// This is your Prisma schema file,

// learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {

  provider = "prisma-client-js"

  output   = "../lib/generated/prisma"

}

datasource db {

  provider = "postgresql"

  url      = env("DATABASE\_URL")

}

////////////////////////

/////  auth model  /////

////////////////////////

enum KycStatus {

  NONE

  PENDING

  VERIFIED

  REJECTED

}

model User {

  id             String    @id @default(cuid())

  name           String?

  email          String    @unique

  emailVerified  Boolean   @default(false)

  image          String?

  createdAt      DateTime  @default(now())

  updatedAt      DateTime  @updatedAt

  roles          Role[]

  kycStatus      KycStatus @default(NONE)

  // Authentication

  accounts       Account[]

  sessions       Session[]

  // Profile

  profile        Profile?

  addresses      Address[]

  // Content

  posts          Post[]

  comments       Comment[]

  reactions      Reaction[]

  reviewAssignments ReviewAssignment[]

  // Projects & Funding

  ownedProjects  Project[]  @relation("ProjectOwner")

  projectMemberships ProjectMember[]

  donations      ProjectDonation[]

  investments    ProjectInvestment[]

  loans          ProjectLoan[]

  pledges        Pledge[]

  // Circles & Community

  ownedCircles   Circle[]   @relation("CircleOwner")

  circleMemberships CircleMember[]

  subscriptions  Subscription[]

  // Legal & Compliance

  legalEntities  LegalEntity[]

  // Wallet & Transactions

  wallets        Wallet[]

  transactions   Transaction[]

  // Gamification

  userXP         UserXP?

  userBadges     UserBadge[]

  xpEvents       XPEvent[]

  @@map("users")

}

model Session {

  id         String   @id @default(cuid())

  userId     String

  user       User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  token      String   @unique

  expiresAt  DateTime

  ipAddress  String?

  userAgent  String?

  createdAt  DateTime @default(now())

  updatedAt  DateTime @updatedAt

  @@map("sessions")

}

model Account {

  id                        String    @id @default(cuid())

  userId                    String

  user                      User      @relation(fields: [userId], references: [id], onDelete: Cascade)

  accountId                 String

  providerId                String

  accessToken               String?

  refreshToken              String?

  accessTokenExpiresAt      DateTime?

  refreshTokenExpiresAt     DateTime?

  scope                     String?

  idToken                   String?

  password                  String?

  createdAt                 DateTime  @default(now())

  updatedAt                 DateTime  @updatedAt

  @@unique([providerId, accountId])

  @@map("accounts")

}

model Verification {

  id         String    @id @default(cuid())

  identifier String

  value      String

  expiresAt  DateTime

  createdAt  DateTime  @default(now())

  updatedAt  DateTime  @updatedAt

  @@map("verifications")

}

enum Role {

  USER

  ADMIN

  DEV

  AUTHOR

  MODERATOR

  OWNER

  MEMBER

  VISITOR

}

///////////////////////////

/////   user models   /////

///////////////////////////

model Profile {

  id             String   @id @default(cuid())

  userId         String   @unique @map("user\_id")

  age            Int?

  profession     String?

  motivation     String?

  interests      String[]

  experience     String?

  disponibility  String?

  bio            String?

  firstName      String?  @map("first\_name")

  lastName       String?  @map("last\_name")

  phone          String?

  phoneVerified  Boolean  @default(false) @map("phone\_verified")

  user           User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@map("profiles")

}

model Address {

  id          String  @id @default(cuid())

  userId      String  @map("user\_id")

  street      String?

  city        String?

  state       String?

  postalCode  String? @map("postal\_code")

  country     String?

  user        User    @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@map("addresses")

}

///////////////////////

///// blog models /////

///////////////////////

enum PostStatus {

  DRAFT

  UNDER\_REVIEW

  PUBLISHED

  SCHEDULED

  ARCHIVED

}

enum PostVisibility {

  PUBLIC

  UNLISTED

  PRIVATE

}

enum MediaType {

  IMAGE

  VIDEO

  AUDIO

  FILE

  EMBED

  CODE

}

enum ReactionType {

  LIKE

  CLAP

  UPVOTE

  DOWNVOTE

  LOVE

}

enum CommentStatus {

  VISIBLE

  HIDDEN

  SPAM

  DELETED

}

model Post {

  id             String         @id @default(cuid())

  slug           String         @unique

  title          String

  subtitle       String?

  content        Json?

  contentText    String?        @map("content\_text")

  excerpt        String?

  coverImageUrl  String?        @map("cover\_image\_url")

  canonicalUrl   String?        @map("canonical\_url")

  ogImageUrl     String?        @map("og\_image\_url")

  status         PostStatus     @default(DRAFT)

  visibility     PostVisibility @default(PUBLIC)

  publishedAt    DateTime?      @map("published\_at")

  scheduledAt    DateTime?      @map("scheduled\_at")

  readingTime    Int?           @map("reading\_time")

  wordCount      Int?           @map("word\_count")

  language       String?

  // Author

  authorId       String         @map("author\_id")

  author         User           @relation(fields: [authorId], references: [id], onDelete: Cascade)

  // Threading

  parentId       String?        @map("parent\_id")

  parent         Post?          @relation("PostReplies", fields: [parentId], references: [id])

  children       Post[]         @relation("PostReplies")

  // Media

  attachments    MediaAsset[]

  codeSnippets   CodeSnippet[]

  // Taxonomy

  tags           PostsOnTags[]

  categories     PostsOnCategories[]

  // Interactions

  comments       Comment[]

  reactions      Reaction[]

  // Project association

  projectId      String?        @map("project\_id")

  project        Project?       @relation(fields: [projectId], references: [id])

  // Circle association

  circleId       String?        @map("circle\_id")

  circle         Circle?        @relation(fields: [circleId], references: [id])

  // Peer review

  reviewAssignments ReviewAssignment[]

  createdAt      DateTime       @default(now()) @map("created\_at")

  updatedAt      DateTime       @updatedAt      @map("updated\_at")

  @@index([authorId, status, publishedAt])

  @@index([slug])

  @@index([circleId, visibility])

  @@map("posts")

}

model MediaAsset {

  id         String   @id @default(cuid())

  postId     String   @map("post\_id")

  post       Post     @relation(fields: [postId], references: [id], onDelete: Cascade)

  type       MediaType

  url        String

  provider   String?

  title      String?

  alt        String?

  caption    String?

  position   Int?

  mimeType   String?  @map("mime\_type")

  sizeBytes  Int?     @map("size\_bytes")

  width      Int?

  height     Int?

  durationMs Int?     @map("duration\_ms")

  meta       Json?

  createdAt  DateTime @default(now()) @map("created\_at")

  updatedAt  DateTime @updatedAt      @map("updated\_at")

  @@index([postId, type, position])

  @@map("media\_assets")

}

model CodeSnippet {

  id               String   @id @default(cuid())

  postId           String   @map("post\_id")

  post             Post     @relation(fields: [postId], references: [id], onDelete: Cascade)

  title            String?

  language         String

  filename         String?

  code             String

  highlightedHtml  String?  @map("highlighted\_html")

  createdAt        DateTime @default(now()) @map("created\_at")

  updatedAt        DateTime @updatedAt      @map("updated\_at")

  @@index([postId, language])

  @@map("code\_snippets")

}

model Tag {

  id          String          @id @default(cuid())

  name        String          @unique

  slug        String          @unique

  description String?

  posts       PostsOnTags[]

  createdAt   DateTime        @default(now()) @map("created\_at")

  updatedAt   DateTime        @updatedAt      @map("updated\_at")

  @@map("tags")

