Procédure complète de migration vers l'architecture DDD/CQRS avec IA

🚨 PHASE 0 : PRÉPARATIFS ET SAUVEGARDE

Étape 0.1 : Sauvegarde complète

```bash
# 1. Créer une branche de sauvegarde
git add .
git commit -m "💾 Sauvegarde avant migration vers architecture DDD/CQRS/IA"
git branch backup-before-ddd-migration
git push origin backup-before-ddd-migration

# 2. Créer une branche pour la nouvelle architecture
git checkout -b feature/ddd-cqrs-ai-architecture
```

Étape 0.2 : Vérification environnement

```bash
# Vérifier versions
scala --version  # Doit être 2.13.x
sbt --version   # Doit être 1.8.x+
docker --version
docker-compose --version

# Vérifier base de données actuelle
docker-compose ps
```

Étape 0.3 : Arrêt services actuels

```bash
# Arrêter l'application actuelle
sbt stop  # ou Ctrl+C si en cours
docker-compose down
```

🔄 PHASE 0.5 : MIGRATION DESTRUCTIVE

Étape 0.5.1 : Sauvegarde sélective données importantes

```bash
```

```
# Créer dossier de sauvegarde
mkdir -p migration-backup/$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="migration-backup/$(date +%Y%m%d_%H%M%S)"

# Sauvegarder les fichiers à conserver
cp -r conf/reseed/ $BACKUP_DIR/
cp conf/application.conf $BACKUP_DIR/
cp docker-compose.yml $BACKUP_DIR/
cp build.sbt $BACKUP_DIR/

# Sauvegarder données existantes si important
if [ -d "app/controllers" ]; then
  cp -r app/controllers $BACKUP_DIR/old-controllers/
fi
if [ -d "app/models" ]; then
  cp -r app/models $BACKUP_DIR/old-models/
fi
```

Étape 0.5.2 : Nettoyage structure actuelle

```bash
# ⚠️ ATTENTION : CECI VA SUPPRIMER LA STRUCTURE ACTUELLE
# Supprimer uniquement les dossiers que nous allons remplacer
rm -rf app/controllers app/models app/repositories app/services
rm -rf app/infrastructure app/interfaces app/modules app/utils
rm -rf elm/admin elm/public

# Garder conf/, public/, test/, project/, docs/ pour l'instant
# Nous les migrerons de façon contrôlée
```

Étape 0.5.3 : Exécution du script de création

```bash
# Télécharger et exécuter le script
chmod +x create-project-structure.sh
./create-project-structure.sh

echo "✅ Nouvelle structure créée !"
```

Étape 0.5.4 : Récupération configuration existante

```bash
```

```
# Merger les configurations existantes
if [ -f "$BACKUP_DIR/application.conf" ]; then
  echo "# Configuration fusionnée - voir backup pour ancienne config" >> conf/application.conf
  echo "" >> conf/application.conf
  cat $BACKUP_DIR/application.conf >> conf/application.conf
fi

# Récupérer les données CSV existantes
if [ -d "$BACKUP_DIR/reseed" ]; then
  cp -r $BACKUP_DIR/reseed/* conf/reseed/ingredients/ 2>/dev/null || true
fi
```

🏗️ PHASE 1 : FONDATIONS DDD (Semaine 1-2)

Étape 1.1 : Value Objects de base

Jour 1 : Value Objects fondamentaux

```scala
// app/domain/common/ValueObject.scala
trait ValueObject[A] {
  def value: A

  override def equals(obj: Any): Boolean = obj match {
    case vo: ValueObject[_] => vo.value == value
    case _ => false
  }

  override def hashCode(): Int = value.hashCode()
  override def toString: String = s"${getClass.getSimpleName}($value)"
}

// app/domain/common/DomainError.scala
sealed trait DomainError {
  def message: String
}

case class ValidationError(field: String, message: String) extends DomainError
case class BusinessRuleViolation(rule: String, message: String) extends DomainError
case class NotFoundError(resource: String, id: String) extends DomainError {
  def message: String = s"$resource with ID $id not found"
}
```

