOOR

14.1 SYSTEME DE COORDONNEES

Conversion GPS -> Indoor

```
// building-service/src/geolocation/coordinate-converter.ts

interface GPS {
  lat: number;
  lng: number;
}
```

```typescript
interface Indoor {
  buildingId: string;
  floorNumber: number;
  x: number;  // metres depuis origine batiment
  y: number;
}

@Injectable()
export class CoordinateConverter {
  async gpsToIndoor(gps: GPS): Promise<Indoor | null> {
    // 1. Trouver batiment le plus proche
    const building = await this.findNearestBuilding(gps);

    if (!building) return null;

    // 2. Calculer position relative
    const origin = { lat: building.latitude, lng: building.longitude };
    const distance = this.haversineDistance(gps, origin);
    const bearing = this.calculateBearing(origin, gps);

    // 3. Convertir en coordonnees cartesiennes
    const x = distance * Math.sin(bearing * Math.PI / 180);
    const y = distance * Math.cos(bearing * Math.PI / 180);

    // 4. Determiner etage (suppose RDC)
    const floorNumber = 0;

    return {
      buildingId: building.id,
      floorNumber,
      x,
      y
    };
  }

  private haversineDistance(p1: GPS, p2: GPS): number {
    const R = 6371e3; // Rayon terre en metres
    const φ1 = p1.lat * Math.PI / 180;
    const φ2 = p2.lat * Math.PI / 180;
    const Δφ = (p2.lat - p1.lat) * Math.PI / 180;
    const Δλ = (p2.lng - p1.lng) * Math.PI / 180;

    const a = Math.sin(Δφ/2) * Math.sin(Δφ/2) +
              Math.cos(φ1) * Math.cos(φ2) *
              Math.sin(Δλ/2) * Math.sin(Δλ/2);

    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));

    return R * c;
  }
}
```

14.2 GENERATION SVG INTERACTIF

```
// building-service/src/maps/svg-generator.service.ts

@Injectable()
export class SVGMapGenerator {
  async generateFloorPlan(
    buildingId: string,
    floorNumber: number
  ): Promise<string> {
    const floor = await this.floorRepo.findByNumber(buildingId, floorNumber);
    const spaces = await this.spaceRepo.findByFloorId(floor.id);

    const width = 1000;
    const height = 800;

    let svg = `
      <svg width="${width}" height="${height}" xmlns="http://www.w3.org/2000/svg">
        <defs>
          <style>
            .space { fill: #e0e0e0; stroke: #333; stroke-width: 2; }
            .space.available { fill: #4caf50; }
            .space.occupied { fill: #2196f3; }
            .space.maintenance { fill: #ff9800; }
            .space-label { font-size: 14px; text-anchor: middle; }
          </style>
        </defs>

        <!-- Grid background -->
        <rect width="${width}" height="${height}" fill="#f5f5f5"/>
        <pattern id="grid" width="50" height="50" patternUnits="userSpaceOnUse">
          <path d="M 50 0 L 0 0 0 50" fill="none" stroke="#ccc" stroke-width="1"/>
        </pattern>
        <rect width="${width}" height="${height}" fill="url(#grid)"/>
    `;

    // Dessiner chaque espace
    for (const space of spaces) {
      const coords = space.coordinates;

      if (coords.polygon) {
        svg += this.renderPolygon(space, coords.polygon);
      } else {
        svg += this.renderRectangle(space, coords);
      }

      svg += this.renderSpaceLabel(space, coords);
      svg += this.renderEquipmentIcons(space);
    }

    svg += `</svg>`;
```

```
    return svg;
  }

  private renderPolygon(space: SpaceEntity, polygon: Point[]): string {
    const points = polygon.map(p => `${p.x},${p.y}`).join(' ');
    const cssClass = `space ${space.status.toLowerCase()}`;

    return `
      <polygon
        points="${points}"
        class="${cssClass}"
        data-space-id="${space.id}"
        data-space-number="${space.number}"
        onclick="selectSpace('${space.id}')"
      />
    `;
  }

  private renderSpaceLabel(space: SpaceEntity, coords: any): string {
    const centerX = coords.x + 25;
    const centerY = coords.y + 25;

    return `
      <text x="${centerX}" y="${centerY}" class="space-label">
        ${space.number}
      </text>
    `;
  }

  private renderEquipmentIcons(space: SpaceEntity): string {
    // Icones pour equipements (AC, TV, etc.)
    let icons = '';
    const iconSize = 20;
    let offsetX = space.coordinates.x + 5;
    const offsetY = space.coordinates.y + 5;

    if (space.features.hasAC) {
      icons += `<image href="/icons/ac.svg" x="${offsetX}" y="${offsetY}"
width="${iconSize}" height="${iconSize}"/>`;
      offsetX += iconSize + 5;
    }

    if (space.features.hasWindow) {
      icons += `<image href="/icons/window.svg" x="${offsetX}" y="${offsetY}"
width="${iconSize}" height="${iconSize}"/>`;
      offsetX += iconSize + 5;
    }

    return icons;
  }
}
```