}

model PostsOnTags {

  postId     String   @map("post\_id")

  tagId      String   @map("tag\_id")

  post       Post     @relation(fields: [postId], references: [id], onDelete: Cascade)

  tag        Tag      @relation(fields: [tagId], references: [id], onDelete: Cascade)

  assignedAt DateTime @default(now()) @map("assigned\_at")

  assignedBy String?  @map("assigned\_by")

  @@id([postId, tagId])

  @@map("posts\_on\_tags")

}

model Category {

  id          String           @id @default(cuid())

  name        String           @unique

  slug        String           @unique

  description String?

  parentId    String?          @map("parent\_id")

  parent      Category?        @relation("CategoryHierarchy", fields: [parentId], references: [id])

  children    Category[]       @relation("CategoryHierarchy")

  posts       PostsOnCategories[]

  createdAt   DateTime         @default(now()) @map("created\_at")

  updatedAt   DateTime         @updatedAt      @map("updated\_at")

  @@map("categories")

}

model PostsOnCategories {

  postId     String   @map("post\_id")

  categoryId String   @map("category\_id")

  post       Post     @relation(fields: [postId], references: [id], onDelete: Cascade)

  category   Category @relation(fields: [categoryId], references: [id], onDelete: Cascade)

  position   Int?

  assignedAt DateTime @default(now()) @map("assigned\_at")

  @@id([postId, categoryId])

  @@map("posts\_on\_categories")

}

model Comment {

  id          String        @id @default(cuid())

  postId      String        @map("post\_id")

  post        Post          @relation(fields: [postId], references: [id], onDelete: Cascade)

  authorId    String        @map("author\_id")

  author      User          @relation(fields: [authorId], references: [id], onDelete: Cascade)

  content     Json?

  contentText String?       @map("content\_text")

  status      CommentStatus @default(VISIBLE)

  parentId    String?       @map("parent\_id")

  parent      Comment?      @relation("CommentThread", fields: [parentId], references: [id])

  children    Comment[]     @relation("CommentThread")

  createdAt   DateTime      @default(now()) @map("created\_at")

  updatedAt   DateTime      @updatedAt      @map("updated\_at")

  @@index([postId, parentId])

  @@map("comments")

}

model Reaction {

  id        String       @id @default(cuid())

  postId    String       @map("post\_id")

  post      Post         @relation(fields: [postId], references: [id], onDelete: Cascade)

  userId    String       @map("user\_id")

  user      User         @relation(fields: [userId], references: [id], onDelete: Cascade)

  type      ReactionType @default(LIKE)

  createdAt DateTime     @default(now()) @map("created\_at")

  @@unique([postId, userId, type])

  @@map("reactions")

}

///////////////////////////////

/////  peer review (blog)  /////

///////////////////////////////

model ReviewAssignment {

  id         String   @id @default(cuid())

  postId     String   @map("post\_id")

  reviewerId String   @map("reviewer\_id")

  status     String   @default("pending")

  score      Int?

  feedback   String?

  assignedAt DateTime @default(now()) @map("assigned\_at")

  reviewedAt DateTime? @map("reviewed\_at")

  post       Post     @relation(fields: [postId], references: [id], onDelete: Cascade)

  reviewer   User     @relation(fields: [reviewerId], references: [id], onDelete: Cascade)

  @@unique([postId, reviewerId])

  @@map("review\_assignments")

}

//////////////////////

///// FAQ models /////

//////////////////////

model FAQ {

  id        String   @id @default(cuid())

  question  String

  answer    String

  category  String? /// category of the FAQ : "general", "The Projects", "The Blog" "the Circles" "other")

  rating    Int?     @default(0)

  createdAt DateTime @default(now()) @map("created\_at")

  updatedAt DateTime @updatedAt      @map("updated\_at")

  @@map("faqs")

}

//////////////////////////

///// project models /////

//////////////////////////

enum ProjectType {

  DONATION

  REWARD

  EQUITY

  CROWDLENDING

  KITTY

  IMPACT\_SPRINT

}

enum ProjectStatus {

  DRAFT

  PENDING

  ACTIVE

  SUCCESSFUL

  FAILED

  CANCELLED

  SUSPENDED

}

enum ProjectCategory {

  ECOLOGY

  SOCIAL

  EDUCATION

  TECHNOLOGY

  HEALTH

  CULTURE

  INNOVATION

  OTHER

}

enum ProjectImpactType {

  ENVIRONMENTAL

  SOCIAL

  ECONOMIC

}

model Project {

  id               String   @id @default(cuid())

  name             String

  slug             String   @unique

  description      String?

  shortDescription String?  @map("short\_description")

  // General info

  type            ProjectType

  status          ProjectStatus @default(DRAFT)

  category        ProjectCategory

  location        String?

  // Funding

  fundingGoal     Float    @map("funding\_goal")

  amountRaised    Float    @map("amount\_raised") @default(0)

  currency        String   @default("EUR")

  // Timeline

  startDate       DateTime? @map("start\_date")

  endDate         DateTime? @map("end\_date")

  duration        Int?

  // Project owner

  ownerId         String    @map("owner\_id")

  owner           User      @relation("ProjectOwner", fields: [ownerId], references: [id], onDelete: Cascade)

  organization    String?

  legalStatus     String?   @map("legal\_status")

  // Legal entity

  legalEntityId   String?   @map("legal\_entity\_id")

  legalEntity     LegalEntity? @relation(fields: [legalEntityId], references: [id])

  // Contact

  website         String?

  socialMedia     Json?     @map("social\_media")

  contactEmail    String?   @map("contact\_email")

  contactPhone    String?   @map("contact\_phone")

  // Impact

  impactTypes        ProjectImpactType[] @map("impact\_types")

  impactDescription  String?            @map("impact\_description")

  impactIndicators   String[]           @map("impact\_indicators")

  sdgAlignment       Int[]              @map("sdg\_alignment")

  // Media

  images         String[]

  videos         String[]

  documents      String[]

  // Settings

  visibility        PostVisibility @default(PUBLIC)

  allowComments     Boolean        @map("allow\_comments") @default(true)

  enableImpactSprint Boolean       @map("enable\_impact\_sprint") @default(false)

  // Relations

  members       ProjectMember[]

  posts         Post[]

  updates       ProjectUpdate[]

  rewards       ProjectReward[]

  donations     ProjectDonation[]

  investments   ProjectInvestment[]

  loans         ProjectLoan[]

  wallets       Wallet[]

  // Financial data

  valuation         Float?

  minInvestment     Float? @map("min\_investment")

  capitalPercentage Float? @map("capital\_percentage")

  expectedInvestors Int?   @map("expected\_investors")

  taxAdvantages     String? @map("tax\_advantages")

  // Lending data

  interestRate      Float?  @map("interest\_rate")

  loanDuration      Int?    @map("loan\_duration")

  repaymentFrequency String? @map("repayment\_frequency")

  minLoanAmount     Float?  @map("min\_loan\_amount")

  guarantees        String?