Jour 2-3 : Value Objects spécialisés

scala

```scala
// app/domain/shared/Email.scala
import scala.util.matching.Regex

case class Email private (value: String) extends ValueObject[String]

object Email {
  private val EmailRegex: Regex = """^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$""".r

  def create(value: String): Either[ValidationError, Email] = {
    val trimmed = value.trim.toLowerCase
    if (EmailRegex.matches(trimmed)) Right(Email(trimmed))
    else Left(ValidationError("email", s"Invalid email format: $value"))
  }
}

// app/domain/shared/NonEmptyString.scala
case class NonEmptyString private (value: String) extends ValueObject[String]

object NonEmptyString {
  def create(value: String): Either[ValidationError, NonEmptyString] = {
    val trimmed = value.trim
    if (trimmed.nonEmpty) Right(NonEmptyString(trimmed))
    else Left(ValidationError("value", "String cannot be empty"))
  }
}

// app/domain/shared/Volume.scala
case class Volume private (liters: Double) extends ValueObject[Double] {
  def value: Double = liters
  def toGallons: Double = liters * 0.264172
  def toBarrels: Double = liters / 117.348
}

object Volume {
  def fromLiters(liters: Double): Either[ValidationError, Volume] = {
    if (liters > 0) Right(Volume(liters))
    else Left(ValidationError("volume", "Volume must be positive"))
  }

  def fromGallons(gallons: Double): Either[ValidationError, Volume] = {
    fromLiters(gallons / 0.264172)
  }
}
```

Étape 1.2 : Domaine Admin sécurisé

Jour 4-5 : Admin avec permissions

scala

```scala
// app/domain/admin/model/AdminPermission.scala
sealed trait AdminPermission
object AdminPermission {
  case object MANAGE_REFERENTIALS extends AdminPermission
  case object MANAGE_INGREDIENTS extends AdminPermission
  case object MANAGE_USERS extends AdminPermission
  case object VIEW_ANALYTICS extends AdminPermission
  case object IMPORT_DATA extends AdminPermission
  case object APPROVE_AI_PROPOSALS extends AdminPermission
  case object CONFIGURE_AI_DISCOVERY extends AdminPermission

  val ALL: Set[AdminPermission] = Set(
    MANAGE_REFERENTIALS, MANAGE_INGREDIENTS, MANAGE_USERS,
    VIEW_ANALYTICS, IMPORT_DATA, APPROVE_AI_PROPOSALS, CONFIGURE_AI_DISCOVERY
  )
}

// app/domain/admin/model/AdminRole.scala
case class AdminRole private (
  name: String,
  permissions: Set[AdminPermission]
)

object AdminRole {
  val SUPER_ADMIN = AdminRole("super_admin", AdminPermission.ALL)
  val CONTENT_MANAGER = AdminRole("content_manager", Set(
    AdminPermission.MANAGE_REFERENTIALS, AdminPermission.MANAGE_INGREDIENTS,
    AdminPermission.IMPORT_DATA, AdminPermission.APPROVE_AI_PROPOSALS
  ))
  val DATA_ANALYST = AdminRole("data_analyst", Set(
    AdminPermission.VIEW_ANALYTICS
  ))
}

// app/domain/admin/model/AdminAggregate.scala
import java.time.Instant

case class AdminAggregate private (
  id: AdminId,
  email: Email,
  name: AdminName,
  role: AdminRole,
  isActive: Boolean,
  lastLoginAt: Option[Instant],
  createdAt: Instant,
  version: Long
) {

  def hasPermission(permission: AdminPermission): Boolean =
    isActive && role.permissions.contains(permission)

  def login(): AdminAggregate =
    this.copy(lastLoginAt = Some(Instant.now()), version = version + 1)

  def deactivate(): AdminAggregate =
    this.copy(isActive = false, version = version + 1)
}

object AdminAggregate {
  def create(
    id: AdminId,
    email: Email,
    name: AdminName,
    role: AdminRole
  ): AdminAggregate = AdminAggregate(
    id = id,
    email = email,
    name = name,
    role = role,
    isActive = true,
    lastLoginAt = None,
    createdAt = Instant.now(),
    version = 0L
  )
}
```