14.3 RECHERCHE SPATIALE

```
// building-service/src/geolocation/spatial-search.service.ts

@Injectable()
export class SpatialSearchService {
  async findNearbySpaces(
    referenceSpaceId: string,
    radiusMeters: number,
    filters?: SpaceFilters
  ): Promise<SpaceEntity[]> {
    const referenceSpace = await this.spaceRepo.findById(referenceSpaceId);

    // Requete PostGIS
    const query = `
      SELECT s.*,
        ST_Distance(
          ST_MakePoint($1, $2),
          ST_MakePoint(
            (s.coordinates->>'x')::float,
            (s.coordinates->>'y')::float
          )
        ) as distance
      FROM spaces s
      WHERE
        s.building_id = $3
        AND s.floor_id = $4
        AND ST_DWithin(
          ST_MakePoint($1, $2),
          ST_MakePoint(
            (s.coordinates->>'x')::float,
            (s.coordinates->>'y')::float
          ),
          $5
        )
        AND s.id != $6
      ORDER BY distance ASC
    `;

    const results = await this.db.query(query, [
      referenceSpace.coordinates.x,
      referenceSpace.coordinates.y,
      referenceSpace.buildingId,
      referenceSpace.floorId,
      radiusMeters,
      referenceSpaceId
    ]);

    return results.rows.map(row => this.mapToEntity(row));
  }

  async findOptimalPath(
```

```
    fromSpaceId: string,
    toSpaceId: string
  ): Promise<PathSegment[]> {
    // Algorithme A* pour trouver chemin optimal
    // Prend en compte couloirs, escaliers, ascenseurs

    const from = await this.spaceRepo.findById(fromSpaceId);
    const to = await this.spaceRepo.findById(toSpaceId);

    if (from.buildingId !== to.buildingId) {
      throw new Error('Cross-building paths not supported');
    }

    // Implementation A*
    return this.aStarPathfinding(from, to);
  }
}
```

---

15. RECOMMANDATIONS D'IMPLEMENTATION

15.1 ORDRE D'IMPLEMENTATION

Phase 1: Fondations (Semaines 1-2)

```
1. Setup infrastructure
   - PostgreSQL databases
   - Redis
   - RabbitMQ
   - Docker Compose

2. Building-Service (MVP)
   - Entities: Site, Building, Floor, Space
   - CRUD operations
   - Import Excel basique

3. Equipment-Service (MVP)
   - Entities: Category, Equipment
   - CRUD operations
   - Assignation equipements <-> espaces
```