  // Impact Sprint

  sprintProjects    SprintProject[]

  pledgeSelections  PledgeSelection[]

  createdAt     DateTime @default(now()) @map("created\_at")

  updatedAt     DateTime @updatedAt      @map("updated\_at")

  @@index([ownerId])

  @@index([status])

  @@index([type])

  @@index([category])

  @@map("projects")

  Sprint Sprint[]

  Transaction Transaction[]

  CircleProject CircleProject[]

}

model ProjectMember {

  id        String   @id @default(cuid())

  projectId String   @map("project\_id")

  userId    String   @map("user\_id")

  role      String

  joinedAt  DateTime @default(now()) @map("joined\_at")

  project   Project  @relation(fields: [projectId], references: [id], onDelete: Cascade)

  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([projectId, userId])

  @@map("project\_members")

}

model ProjectUpdate {

  id        String   @id @default(cuid())

  projectId String   @map("project\_id")

  title     String

  content   String

  images    String[]

  createdAt DateTime @default(now()) @map("created\_at")

  updatedAt DateTime @updatedAt @map("updated\_at")

  project   Project  @relation(fields: [projectId], references: [id], onDelete: Cascade)

  @@map("project\_updates")

}

model ProjectReward {

  id          String   @id @default(cuid())

  projectId   String   @map("project\_id")

  amount      Float

  title       String

  description String

  image       String?

  deliveryDate DateTime? @map("delivery\_date")

  quantity    Int?

  claimed     Int       @default(0)

  project     Project   @relation(fields: [projectId], references: [id], onDelete: Cascade)

  donations   ProjectDonation[]

  @@map("project\_rewards")

}

model ProjectDonation {

  id          String   @id @default(cuid())

  projectId   String   @map("project\_id")

  userId      String   @map("user\_id")

  amount      Float

  isAnonymous Boolean  @default(false) @map("is\_anonymous")

  message     String?

  rewardId    String?  @map("reward\_id")

  status      String   @default("completed")

  createdAt   DateTime @default(now()) @map("created\_at")

  project     Project  @relation(fields: [projectId], references: [id], onDelete: Cascade)

  user        User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  reward      ProjectReward? @relation(fields: [rewardId], references: [id])

  @@map("project\_donations")

}

model ProjectInvestment {

  id        String   @id @default(cuid())

  projectId String   @map("project\_id")

  userId    String   @map("user\_id")

  amount    Float

  shares    Float?

  status    String   @default("pending")

  createdAt DateTime @default(now()) @map("created\_at")

  project   Project  @relation(fields: [projectId], references: [id], onDelete: Cascade)

  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@map("project\_investments")

}

model ProjectLoan {

  id        String   @id @default(cuid())

  projectId String   @map("project\_id")

  userId    String   @map("user\_id")

  amount    Float

  status    String   @default("active")

  createdAt DateTime @default(now()) @map("created\_at")

  project   Project  @relation(fields: [projectId], references: [id], onDelete: Cascade)

  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@map("project\_loans")

}

/////////////////////////

///// Circle models /////

/////////////////////////

enum CircleType {

  THEMATIC

  SUPPORT

  SYNERGY

}

enum CircleMemberRole {

  OWNER

  ADMIN

  MEMBER

  FACILITATOR

  SECRETARY

  PROJECT\_REFERENT

  ETHICS\_GUARDIAN

}

enum CircleParticipationMode {

  SYNCHRONOUS

  ASYNCHRONOUS

}

model Circle {

  id                String   @id @default(cuid())

  name              String

  slug              String   @unique

  description       String?

  type              CircleType

  theme             String?

  participationMode CircleParticipationMode @map("participation\_mode")

  // Configuration

  maxMembers        Int?     @map("max\_members")

  isPublic          Boolean  @map("is\_public") @default(true)

  isActive          Boolean  @map("is\_active") @default(true)

  // Financial management

  monthlyFee        Float?   @map("monthly\_fee") @default(10)

  funds             Float    @default(0)

  allocatedFunds    Float    @map("allocated\_funds") @default(0)

  // Relations

  creatorId         String   @map("creator\_id")

  creator           User     @relation("CircleOwner", fields: [creatorId], references: [id], onDelete: Cascade)

  members           CircleMember[]

  circleProjects    CircleProject[]

  posts             Post[]

  subscriptions     Subscription[]

  allocations       Allocation[]

  wallets           Wallet[]

  transactions      Transaction[]

  createdAt         DateTime @default(now()) @map("created\_at")

  updatedAt         DateTime @updatedAt @map("updated\_at")

  @@index([type])

  @@index([creatorId])

  @@map("circles")

}

model CircleMember {

  id        String   @id @default(cuid())

  circleId  String   @map("circle\_id")

  userId    String   @map("user\_id")

  role      CircleMemberRole

  joinedAt  DateTime @default(now()) @map("joined\_at")

  xp        Int      @default(0)

  level     Int      @default(1)

  circle    Circle   @relation(fields: [circleId], references: [id], onDelete: Cascade)

  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@unique([circleId, userId])

  @@map("circle\_members")

}

model CircleProject {

  id              String   @id @default(cuid())

  circleId        String   @map("circle\_id")

  projectId       String   @map("project\_id")

  allocatedAmount Float    @map("allocated\_amount") @default(0)

  supportType     String   @map("support\_type")

  votedAt         DateTime? @map("voted\_at")

  status          String   @default("pending")

  circle          Circle   @relation(fields: [circleId], references: [id], onDelete: Cascade)

  project         Project  @relation(fields: [projectId], references: [id], onDelete: Cascade)

  @@unique([circleId, projectId])

  @@map("circle\_projects")

}

//////////////////////////////

/////  payment models    /////

//////////////////////////////

model Transaction {

  id          String   @id @default(cuid())

  userId      String   @map("user\_id")

  type        String

  amount      Float

  currency    String   @default("EUR")

  status      String   @default("pending")

  description String?

  projectId   String?  @map("project\_id")

  circleId    String?  @map("circle\_id")

  metadata    Json?

  createdAt   DateTime @default(now()) @map("created\_at")

  updatedAt   DateTime @updatedAt @map("updated\_at")

  user        User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  project     Project? @relation(fields: [projectId], references: [id])

  circle      Circle?  @relation(fields: [circleId], references: [id])

  allocations Allocation[]

  @@index([type, status, createdAt])

  @@map("transactions")

}

///////////////////////////////

/////  loyalty models     /////

///////////////////////////////

model UserXP {

  id           String   @id @default(cuid())

  userId       String   @unique @map("user\_id")

  totalXP      Int      @default(0) @map("total\_xp")

  level        Int      @default(1)

  impactRank   String?  @map("impact\_rank")

  donationXP   Int      @default(0) @map("donation\_xp")

  investmentXP Int      @default(0) @map("investment\_xp")

  contentXP    Int      @default(0) @map("content\_xp")

  communityXP  Int      @default(0) @map("community\_xp")

  user         User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  createdAt    DateTime @default(now()) @map("created\_at")

  updatedAt    DateTime @updatedAt @map("updated\_at")

  @@map("user\_xp")

}

enum XPEventType {

  READ

  SHARE

  COMMENT

  REVIEW

  PLEDGE

  WIN

  BADGE

  SPEND

}

model XPEvent {

  id         String      @id @default(cuid())

  userId     String      @map("user\_id")

  type       XPEventType

  delta      Int

  sourceType String?     @map("source\_type")

  sourceId   String?     @map("source\_id")

  metadata   Json?

  createdAt  DateTime    @default(now()) @map("created\_at")

  user       User        @relation(fields: [userId], references: [id], onDelete: Cascade)

  @@index([userId, createdAt])

  @@map("xp\_events")

}

model Badge {

  id          String   @id @default(cuid())

  name        String

  description String?

  image       String?

  criteria    Json?

  userBadges  UserBadge[]

  createdAt   DateTime @default(now()) @map("created\_at")

  @@map("badges")

}

model UserBadge {

  id        String   @id @default(cuid())

  userId    String   @map("user\_id")

  badgeId   String   @map("badge\_id")

  earnedAt  DateTime @default(now()) @map("earned\_at")

  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  badge     Badge    @relation(fields: [badgeId], references: [id], onDelete: Cascade)

  @@unique([userId, badgeId])

  @@map("user\_badges")

}

////////////////////////////////

/////  compliance & flags  /////

////////////////////////////////

model LegalEntity {

  id                 String   @id @default(cuid())

  type               String

  name               String?