Étape 1.3 : Base de données avec audit

Jour 6-7 : Configuration Slick et évolutions

```
sql
```

```sql
-- conf/evolutions/default/1.sql

-- Admins avec permissions
CREATE TABLE admins (
   id VARCHAR(255) PRIMARY KEY,
   email VARCHAR(255) UNIQUE NOT NULL,
   first_name VARCHAR(255) NOT NULL,
   last_name VARCHAR(255) NOT NULL,
   password_hash VARCHAR(255) NOT NULL,
   role VARCHAR(50) NOT NULL,
   is_active BOOLEAN DEFAULT true,
   last_login_at TIMESTAMP,
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   version BIGINT DEFAULT 0
);

-- Table audit pour traçabilité
CREATE TABLE audit_logs (
   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
   admin_id VARCHAR(255),
   action VARCHAR(50) NOT NULL,
   resource_type VARCHAR(100) NOT NULL,
   resource_id VARCHAR(255),
   changes JSONB,
   ip_address INET,
   user_agent TEXT,
   timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

   FOREIGN KEY (admin_id) REFERENCES admins(id)
);

-- Index pour performance
CREATE INDEX idx_audit_logs_admin_id ON audit_logs(admin_id);
CREATE INDEX idx_audit_logs_timestamp ON audit_logs(timestamp);
CREATE INDEX idx_audit_logs_resource ON audit_logs(resource_type, resource_id);
```

scala

```scala
// app/infrastructure/persistence/slick/tables/AdminTables.scala
import slick.jdbc.PostgresProfile.api._
import java.time.Instant

case class AdminRow(
  id: String,
  email: String,
  firstName: String,
  lastName: String,
  passwordHash: String,
  role: String,
  isActive: Boolean,
  lastLoginAt: Option[Instant],
  createdAt: Instant,
  version: Long
)

class Admins(tag: Tag) extends Table[AdminRow](tag, "admins") {
  def id = column[String]("id", O.PrimaryKey)
  def email = column[String]("email", O.Unique)
  def firstName = column[String]("first_name")
  def lastName = column[String]("last_name")
  def passwordHash = column[String]("password_hash")
  def role = column[String]("role")
  def isActive = column[Boolean]("is_active")
  def lastLoginAt = column[Option[Instant]]("last_login_at")
  def createdAt = column[Instant]("created_at")
  def version = column[Long]("version")

  def * = (id, email, firstName, lastName, passwordHash, role,
           isActive, lastLoginAt, createdAt, version) <> (AdminRow.tupled, AdminRow.unapply)
}
```

Étape 1.4 : Actions sécurisées

Jour 8-9 : Actions Play Framework sécurisées

```
scala
```

```scala
// app/interfaces/actions/AdminSecuredAction.scala
import javax.inject._
import play.api.mvc._
import scala.concurrent.{ExecutionContext, Future}

case class AdminRequest[A](admin: AdminAggregate, request: Request[A]) extends WrappedRequest[A](reques

@Singleton
class AdminSecuredAction @Inject()(
  parser: BodyParsers.Default,
  adminRepo: AdminReadRepository
)(implicit ec: ExecutionContext) extends ActionBuilder[AdminRequest, AnyContent] {

  override def parser: BodyParser[AnyContent] = parser.default
  override protected def executionContext: ExecutionContext = ec

  override def invokeBlock[A](request: Request[A], block: AdminRequest[A] => Future[Result]): Future[Result] = {
    extractAdminFromSession(request).flatMap {
      case Some(admin) if admin.isActive =>
        block(AdminRequest(admin, request))
      case Some(_) =>
        Future.successful(Results.Forbidden(Json.obj("error" -> "account_disabled")))
      case None =>
        Future.successful(Results.Unauthorized(Json.obj("error" -> "authentication_required")))
    }
  }

  private def extractAdminFromSession[A](request: Request[A]): Future[Option[AdminAggregate]] = {
    request.session.get("admin_id") match {
      case Some(adminId) => adminRepo.byId(AdminId(adminId))
      case None => Future.successful(None)
    }
  }

  // Action avec permission spécifique
  def withPermission(permission: AdminPermission): ActionBuilder[AdminRequest, AnyContent] =
    new ActionBuilder[AdminRequest, AnyContent] {
      override def parser = AdminSecuredAction.this.parser
      override protected def executionContext = AdminSecuredAction.this.executionContext

      override def invokeBlock[A](request: Request[A], block: AdminRequest[A] => Future[Result]): Future[Result] 
        AdminSecuredAction.this.invokeBlock(request, { adminReq =>
          if (adminReq.admin.hasPermission(permission)) {
            block(adminReq)
          } else {
            Future.successful(Results.Forbidden(Json.obj(
        "error" -> "insufficient_permissions",
        "required_permission" -> permission.toString
      )))
    }
  })
 }
}
}
```