Phase 2: Core Features (Semaines 3-5)

```
4. Incidents Management
   - Incident entity
   - Create/Assign/Resolve workflow
   - Photo upload (S3)
```

```
5. Occupant Sessions
   - OccupantSession entity
   - Assignation/Expiration
   - Integration user-service

6. Push Notifications
   - FCM integration
   - Event-driven notifications

7. Mobile App (Flutter)
   - Login
   - View assigned space
   - Create incident
   - View incident history
```

Phase 3: Advanced Features (Semaines 6-8)

```
8. Analytics & Statistics
   - KPI dashboards
   - Reporting Excel
   - Graphiques

9. Indoor Mapping
   - SVG generation
   - Interactive maps
   - Spatial search

10. Offline Support
    - IndexedDB/SQLite
    - Sync queue
    - Conflict resolution

11. Predictions ML
    - Feature extraction
    - Python service
    - Integration
```

Phase 4: Polish & Optimization (Semaines 9-10)

```
12. Performance
    - Caching strategy
    - Database optimization
    - Load testing

13. Security
    - Input validation
    - Rate limiting
    - Audit logs
```

```
14. Documentation
    - API documentation
    - User guides
    - Admin manual
```

15.2 TESTS RECOMMANDES

Tests Unitaires

```typescript
// equipment-service/src/application/use-cases/create-incident.use-case.spec.ts

describe('CreateIncidentUseCase', () => {
  let useCase: CreateIncidentUseCase;
  let equipmentRepo: jest.Mocked<IEquipmentRepository>;
  let incidentRepo: jest.Mocked<IIncidentRepository>;

  beforeEach(() => {
    equipmentRepo = {
      findById: jest.fn()
    } as any;

    incidentRepo = {
      save: jest.fn()
    } as any;

    useCase = new CreateIncidentUseCase(
      equipmentRepo,
      incidentRepo,
      // ...
    );
  });

  it('should create incident with calculated priority', async () => {
    const equipment = new EquipmentEntity(/* ... */);
    equipment.category.criticality = Criticality.HIGH;

    equipmentRepo.findById.mockResolvedValue(equipment);

    const result = await useCase.execute({
      equipmentId: 'eq-1',
      category: IncidentCategory.BREAKDOWN,
      description: 'Panne AC'
    });

    expect(result.priority).toBe(IncidentPriority.HIGH);
    expect(incidentRepo.save).toHaveBeenCalled();
  });
});
```

Tests Integration

```
// equipment-service/test/integration/incident-workflow.spec.ts

describe('Incident Workflow Integration', () => {
  let app: INestApplication;

  beforeAll(async () => {
    const module = await Test.createTestingModule({
      imports: [AppModule]
    }).compile();

    app = module.createNestApplication();
    await app.init();
  });

  it('should complete full incident lifecycle', async () => {
    // 1. Create incident
    const createResponse = await request(app.getHttpServer())
      .post('/incidents')
      .send({
        equipmentId: 'eq-1',
        title: 'AC not working',
        description: 'No cold air'
      })
      .expect(201);

    const incidentId = createResponse.body.id;

    // 2. Assign to agent
    await request(app.getHttpServer())
      .post(`/incidents/${incidentId}/assign`)
      .send({ agentId: 'agent-1' })
      .expect(200);

    // 3. Resolve
    await request(app.getHttpServer())
      .post(`/incidents/${incidentId}/resolve`)
      .send({
        resolution: 'Filter cleaned',
        newCondition: EquipmentCondition.GOOD
      })
      .expect(200);

    // 4. Verify status
    const getResponse = await request(app.getHttpServer())
      .get(`/incidents/${incidentId}`)
      .expect(200);

    expect(getResponse.body.status).toBe(IncidentStatus.RESOLVED);
  });
});
```

15.3 SECURITE

Input Validation

```
// equipment-service/src/application/dto/create-incident.dto.ts

import { IsString, IsUUID, IsEnum, IsOptional, MaxLength, IsArray } from 'class-validator';

export class CreateIncidentDto {
  @IsUUID()
  equipmentId: string;

  @IsUUID()
  spaceId: string;

  @IsString()
  @MaxLength(255)
  title: string;

  @IsString()
  @MaxLength(2000)
  description: string;

  @IsEnum(IncidentCategory)
  category: IncidentCategory;

  @IsOptional()
  @IsEnum(IncidentPriority)
  priority?: IncidentPriority;

  @IsOptional()
  @IsArray()
  photos?: File[];
}
```