  country            String?

  registrationNumber String?  @map("registration\_number")

  taxId              String?  @map("tax\_id")

  contactEmail       String?  @map("contact\_email")

  contactPhone       String?  @map("contact\_phone")

  ownerId            String?  @map("owner\_id")

  owner              User?    @relation(fields: [ownerId], references: [id], onDelete: SetNull)

  projects           Project[]

  createdAt          DateTime @default(now()) @map("created\_at")

  updatedAt          DateTime @updatedAt @map("updated\_at")

  @@map("legal\_entities")

}

model FeatureFlag {

  id        String   @id @default(cuid())

  key       String   @unique

  enabled   Boolean  @default(false)

  rollout   Json?

  createdAt DateTime @default(now()) @map("created\_at")

  updatedAt DateTime @updatedAt @map("updated\_at")

  @@map("feature\_flags")

}

////////////////////////////////

/////  subscriptions/split /////

////////////////////////////////

model Subscription {

  id            String   @id @default(cuid())

  userId        String   @map("user\_id")

  circleId      String   @map("circle\_id")

  amount        Float    @default(10)

  currency      String   @default("EUR")

  status        String   @default("active")

  startedAt     DateTime @default(now()) @map("started\_at")

  nextBillingAt DateTime? @map("next\_billing\_at")

  cancelledAt   DateTime? @map("cancelled\_at")

  user          User     @relation(fields: [userId], references: [id], onDelete: Cascade)

  circle        Circle   @relation(fields: [circleId], references: [id], onDelete: Cascade)

  @@unique([userId, circleId])

  @@map("subscriptions")

}

model Allocation {

  id              String     @id @default(cuid())

  circleId        String     @map("circle\_id")

  transactionId   String     @map("transaction\_id")

  operatingAmount Float      @map("operating\_amount")

  projectAmount   Float      @map("project\_amount")

  period          String?

  createdAt       DateTime   @default(now()) @map("created\_at")

  circle          Circle     @relation(fields: [circleId], references: [id], onDelete: Cascade)

  transaction     Transaction @relation(fields: [transactionId], references: [id], onDelete: Cascade)

  @@map("allocations")

}

/////////////////////////

///// Impact Sprint  ////

/////////////////////////

enum SprintStatus {

  DRAFT

  UPCOMING

  RUNNING

  SETTLED

  CANCELLED

}

enum PledgeStatus {

  PROMISED

  SETTLED

  CANCELLED

  EXPIRED

}

model Sprint {

  id               String       @id @default(cuid())

  name             String

  slug             String       @unique

  theme            String?

  startAt          DateTime?    @map("start\_at")

  endAt            DateTime?    @map("end\_at")

  status           SprintStatus @default(DRAFT)

  winnerProjectId  String?      @map("winner\_project\_id")

  winnerProject    Project?     @relation(fields: [winnerProjectId], references: [id])

  sprintProjects   SprintProject[]

  pledges          Pledge[]

  createdAt        DateTime     @default(now()) @map("created\_at")

  updatedAt        DateTime     @updatedAt @map("updated\_at")

  @@map("sprints")

}

model SprintProject {

  sprintId  String

  projectId String

  position  Int?

  sprint    Sprint  @relation(fields: [sprintId], references: [id], onDelete: Cascade)

  project   Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

  @@id([sprintId, projectId])

  @@map("sprint\_projects")

}

model Pledge {

  id               String       @id @default(cuid())

  sprintId         String       @map("sprint\_id")

  userId           String       @map("user\_id")

  amount           Float

  status           PledgeStatus @default(PROMISED)

  settledProjectId String?      @map("settled\_project\_id")

  settledAt        DateTime?    @map("settled\_at")

  sprint           Sprint       @relation(fields: [sprintId], references: [id], onDelete: Cascade)

  user             User         @relation(fields: [userId], references: [id], onDelete: Cascade)

  pledgeSelections PledgeSelection[]

  createdAt        DateTime     @default(now()) @map("created\_at")

  @@index([sprintId, userId])

  @@map("pledges")

}

model PledgeSelection {

  pledgeId  String

  projectId String

  pledge    Pledge  @relation(fields: [pledgeId], references: [id], onDelete: Cascade)

  project   Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

  @@id([pledgeId, projectId])

  @@map("pledge\_selections")

}

////////////////////////

///// Wallet models ////

////////////////////////

enum WalletLedgerType {

  DEPOSIT

  WITHDRAW

  PROMISE\_LOCK

  PROMISE\_RELEASE

  PLEDGE\_SETTLE

  REFUND

  SUBSCRIPTION\_FEE

  ALLOCATION

  TRANSFER\_IN

  TRANSFER\_OUT

}

enum WalletKind {

  USER

  CIRCLE

  PROJECT

}

model Wallet {

  id         String    @id @default(cuid())

  userId     String?   @map("user\_id")

  circleId   String?   @map("circle\_id")

  projectId  String?   @map("project\_id")

  ownerKind  WalletKind

  currency   String    @default("EUR")

  chain      String?

  asset      String?

  address    String?

  available  Float     @default(0)

  locked     Float     @default(0)

  user       User?     @relation(fields: [userId], references: [id], onDelete: Cascade)

  circle     Circle?   @relation(fields: [circleId], references: [id], onDelete: Cascade)

  project    Project?  @relation(fields: [projectId], references: [id], onDelete: Cascade)

  ledger     WalletLedger[]

  createdAt  DateTime  @default(now()) @map("created\_at")

  updatedAt  DateTime  @updatedAt @map("updated\_at")

  @@index([userId])

  @@index([circleId])

  @@index([projectId])

  @@map("wallets")

}

model WalletLedger {

  id            String           @id @default(cuid())

  walletId      String           @map("wallet\_id")

  type          WalletLedgerType

  amount        Float

  balanceAfter  Float            @map("balance\_after")

  referenceType String?          @map("reference\_type")

  referenceId   String?          @map("reference\_id")

  metadata      Json?

  createdAt     DateTime         @default(now()) @map("created\_at")

  wallet        Wallet           @relation(fields: [walletId], references: [id], onDelete: Cascade)

  @@index([walletId, createdAt])

  @@map("wallet\_ledgers")

}

**⚙️ Partie 1 — Authentification et Utilisateurs**

**### 1.1 Modèle User**

**📘 Description**

Le modèle **User** est la pierre angulaire du système.  
Il représente tout **utilisateur inscrit sur la plateforme**, qu’il s’agisse d’un contributeur, d’un porteur de projet, d’un membre de cercle ou d’un administrateur.

Il contient les informations générales d’un compte, ainsi que les **liens vers toutes les autres entités** (profils, transactions, projets, etc.).

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique généré automatiquement (cuid()). |
| name | String? | Nom d’affichage (optionnel). |
| email | String | Adresse email unique, utilisée pour la connexion et la vérification. |
| emailVerified | Boolean | Indique si l’email a été vérifié. |
| image | String? | URL de l’image de profil. |
| createdAt | DateTime | Date de création du compte. |
| updatedAt | DateTime | Mise à jour automatique. |
| roles | Role[] | Rôles attribués (ex: USER, ADMIN, AUTHOR etc.). |
| kycStatus | KycStatus | Statut de vérification d’identité (KYC). |

**Relations principales**

* accounts, sessions → authentification OAuth ou locale
* profile, addresses → informations personnelles
* posts, comments, reactions → contenu produit
* ownedProjects, projectMemberships → participation aux projets
* wallets, transactions → gestion financière
* userXP, userBadges → gamification

**💡 Cas d’usage**

* **Inscription d’un utilisateur** : un User est créé avec un rôle par défaut USER et emailVerified = false.
* **Vérification KYC** : kycStatus passe de PENDING à VERIFIED après validation d’identité.
* **Multi-rôle** : un utilisateur peut être à la fois ADMIN d’un cercle et AUTHOR d’un post.