Étape 1.5 : Tests fondamentaux

Jour 10 : Tests unitaires de base

```
scala
```

```scala
// test/domain/shared/EmailSpec.scala
import org.scalatest.wordspec.AnyWordSpec
import org.scalatest.matchers.should.Matchers

class EmailSpec extends AnyWordSpec with Matchers {

  "Email" should {
    "create valid email successfully" in {
      val result = Email.create("test@example.com")
      result shouldBe Right(Email("test@example.com"))
    }

    "reject invalid email format" in {
      val result = Email.create("invalid-email")
      result.isLeft shouldBe true
    }

    "normalize email to lowercase" in {
      val result = Email.create("Test@EXAMPLE.COM")
      result shouldBe Right(Email("test@example.com"))
    }
  }
}

// test/domain/admin/AdminAggregateSpec.scala
class AdminAggregateSpec extends AnyWordSpec with Matchers {

  "AdminAggregate" should {
    "create new admin successfully" in {
      val admin = AdminAggregate.create(
        AdminId("test-id"),
        Email("admin@test.com"),
        AdminName("John", "Doe"),
        AdminRole.CONTENT_MANAGER
      )

      admin.isActive shouldBe true
      admin.role shouldBe AdminRole.CONTENT_MANAGER
      admin.hasPermission(AdminPermission.MANAGE_REFERENTIALS) shouldBe true
      admin.hasPermission(AdminPermission.MANAGE_USERS) shouldBe false
    }

    "deactivate admin" in {
      val admin = AdminAggregate.create(/*...*/)
      val deactivated = admin.deactivate()
      deactivated.isActive shouldBe false
      deactivated.hasPermission(AdminPermission.MANAGE_REFERENTIALS) shouldBe false
    }
  }
}
```

✅ CHECKPOINT PHASE 1

```bash
# Vérifier que tout compile
sbt compile

# Lancer les tests
sbt test

# Créer admin par défaut via evolution
# conf/evolutions/default/2.sql
INSERT INTO admins (id, email, first_name, last_name, password_hash, role)
VALUES (
  'admin-1',
  'admin@brewery.com',
  'Super',
  'Admin',
  '$2a$10$hash...', -- Générer avec bcrypt
  'super_admin'
);

# Démarrer l'application
sbt run
```

🍺 PHASE 2 : DOMAINE HOUBLONS (Semaine 3-4)

Jour 11-12 : Modèle domaine Hop

```scala
```

```scala
// app/domain/hops/model/HopAggregate.scala
case class HopAggregate private (
  id: HopId,
  name: HopName,
  alphaAcid: Option[AlphaAcidPercentage],
  betaAcid: Option[BetaAcidPercentage],
  usage: HopUsage,
  origins: Set[OriginId],
  aromas: Set[AromaId],
  status: HopStatus,
  source: HopSource, // Manual, AI_Discovered, Import
  credibility: HopCredibility, // Score 0-100
  createdAt: Instant,
  version: Long
) {

  def updateAlphaAcid(alpha: AlphaAcidPercentage): Either[DomainError, HopAggregate] = {
    if (alpha.value >= 0 && alpha.value <= 30) {
      Right(this.copy(alphaAcid = Some(alpha), version = version + 1))
    } else {
      Left(ValidationError("alpha_acid", "Alpha acid must be between 0 and 30%"))
    }
  }

  def addAroma(aromaId: AromaId): Either[DomainError, HopAggregate] = {
    if (aromas.contains(aromaId)) {
      Left(BusinessRuleViolation("duplicate_aroma", s"Aroma ${aromaId.value} already exists"))
    } else {
      Right(this.copy(aromas = aromas + aromaId, version = version + 1))
    }
  }

  def isReliable: Boolean = credibility.value >= 70
  def isAiDiscovered: Boolean = source == HopSource.AI_DISCOVERED
}
```