Authorization Guards

```
// building-service/src/infrastructure/http/guards/space-access.guard.ts

@Injectable()
export class SpaceAccessGuard implements CanActivate {
  constructor(
    private sessionRepo: IOccupantSessionRepository
  ) {}

  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest();
```

```
    const user = request.user;
    const spaceId = request.params.spaceId;

    // Superadmin peut tout
    if (user.role === UserRole.SUPERADMIN) {
      return true;
    }

    // Agent peut tout
    if (user.role === UserRole.AGENT) {
      return true;
    }

    // Occupant peut seulement ses espaces
    if (user.role === UserRole.OCCUPANT) {
      const sessions = await this.sessionRepo.findActiveByOccupant(user.id);
      const hasAccess = sessions.some(s => s.spaceId === spaceId);

      if (!hasAccess) {
        throw new ForbiddenException('Access denied to this space');
      }

      return true;
    }

    return false;
  }
}

// Usage
@Get('spaces/:spaceId/equipments')
@UseGuards(JwtAuthGuard, SpaceAccessGuard)
async getSpaceEquipments(@Param('spaceId') spaceId: string) {
  // ...
}
```

15.4 MONITORING & LOGGING

Logging Structure

```
// Logger centralisé

@Injectable()
export class AppLogger {
  private logger = winston.createLogger({
    format: winston.format.combine(
      winston.format.timestamp(),
      winston.format.json()
    ),
    transports: [
      new winston.transports.File({ filename: 'error.log', level: 'error' }),
```

```
      new winston.transports.File({ filename: 'combined.log' })
    ]
  });

  logIncidentCreated(incident: IncidentEntity, userId: string) {
    this.logger.info('Incident created', {
      incidentId: incident.id,
      equipmentId: incident.equipmentId,
      spaceId: incident.spaceId,
      priority: incident.priority,
      reportedBy: userId,
      timestamp: new Date()
    });
  }

  logError(error: Error, context: string) {
    this.logger.error('Application error', {
      message: error.message,
      stack: error.stack,
      context,
      timestamp: new Date()
    });
  }
}
```

Health Checks

```
// building-service/src/health/health.controller.ts

@Controller('health')
export class HealthController {
  constructor(
    private db: DatabaseHealthIndicator,
    private redis: RedisHealthIndicator,
    private rabbitmq: RabbitMQHealthIndicator
  ) {}

  @Get()
  async check() {
    return {
      status: 'ok',
      timestamp: new Date(),
      checks: {
        database: await this.db.check(),
        redis: await this.redis.check(),
        rabbitmq: await this.rabbitmq.check()
      }
    };
  }
}
```

CONCLUSION

Cette etude architecturale fournit une base solide pour implementer les microservices Building et Equipment dans le projet IMMO360.

Points cles:

1. Architecture hexagonale garantit maintenabilite et testabilite
2. Design patterns (DDD, Repository, Strategy, Saga) assurent evolutivite
3. Event-driven architecture permet scalabilite horizontale
4. Offline-first garantit disponibilite mobile
5. Machine Learning apporte valeur predictive
6. Indoor mapping resout probleme geolocalisation

Prochaines etapes:

1. Valider architecture avec equipe
2. Prioriser features (MVP vs Nice-to-have)
3. Implementer Phase 1 (Fondations)
4. Iterer progressivement

Estimation totale: 10 semaines developpement + 2 semaines tests/deployment

Equipe recommandee:

- 2 Backend developers (NestJS)
- 1 Mobile developer (Flutter)
- 1 DevOps
- 1 ML Engineer (Python)