**### 1.2 Modèle Session**

**📘 Description**

Gère les **sessions de connexion** actives d’un utilisateur, avec les informations de sécurité associées (token, IP, expiration).

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique de la session. |
| userId | String | Lien vers User. |
| token | String | Jeton unique de session. |
| expiresAt | DateTime | Date d’expiration du jeton. |
| ipAddress | String? | Adresse IP du client. |
| userAgent | String? | Informations sur le navigateur ou appareil. |
| createdAt | DateTime | Création. |
| updatedAt | DateTime | Dernière activité. |

**💡 Cas d’usage**

* Authentification persistante (connexion via cookie ou JWT).
* Déconnexion → suppression de la session.
* Gestion de la sécurité (liste des appareils connectés).

**### 1.3 Modèle Account**

**📘 Description**

Stocke les **comptes d’authentification externes** (Google, GitHub, etc.) ou internes (email/mot de passe).  
Permet la connexion multi-fournisseurs.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant du compte. |
| userId | String | Lien vers User. |
| accountId | String | Identifiant de l’utilisateur sur le fournisseur externe. |
| providerId | String | Nom du fournisseur (google, github, local). |
| accessToken, refreshToken | String? | Jetons d’accès OAuth. |
| accessTokenExpiresAt | DateTime? | Expiration du jeton. |
| password | String? | Mot de passe haché (pour les connexions locales). |

**💡 Cas d’usage**

* **Connexion OAuth Google** → création d’un Account avec providerId="google".
* **Connexion par mot de passe** → providerId="local" avec champ password défini.
* **Rafraîchissement automatique** → refreshToken mis à jour périodiquement.

**### 1.4 Modèle Verification**

**📘 Description**

Gère les **liens ou codes de vérification temporaires**, utilisés pour :

* la vérification d’email,
* la réinitialisation de mot de passe,
* la validation KYC ou 2FA.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| identifier | String | Email ou identifiant associé. |
| value | String | Code ou token envoyé. |
| expiresAt | DateTime | Date d’expiration du lien/code. |
| createdAt, updatedAt | DateTime | Suivi temporel. |

**💡 Cas d’usage**

* Envoi d’un lien de confirmation d’email.
* Validation d’un mot de passe oublié.
* Vérification d’un numéro de téléphone.

**### 1.5 Enum Role**

Définit les **rôles d’accès** possibles dans l’application.

| **Valeur** | **Description** |
| --- | --- |
| USER | Utilisateur standard. |
| ADMIN | Administrateur global. |
| DEV | Développeur technique. |
| AUTHOR | Rédacteur de contenu. |
| MODERATOR | Modérateur du blog ou des cercles. |
| OWNER | Propriétaire d’un projet ou d’un cercle. |
| MEMBER | Membre standard d’une entité. |
| VISITOR | Utilisateur non inscrit (lecture seule). |

**### 1.6 Enum KycStatus**

Représente le **niveau de vérification d’identité**.

| **Valeur** | **Description** |
| --- | --- |
| NONE | Aucun processus commencé. |
| PENDING | En attente de validation. |
| VERIFIED | Identité validée. |
| REJECTED | Validation refusée. |

**🧩 Partie 2 — Profils et Adresses**

**### 2.1 Modèle Profile**

**📘 Description**

Complète le modèle User avec des **informations personnelles détaillées** et des préférences.  
Chaque utilisateur a au plus **un seul profil** (1:1).

**🧩 Champs principaux**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| userId | String | Lien unique vers User. |
| age | Int? | Âge de l’utilisateur. |
| profession | String? | Métier actuel. |
| motivation | String? | Motivation à rejoindre la plateforme. |
| interests | String[] | Centres d’intérêt (liste). |
| experience | String? | Expérience professionnelle ou sociale. |
| disponibility | String? | Disponibilité (temps, fréquence). |
| bio | String? | Biographie libre. |
| firstName, lastName | String? | Identité réelle. |
| phone | String? | Numéro de téléphone. |
| phoneVerified | Boolean | Indique si le téléphone est validé. |

**💡 Cas d’usage**

* **Compléter le profil utilisateur** après inscription.
* **Afficher des suggestions de projets ou cercles** basées sur les intérêts.
* **Filtrer les membres d’un cercle** par compétences ou disponibilité.

**### 2.2 Modèle Address**

**📘 Description**

Gère les **adresses physiques** liées à un utilisateur (résidence, facturation, livraison, etc.).

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| street, city, state, postalCode, country | String? | Détails de l’adresse. |
| userId | String | Référence vers User. |

**💡 Cas d’usage**

* Enregistrer l’adresse pour des dons ou livraisons de récompenses.
* Permettre à un projet de localiser ses soutiens par région.
* Utiliser dans la facturation ou les documents légaux.

**📰 Partie 3 — Blog, Contenu et Interactions Sociales**

**### 3.1 Modèle Post**

**📘 Description**

Le modèle Post représente un **contenu publié sur la plateforme** : article de blog, actualité de projet, mise à jour, ou texte communautaire.

Chaque post est **créé par un utilisateur** (authorId) et peut appartenir à **plusieurs catégories** ou **tags**.  
Les posts peuvent aussi être **soumis à révision** (via ReviewAssignment).

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique du post. |
| title | String | Titre de l’article. |
| slug | String | Identifiant lisible dans l’URL (généré à partir du titre). |
| content | String | Contenu HTML ou Markdown du post. |
| excerpt | String? | Résumé ou introduction courte. |
| coverImage | String? | URL d’une image d’illustration. |
| published | Boolean | Indique si le post est visible publiquement. |
| featured | Boolean | Mise en avant sur la page d’accueil. |
| publishedAt | DateTime? | Date de publication effective. |
| authorId | String | Référence vers User. |
| createdAt, updatedAt | DateTime | Suivi des modifications. |

**Relations**

* author → vers User (créateur du contenu).
* tags → via PostTag (relation N:N).
* categories → via PostCategory (relation N:N).
* comments, reactions → interactions des utilisateurs.
* reviewAssignments → système d’évaluation et modération.

**💡 Cas d’usage**

* Un **auteur** publie un article de blog : published = true, publishedAt = now().
* Un **modérateur** attribue une ReviewAssignment avant validation.
* Un **lecteur** ajoute un Comment ou une Reaction (like, dislike, etc.).
* Les posts featured = true sont affichés sur la page d’accueil.

**### 3.2 Modèle Comment**

**📘 Description**

Le modèle Comment permet aux utilisateurs d’**interagir avec un post** ou un projet via des commentaires.  
Il supporte les **réponses imbriquées** (sous-commentaires) via parentCommentId.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant du commentaire. |
| postId | String? | Post associé (optionnel si commentaire sur un autre objet). |
| projectId | String? | Projet associé (si commentaire de projet). |
| authorId | String | Utilisateur auteur du commentaire. |
| parentCommentId | String? | Commentaire parent (si réponse). |
| content | String | Texte du commentaire. |
| createdAt, updatedAt | DateTime | Suivi des modifications. |

**Relations**

* author → vers User.
* post → vers Post.
* project → vers Project.
* parentComment → auto-référence.
* replies → liste des sous-commentaires.

**💡 Cas d’usage**

* **Discussion sous un article** → postId défini, projectId nul.
* **Feedback sur un projet** → projectId défini, postId nul.
* **Réponse à un commentaire** → parentCommentId défini.
* Les commentaires peuvent être **modérés** ou filtrés via un rôle MODERATOR.