Jour 13-14 : Repository et infrastructure Hop

```
scala
```

```scala
// app/domain/hops/repositories/HopReadRepository.scala
trait HopReadRepository {
  def byId(id: HopId): Future[Option[HopAggregate]]
  def byName(name: HopName): Future[Option[HopAggregate]]
  def all(page: Int = 0, size: Int = 20): Future[Page[HopAggregate]]
  def searchByCharacteristics(filter: HopFilter): Future[Seq[HopAggregate]]
  def byStatus(status: HopStatus): Future[Seq[HopAggregate]]
  def aiDiscovered(limit: Int = 50): Future[Seq[HopAggregate]]
}

// app/infrastructure/persistence/slick/repositories/ingredients/SlickHopReadRepository.scala
@Singleton
class SlickHopReadRepository @Inject()(
  dbConfigProvider: DatabaseConfigProvider
)(implicit ec: ExecutionContext) extends HopReadRepository {

  private val db = dbConfigProvider.get[PostgresProfile].db
  import HopTables._

  override def byId(id: HopId): Future[Option[HopAggregate]] = {
    val query = for {
      hop <- hops.filter(_.id === id.value)
      origins <- hopOrigins.filter(_.hopId === id.value)
      aromas <- hopAromas.filter(_.hopId === id.value)
    } yield (hop, origins, aromas)

    db.run(query.result).map(_.headOption.map { case (hopRow, originRows, aromaRows) =>
      toDomainWithRelations(hopRow, originRows, aromaRows)
    })
  }

  private def toDomainWithRelations(
    hopRow: HopRow,
    originRows: Seq[HopOriginRow],
    aromaRows: Seq[HopAromaRow]
  ): HopAggregate = {
    HopAggregate(
      id = HopId(hopRow.id),
      name = HopName(hopRow.name),
      alphaAcid = hopRow.alphaAcid.map(AlphaAcidPercentage(_)),
      // ... autres champs
      origins = originRows.map(row => OriginId(row.originId)).toSet,
      aromas = aromaRows.map(row => AromaId(row.aromaId)).toSet,
      // ...
    )
  }
}
```

Jour 15-16 : API publique Hops (lecture seule)

```scala
```

```scala
// app/interfaces/http/api/v1/public/HopsController.scala
@Singleton
class HopsController @Inject()(
  cc: ControllerComponents,
  hopQueryHandler: HopListQueryHandler,
  hopDetailQueryHandler: HopDetailQueryHandler,
  hopFilterQueryHandler: HopAdvancedFilterQueryHandler
)(implicit ec: ExecutionContext) extends AbstractController(cc) {

  // GET /api/v1/hops
  def list(page: Int = 0, size: Int = 20): Action[AnyContent] = Action.async {
    val query = HopListQuery(page, size)
    hopQueryHandler.handle(query).map {
      case Right(hopsPage) => Ok(Json.toJson(hopsPage))
      case Left(error) => BadRequest(Json.obj("error" -> error.message))
    }
  }

  // GET /api/v1/hops/:id
  def detail(id: String): Action[AnyContent] = Action.async {
    val query = HopDetailQuery(HopId(id))
    hopDetailQueryHandler.handle(query).map {
      case Right(Some(hop)) => Ok(Json.toJson(hop))
      case Right(None) => NotFound(Json.obj("error" -> s"Hop $id not found"))
      case Left(error) => BadRequest(Json.obj("error" -> error.message))
    }
  }

  // POST /api/v1/hops/search (filtrage avancé)
  def search: Action[JsValue] = Action(parse.json).async { implicit request =>
    request.body.validate[HopFilterRequest].fold(
      errors => Future.successful(BadRequest(JsError.toJson(errors))),
      filterReq => {
        val query = HopAdvancedFilterQuery(
          alphaAcidRange = filterReq.alphaAcidRange,
          usage = filterReq.usage,
          origins = filterReq.origins,
          aromas = filterReq.aromas
        )
        hopFilterQueryHandler.handle(query).map {
          case Right(hops) => Ok(Json.toJson(hops))
          case Left(error) => BadRequest(Json.obj("error" -> error.message))
        }
      }
    )
  }
}
```

Jour 17-18 : API admin Hops (écriture sécurisée)

scala

```
// app/interfaces/http/api/admin/AdminHopsController.scala
@Singleton
class AdminHopsController @Inject()(
  cc: ControllerComponents,
  secured: AdminSecuredAction,
  createHopHandler: CreateHopCommandHandler,
  updateHopHandler: UpdateHopCommandHandler,
  auditService: AuditService
)(implicit ec: ExecutionContext) extends AbstractController(cc) {

  // POST /api/admin/hops
  def create: Action[JsValue] =
    secured.withPermission(AdminPermission.MANAGE_INGREDIENTS)(parse.json).async { req =>
      req.body.validate[CreateHopRequest].fold(
        errors => Future.successful(BadRequest(JsError.toJson(errors))),
        hopReq => {
          val command = CreateHopCommand(
            id = HopId.generate(),
            name = hopReq.name,
            alphaAcid = hopReq.alphaAcid,
            usage = hopReq.usage
          )

          for {
            result <- createHopHandler.handle(command)
            _ <- auditService.logHopCreation(req.admin.id, command, req.remoteAddress)
          } yield result match {
            case Right(hopId) => Created(Json.obj("id" -> hopId.value))
            case Left(error) => BadRequest(Json.obj("error" -> error.message))
          }
        }
      )
    }

  // PUT /api/admin/hops/:id
  def update(id: String): Action[JsValue] =
    secured.withPermission(AdminPermission.MANAGE_INGREDIENTS)(parse.json).async { req =>
      // Similaire à create mais avec UpdateHopCommand
      ???
    }
}
```