**### 3.3 Modèle Reaction**

**📘 Description**

Les **réactions** permettent aux utilisateurs d’exprimer rapidement une émotion ou une opinion sur un post ou un commentaire (like, love, dislike, etc.).

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| userId | String | Utilisateur ayant réagi. |
| postId | String? | Lien vers un Post. |
| commentId | String? | Lien vers un Comment. |
| type | ReactionType | Type de réaction (LIKE, LOVE, etc.). |
| createdAt | DateTime | Date de création. |

**Relations**

* user → auteur de la réaction.
* post ou comment → cible de la réaction.

**💡 Cas d’usage**

* ReactionType = LIKE → “J’aime cet article.”
* ReactionType = DISAGREE → “Je ne suis pas d’accord.”
* Agrégation : calcul du **nombre total de likes** d’un post.
* Empêcher les doublons via (userId, postId) unique.

**### 3.4 Modèle Tag**

**📘 Description**

Un **tag** est un mot-clé ou étiquette thématique associé à plusieurs posts.  
Il sert à **organiser le contenu** et faciliter la navigation (recherche, filtres).

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant du tag. |
| name | String | Nom du tag (ex: “innovation”, “durabilité”). |
| slug | String | URL-friendly name. |
| posts | PostTag[] | Relation N:N vers les posts. |

**💡 Cas d’usage**

* Taguer un article avec #FinanceSolidaire, #EnergieVerte.
* Afficher tous les posts ayant un même tag.
* Statistiques d’utilisation des tags pour générer des **nuages de mots**.

**### 3.5 Modèle Category**

**📘 Description**

Les **catégories** structurent les contenus en grands thèmes hiérarchiques (ex: “Actualités”, “Projets”, “Éducation”).  
Chaque Post peut appartenir à plusieurs catégories.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant. |
| name | String | Nom de la catégorie. |
| slug | String | Identifiant d’URL. |
| posts | PostCategory[] | Relation N:N. |

**💡 Cas d’usage**

* Catégoriser les articles pour navigation et SEO.
* Filtrer par catégorie : “Tous les articles dans Économie Sociale”.
* Associer automatiquement les nouvelles publications à une catégorie “Non classé”.

**### 3.6 Modèle ReviewAssignment**

**📘 Description**

Le modèle ReviewAssignment est utilisé pour la **modération** ou la **relecture** d’un contenu avant publication.  
Un utilisateur (“reviewer”) reçoit une mission de validation d’un post.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant. |
| postId | String | Contenu à évaluer. |
| reviewerId | String | Utilisateur chargé de la relecture. |
| status | ReviewStatus | État du processus (PENDING, APPROVED, REJECTED). |
| comments | String? | Remarques de l’évaluateur. |
| createdAt, updatedAt | DateTime | Suivi des actions. |

**💡 Cas d’usage**

* **Flux de validation éditoriale** :
  + L’auteur soumet un article → création d’un ReviewAssignment PENDING.
  + Le relecteur valide → status = APPROVED, publication automatique.
  + En cas de refus → status = REJECTED + comments.
* Permet de construire un **workflow de qualité éditoriale**.

**### 3.7 Enum ReactionType**

| **Valeur** | **Description** |
| --- | --- |
| LIKE | Appréciation positive. |
| LOVE | Soutien enthousiaste. |
| INSIGHTFUL | Réaction valorisant le contenu. |
| DISAGREE | Désaccord ou opinion contraire. |

**### 3.8 Enum ReviewStatus**

| **Valeur** | **Description** |
| --- | --- |
| PENDING | En attente d’évaluation. |
| APPROVED | Validé pour publication. |
| REJECTED | Refusé, à corriger. |

**💬 Exemples d’usage combiné**

**Cas 1 – Publication d’un article :**

sequenceDiagram

User ->> Post: Crée un nouveau post (draft)

Admin ->> ReviewAssignment: Attribue un relecteur

Reviewer ->> ReviewAssignment: Marque comme APPROVED

System ->> Post: Met à jour published = true

Readers ->> Post: Lisent, commentent, réagissent

**Cas 2 – Interaction sociale :**

* Un lecteur commente un article (Comment)
* D’autres répondent en chaîne (arborescence)
* Chaque commentaire peut recevoir des réactions (Reaction)
* Les métriques de popularité (nombre de likes, nombre de vues) alimentent la page d’accueil

**💼 Partie 4 — Projets et Financement Participatif**

**### 4.1 Modèle Project**

**📘 Description**

Le modèle Project est **l’entité centrale** du module de financement participatif.  
Il représente un projet porté par un utilisateur (ou une organisation), qui peut recevoir des dons, investissements, prêts ou engagements.

Un Project est relié à plusieurs autres entités :

* owner (créateur du projet),
* circle (communauté associée),
* donations, investments, loans, rewards,
* comments, tags, reviews,
* transactions financières liées.

**🧩 Champs principaux**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| title | String | Titre du projet. |
| slug | String | Identifiant d’URL. |
| description | String | Description complète du projet. |
| shortDescription | String? | Résumé court. |
| status | ProjectStatus | État du projet (DRAFT, ACTIVE, COMPLETED, etc.). |
| goalAmount | Float? | Objectif financier du projet. |
| collectedAmount | Float | Montant total collecté (mise à jour automatique). |
| currency | String | Devise (ex : "EUR", "USD"). |
| ownerId | String | Référence vers le porteur de projet (User). |
| circleId | String? | Lien optionnel vers un cercle collectif. |
| categoryId | String? | Catégorie principale du projet. |
| startDate, endDate | DateTime? | Période de collecte. |
| coverImage | String? | Image d’illustration. |
| createdAt, updatedAt | DateTime | Suivi temporel. |

**Relations**

* owner → User : créateur du projet.
* circle → Circle : si le projet est communautaire.
* donations, investments, loans, rewards → contributions financières.
* transactions → suivi des flux financiers.
* tags, comments, reviews → interactions sociales.

**💡 Cas d’usage**

* Création d’un **projet participatif** par un utilisateur.
* Gestion de la **progression de la collecte** (collectedAmount / goalAmount).
* Attribution d’un **cercle** pour gouvernance collective.
* Filtrage par **catégorie ou statut** (projets terminés, en cours, etc.).

**### 4.2 Modèle Donation**

**📘 Description**

Le modèle Donation enregistre un **don volontaire** effectué par un utilisateur à un projet.  
C’est le mode de financement **sans contrepartie**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant. |
| projectId | String | Projet bénéficiaire. |
| donorId | String | Utilisateur donateur. |
| amount | Float | Montant du don. |
| currency | String | Devise utilisée. |
| transactionId | String? | Référence de transaction financière. |
| createdAt | DateTime | Date du don. |

**Relations**

* project → Project
* donor → User
* transaction → Transaction (enregistrement financier)

**💡 Cas d’usage**

* **Don unique** à un projet.
* **Page de transparence** listant les dons publics.
* Calcul de la **somme totale des dons** reçus par projet.

**### 4.3 Modèle Investment**

**📘 Description**

Le modèle Investment représente un **investissement participatif** avec un potentiel **rendement financier**.  
Contrairement au don, l’investisseur peut recevoir une part, un dividende ou un remboursement.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant. |
| projectId | String | Projet financé. |
| investorId | String | Utilisateur investisseur. |
| amount | Float | Montant investi. |
| currency | String | Devise. |
| expectedReturn | Float? | Rendement espéré (% ou montant). |
| maturityDate | DateTime? | Date prévue de remboursement. |
| transactionId | String? | Référence transactionnelle. |
| createdAt | DateTime | Création. |

**💡 Cas d’usage**

* **Investissement dans un projet à impact** : énergie solaire, coopérative, etc.
* Affichage du **rendement potentiel** dans le tableau de bord utilisateur.
* Gestion du **remboursement** via Transaction.