✅ CHECKPOINT PHASE 2

```
bash
```

```
# Tests complets domaine Hops
sbt "testOnly *HopSpec"

# Vérification API publique
curl "http://localhost:9000/api/v1/hops"
curl "http://localhost:9000/api/v1/hops/cascade-us"

# Test API admin (avec auth)
curl -X POST "http://localhost:9000/api/admin/hops" \
  -H "Content-Type: application/json" \
  -H "Csrf-Token: ..." \
  -d '{"name": "Test Hop", "alphaAcid": 5.5, "usage": "AROMA"}'
```

🤖 PHASE 3 : IA DE VEILLE HOUBLONS (Semaine 5-6)

Jour 19-21 : Services IA de base

scala

```scala
```

```scala
// app/domain/ai_monitoring/services/specialized/HopDiscoveryService.scala
@Singleton
class HopDiscoveryService @Inject()(
  openAiClient: OpenAiClient,
  webScrapingClient: WebScrapingClient,
  hopRepo: HopReadRepository,
  proposalRepo: ProposalWriteRepository
)(implicit ec: ExecutionContext) {

  def discoverNewHops(): Future[Seq[ProposedDataEntry]] = {
    val sources = Seq(
      "https://www.yakimachief.com/hops",
      "https://www.barthhaas.com/hop-varieties",
      "https://www.hopunion.com/hop-varieties"
    )

    Future.traverse(sources)(discoverFromSource).map(_.flatten)
  }

  private def discoverFromSource(url: String): Future[Seq[ProposedDataEntry]] = {
    for {
      content <- webScrapingClient.scrape(url)
      extractedHops <- openAiClient.extractHopData(content, url)
      existingHops <- hopRepo.all()
      newProposals <- generateProposals(extractedHops, existingHops)
    } yield newProposals
  }

  private def generateProposals(
    extractedHops: Seq[ExtractedHopData],
    existingHops: Page[HopAggregate]
  ): Future[Seq[ProposedDataEntry]] = {
    val existing = existingHops.items.map(_.name.value).toSet

    val newHops = extractedHops.filterNot(hop => existing.contains(hop.name))

    Future.traverse(newHops) { hop =>
      for {
        confidence <- calculateConfidence(hop)
        proposal <- createProposal(hop, confidence)
        _ <- proposalRepo.save(proposal)
      } yield proposal
    }
  }

  private def calculateConfidence(hop: ExtractedHopData): Future[ProposalConfidence] = {
    // IA pour calculer score de confiance basé sur :
    // - Source fiabilité
    // - Complétude des données
    // - Cohérence avec données existantes
    // - Validation croisée multiple sources
    Future.successful(ProposalConfidence(85.0)) // Exemple
  }
}
```

Jour 22-24 : Interface admin propositions

```scala
```

```scala
// app/interfaces/http/api/admin/AdminProposalsController.scala
@Singleton
class AdminProposalsController @Inject()(
  cc: ControllerComponents,
  secured: AdminSecuredAction,
  pendingProposalsHandler: PendingProposalsQueryHandler,
  approveProposalHandler: ApproveProposalCommandHandler,
  rejectProposalHandler: RejectProposalCommandHandler
)(implicit ec: ExecutionContext) extends AbstractController(cc) {

  // GET /api/admin/proposals/pending
  def pendingProposals(domain: Option[String] = None): Action[AnyContent] =
    secured.withPermission(AdminPermission.APPROVE_AI_PROPOSALS).async { req =>
      val query = PendingProposalsQuery(
        domain = domain.map(DiscoveryTarget.valueOf),
        page = 0,
        size = 50
      )
      pendingProposalsHandler.handle(query).map {
        case Right(proposals) => Ok(Json.toJson(proposals))
        case Left(error) => BadRequest(Json.obj("error" -> error.message))
      }
    }

  // POST /api/admin/proposals/:id/approve
  def approveProposal(id: String): Action[JsValue] =
    secured.withPermission(AdminPermission.APPROVE_AI_PROPOSALS)(parse.json).async { req =>
      req.body.validate[ApproveProposalRequest].fold(
        errors => Future.successful(BadRequest(JsError.toJson(errors))),
        approveReq => {
          val command = ApproveProposalCommand(
            proposalId = ProposalId(id),
            adminId = req.admin.id,
            comment = approveReq.comment,
            modifications = approveReq.modifications
          )

          approveProposalHandler.handle(command).map {
            case Right(_) => Ok(Json.obj("status" -> "approved"))
            case Left(error) => BadRequest(Json.obj("error" -> error.message))
          }
        }
      )
    }
}
```