**### 4.4 Modèle Loan**

**📘 Description**

Un Loan (prêt) est un **mécanisme de financement remboursable**.  
Un utilisateur ou cercle prête une somme à un projet, avec des conditions (taux, échéances).

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant. |
| projectId | String | Projet emprunteur. |
| lenderId | String | Prêteur (User). |
| amount | Float | Montant prêté. |
| interestRate | Float? | Taux d’intérêt. |
| dueDate | DateTime? | Date d’échéance du remboursement. |
| transactionId | String? | Lien vers Transaction. |
| createdAt | DateTime | Création du prêt. |

**💡 Cas d’usage**

* **Prêt sans intérêt** entre membres d’un cercle.
* **Financement remboursable** d’un projet communautaire.
* Automatisation du **remboursement à l’échéance**.

**### 4.5 Modèle Reward**

**📘 Description**

Le modèle Reward représente une **contrepartie** proposée par un projet aux donateurs.  
Exemple : “Donnez 50 €, recevez un T-shirt.”

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant. |
| projectId | String | Projet concerné. |
| title | String | Nom de la récompense. |
| description | String? | Détails de la contrepartie. |
| minAmount | Float | Montant minimum pour y accéder. |
| stock | Int? | Quantité disponible (optionnelle). |
| createdAt | DateTime | Création. |

**💡 Cas d’usage**

* Système de **crowdfunding classique avec paliers** :
  + 10 € → carte de remerciement
  + 50 € → T-shirt
  + 100 € → invitation à l’événement du projet
* Gestion automatique du **stock** et des **contreparties expédiées**.

**### 4.6 Modèle ProjectMember**

**📘 Description**

Définit les **membres associés** à un projet, autres que le propriétaire (collaborateurs, cofondateurs, etc.).

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant. |
| projectId | String | Projet concerné. |
| userId | String | Utilisateur membre. |
| role | ProjectRole | Rôle dans le projet (OWNER, COLLABORATOR, VIEWER). |
| joinedAt | DateTime | Date d’entrée. |

**💡 Cas d’usage**

* Ajouter un **co-porteur de projet** avec rôle COLLABORATOR.
* Déléguer des permissions de gestion selon le rôle.
* Historiser la **composition de l’équipe projet**.

**### 4.7 Enum ProjectStatus**

| **Valeur** | **Description** |
| --- | --- |
| DRAFT | En préparation, non visible. |
| ACTIVE | En cours de financement. |
| COMPLETED | Objectif atteint ou terminé. |
| FAILED | Échec de financement. |
| ARCHIVED | Projet clos. |

**### 4.8 Enum ProjectRole**

| **Valeur** | **Description** |
| --- | --- |
| OWNER | Porteur principal du projet. |
| COLLABORATOR | Membre contributeur. |
| VIEWER | Accès en lecture seule. |

**💡 Exemples d’usage combiné**

**Cas 1 – Campagne de dons :**

sequenceDiagram

User ->> Project: Crée un projet DRAFT

User ->> Project: Publie le projet (status=ACTIVE)

Supporter ->> Donation: Fait un don de 50€

System ->> Project: Met à jour collectedAmount

Project ->> Reward: Attribue la contrepartie correspondante

**Cas 2 – Investissement collectif :**

sequenceDiagram

Investor ->> Project: Investit 500€ (Investment)

System ->> Transaction: Crée une transaction créditée au projet

Project ->> Circle: Verse des parts du projet au cercle associé

Circle ->> Investor: Distribue le rendement à maturité

**🫱‍🫲 Partie 5 — Cercles et Communautés**

**### 5.1 Modèle Circle**

**📘 Description**

Le modèle Circle représente une **communauté, organisation ou collectif** au sein de la plateforme.  
C’est une entité **intermédiaire entre l’individu et les projets** : un espace collaboratif où plusieurs utilisateurs s’organisent autour d’une mission commune (par exemple une ONG, un collectif écologique ou un groupe local).

Un Circle peut :

* posséder des **membres (CircleMember)**,
* gérer des **projets (CircleProject)**,
* disposer d’un **fonds commun (CircleFund)**,
* recevoir des **dons, prêts ou investissements collectifs**,
* participer à des **sprints d’impact**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| name | String | Nom du cercle (ex : *Collectif Climat Paris*). |
| slug | String | Identifiant d’URL unique. |
| description | String? | Description du collectif. |
| ownerId | String | Identifiant du créateur du cercle (User). |
| visibility | CircleVisibility | Niveau de visibilité (PUBLIC, PRIVATE, INVITE\_ONLY). |
| createdAt / updatedAt | DateTime | Suivi temporel. |
| coverImage | String? | Image illustrative du cercle. |
| walletId | String? | Portefeuille financier associé (lien vers Wallet). |

**Relations**

* owner → User (créateur du cercle)
* members → Liste de CircleMember
* projects → Liste de CircleProject
* funds → CircleFund (fonds collectifs)
* transactions → Historique des flux financiers du cercle

**💡 Cas d’usage**

* Création d’un **cercle communautaire** pour regrouper plusieurs projets.
* Attribution d’un **fonds commun** pour soutenir des projets internes.
* Gestion d’un **portefeuille collectif** transparent.
* Filtrage de projets par cercle sur la plateforme.

**### 5.2 Modèle CircleMember**

**📘 Description**

Le modèle CircleMember définit les **utilisateurs membres d’un cercle**, avec leurs **rôles et permissions**.  
Il matérialise la **gouvernance communautaire**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| circleId | String | Référence du cercle. |
| userId | String | Référence de l’utilisateur membre. |
| role | CircleRole | Rôle dans le cercle (ADMIN, MEMBER, GUEST). |
| joinedAt | DateTime | Date d’entrée dans le cercle. |
| status | CircleMemberStatus | État du membre (ACTIVE, INVITED, PENDING, REMOVED). |

**Relations**

* circle → Circle
* user → User

**💡 Cas d’usage**

* Inviter un utilisateur à rejoindre un cercle.
* Gérer les **permissions selon le rôle** :
  + ADMIN → gestion du cercle et des fonds.
  + MEMBER → création de projets.
  + GUEST → consultation uniquement.
* Historiser l’évolution des membres.

**### 5.3 Modèle CircleProject**

**📘 Description**

CircleProject est une **table de liaison** entre un Circle et un Project.  
Elle permet de rattacher un projet à un ou plusieurs cercles, facilitant la **gouvernance partagée** et le **cofinancement**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant. |
| circleId | String | Référence vers le cercle. |
| projectId | String | Référence vers le projet. |
| role | CircleProjectRole | Rôle du cercle vis-à-vis du projet (FUNDER, PARTNER, OWNER). |
| addedAt | DateTime | Date d’ajout du projet au cercle. |

**💡 Cas d’usage**

* Un **cercle financeur** soutient plusieurs projets (role = FUNDER).
* Un **cercle thématique** héberge ses propres projets (role = OWNER).
* Une **ONG partenaire** est rattachée comme PARTNER à un projet.

**### 5.4 Modèle CircleFund**

**📘 Description**

CircleFund représente un **fonds collectif** géré par un cercle.  
C’est une réserve d’argent que les membres peuvent utiliser pour financer des projets, offrir des récompenses, ou organiser des campagnes internes.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| circleId | String | Référence du cercle propriétaire. |
| name | String | Nom du fonds (ex : *Fonds Climat 2025*). |
| balance | Float | Montant disponible. |
| currency | String | Devise. |
| createdAt | DateTime | Date de création. |
| walletId | String? | Lien vers le portefeuille financier. |

**Relations**

* circle → Circle
* wallet → Wallet
* transactions → Historique des mouvements liés au fonds

**💡 Cas d’usage**

* Mise en place d’un **fonds commun** alimenté par des dons des membres.
* **Subvention automatique** d’un projet quand certains critères sont atteints.
* Gestion **transparente du solde** et des mouvements du fonds.