📊 PHASE 4 : AUTRES DOMAINES (Semaine 7-10)

Semaine 7 : Domaine Malts

- Modèle `MaltAggregate` avec spécificités (couleur, extrait, pouvoir diastasique)
- IA découverte malteries (Weyermann, Crisp, Château, etc.)
- API publique/admin malts

Semaine 8 : Domaine Levures

- Modèle `YeastAggregate` avec souches, atténuation, température
- IA découverte laboratoires (White Labs, Wyeast, Lallemand)
- API publique/admin levures

Semaine 9-10 : Référentiels sécurisés

- BeerStyles avec données BJCP
- Origins avec régions productrices
- Aromas avec roue aromatique
- IA découverte mises à jour BJCP, nouvelles régions

🚀 PHASE 5 : RECETTES ET DASHBOARD (Semaine 11-12)

Semaine 11 : Domaine Recettes

- `RecipeAggregate` avec ingrédients et procédures
- Calculs automatiques (IBU, SRM, ABV)
- Scaling par volume (5L, 10L, 20L, 40L)

Semaine 12 : Dashboard et Analytics

- Interface admin complète
- Métriques découverte IA
- Dashboard utilisateur recettes

⚙️ SCRIPTS D'AIDE POUR CHAQUE PHASE

Script de vérification Phase 1

```bash
```

```bash
#!/bin/bash
# scripts/verify-phase1.sh

echo "🔍 Vérification Phase 1..."

# Vérifier compilation
echo "Compilation..."
if sbt compile; then
  echo "✅ Compilation OK"
else
  echo "❌ Erreur compilation"
  exit 1
fi

# Vérifier tests
echo "Tests..."
if sbt "testOnly *shared* *admin*"; then
  echo "✅ Tests Phase 1 OK"
else
  echo "❌ Tests échouent"
  exit 1
fi

# Vérifier base données
echo "Base de données..."
if docker-compose ps | grep -q postgres; then
  echo "✅ PostgreSQL actif"
else
  echo "❌ PostgreSQL non démarré"
  exit 1
fi

echo "🎉 Phase 1 validée !"
```

Script de migration données existantes

bash

```bash
#!/bin/bash
# scripts/migrate-existing-data.sh

echo "📦 Migration données existantes..."

# Sauvegarder données actuelles
if [ -f "conf/reseed/hops.csv" ]; then
  echo "Migration houblons..."
  # Script conversion format ancien -> nouveau
  python3 scripts/convert-hops-data.py conf/reseed/hops.csv conf/reseed/ingredients/hops.csv
fi

echo "✅ Migration terminée"
```

🎯 MÉTRIQUES DE SUCCÈS PAR PHASE

Phase 1 : Fondations

- ✅ Value Objects compilent et testés
- ✅ Admin peut se connecter avec permissions
- ✅ Base données avec audit fonctionnelle
- ✅ Actions sécurisées testées

Phase 2 : Domaine Hops

- ✅ 100+ houblons en base via API admin
- ✅ API publique retourne données correctes
- ✅ Filtrage avancé fonctionnel
- ✅ Audit trail des modifications

Phase 3 : IA Hops

- ✅ ≥5 nouvelles découvertes/jour
- ✅ Score confiance >80% propositions
- ✅ Interface admin révision fluide
- ✅ 0 faux positifs majeurs

Phases suivantes

- Métriques similaires pour malts, levures, référentiels
- Performance API <200ms
- 99% uptime système

- Satisfaction admin >90%

Cette approche progressive vous permet de valider chaque étape et d'ajuster si nécessaire avant de continuer ! 🚀