**### 5.5 Enums associées**

**CircleVisibility**

| **Valeur** | **Description** |
| --- | --- |
| PUBLIC | Accessible à tous. |
| PRIVATE | Visible uniquement par les membres. |
| INVITE\_ONLY | Rejoignable uniquement sur invitation. |

**CircleRole**

| **Valeur** | **Description** |
| --- | --- |
| ADMIN | Gère le cercle et ses fonds. |
| MEMBER | Participe aux activités et projets. |
| GUEST | Simple observateur. |

**CircleMemberStatus**

| **Valeur** | **Description** |
| --- | --- |
| ACTIVE | Membre en activité. |
| INVITED | Invitation envoyée. |
| PENDING | En attente de validation. |
| REMOVED | Membre retiré. |

**CircleProjectRole**

| **Valeur** | **Description** |
| --- | --- |
| OWNER | Le cercle est le créateur du projet. |
| PARTNER | Le cercle collabore sur le projet. |
| FUNDER | Le cercle finance le projet. |

**💡 Exemple d’usage combiné**

sequenceDiagram

User ->> Circle: Crée un cercle "Collectif Climat"

Circle ->> CircleFund: Crée un fonds "Fonds Vert 2025"

Admin ->> Project: Crée un projet "Plantons 1000 arbres"

Circle ->> CircleProject: Associe le projet avec rôle FUNDER

CircleFund ->> Transaction: Effectue un financement de 2000€

Project ->> Donation: Reçoit la contribution du cercle

**💰 Partie 6 — Transactions, Portefeuilles et Allocations**

**### 6.1 Modèle Wallet**

**📘 Description**

Le modèle Wallet représente un **portefeuille financier**.  
Chaque wallet appartient à un **utilisateur, cercle ou projet** et permet de **gérer les fonds disponibles, les fonds bloqués et les flux financiers**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| userId | String? | Propriétaire utilisateur (optionnel). |
| circleId | String? | Propriétaire cercle (optionnel). |
| projectId | String? | Propriétaire projet (optionnel). |
| ownerKind | WalletKind | Type de propriétaire (USER, CIRCLE, PROJECT). |
| currency | String | Devise du portefeuille (ex : "EUR"). |
| available | Float | Montant disponible pour transactions. |
| locked | Float | Montant bloqué (ex : promesses, allocations). |
| createdAt / updatedAt | DateTime | Suivi temporel. |

**Relations**

* user → User
* circle → Circle
* project → Project
* ledger → Liste de WalletLedger pour historiser toutes les opérations

**💡 Cas d’usage**

* Gérer le **solde d’un utilisateur** pour effectuer des dons ou investissements.
* Stocker le **capital d’un projet ou d’un cercle**.
* Séparer les **fonds disponibles** et les **fonds bloqués** pour des promesses ou allocations.

**### 6.2 Modèle Transaction**

**📘 Description**

Le modèle Transaction enregistre **toute opération financière** sur la plateforme :  
dons, paiements, abonnements, transferts entre wallets.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| userId | String | Utilisateur initiateur. |
| type | String | Type de transaction (DEPOSIT, WITHDRAW, PLEDGE, etc.). |
| amount | Float | Montant de la transaction. |
| currency | String | Devise utilisée. |
| status | String | Statut (pending, completed, failed). |
| projectId / circleId | String? | Projet ou cercle concerné (optionnel). |
| metadata | Json? | Informations complémentaires. |
| createdAt / updatedAt | DateTime | Suivi temporel. |

**Relations**

* user → User (initiateur)
* project → Project (si transaction liée à un projet)
* circle → Circle (si transaction liée à un cercle)
* allocations → Liste de Allocation si la transaction finance plusieurs projets

**💡 Cas d’usage**

* Enregistrement d’un **dépôt utilisateur** pour financer un projet.
* Paiement d’un **abonnement à un cercle**.
* Historisation d’un **transfert interne** entre wallets de cercle et projet.

**### 6.3 Modèle WalletLedger**

**📘 Description**

WalletLedger représente **chaque mouvement financier** dans un wallet, assurant **transparence et traçabilité**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| walletId | String | Wallet concerné. |
| type | WalletLedgerType | Nature du mouvement (DEPOSIT, WITHDRAW, PROMISE\_LOCK, PLEDGE\_SETTLE, etc.). |
| amount | Float | Montant de l’opération. |
| balanceAfter | Float | Solde après opération. |
| referenceType / referenceId | String? | Référence vers transaction ou événement lié. |
| metadata | Json? | Infos complémentaires. |
| createdAt | DateTime | Date de l’opération. |

**Relations**

* wallet → Wallet

**💡 Cas d’usage**

* **Tracer chaque mouvement** pour audits internes ou reporting.
* Vérifier **disponibilité vs fonds bloqués**.
* Historiser les **promesses de dons ou d’investissements**.

**### 6.4 Modèle Allocation**

**📘 Description**

Allocation permet de **répartir une transaction entre plusieurs projets ou activités** au sein d’un cercle ou wallet collectif.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| circleId | String | Cercle responsable. |
| transactionId | String | Transaction source. |
| operatingAmount | Float | Montant pour frais internes ou gestion. |
| projectAmount | Float | Montant alloué aux projets. |
| period | String? | Période comptable (ex : "2025-Q4"). |
| createdAt | DateTime | Date de création. |

**Relations**

* circle → Circle
* transaction → Transaction

**💡 Cas d’usage**

* Un cercle reçoit 1000 € et **alloue 800 € à 3 projets** et 200 € à frais internes.
* Suivi du **budget collectif par période**.
* Reporting de **l’impact des fonds** par projet.

**### 6.5 Modèle Subscription**

**📘 Description**

Subscription représente un **paiement récurrent d’un utilisateur à un cercle**.  
C’est le mécanisme derrière les **adhésions ou abonnements mensuels**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| userId | String | Utilisateur abonné. |
| circleId | String | Cercle abonné. |
| amount | Float | Montant récurrent. |
| currency | String | Devise. |
| status | String | Statut (active, cancelled). |
| startedAt | DateTime | Début de l’abonnement. |
| nextBillingAt | DateTime? | Prochaine échéance. |
| cancelledAt | DateTime? | Date de résiliation. |

**Relations**

* user → User
* circle → Circle

**💡 Cas d’usage**

* Facturation **mensuelle automatique** des membres.
* Suspension ou annulation d’un abonnement.
* Suivi des **revenus récurrents par cercle**.

**💡 Exemple d’usage combiné**

sequenceDiagram

User ->> Wallet: Dépose 100€

Wallet ->> Transaction: Crée transaction DEPOSIT

Transaction ->> WalletLedger: Historise l'opération

Wallet ->> Allocation: Alloue 60€ à 2 projets, 40€ frais

User ->> Subscription: Paie abonnement cercle 10€/mois

Subscription ->> Transaction: Crée transaction récurrente

Transaction ->> WalletLedger: Historise

Cette structure **garantit traçabilité, transparence et flexibilité** pour toutes les transactions, qu’elles soient individuelles, collectives ou projets spécifiques.

**🎮 Partie 7 — Gamification et XP**

**### 7.1 Modèle UserXP**

**📘 Description**

UserXP suit **l’expérience accumulée par un utilisateur**.  
Elle reflète son **niveau global**, les **points d’expérience par catégorie** (contenu, communauté, donations, investissements) et son **rang d’impact**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| userId | String | Utilisateur concerné. |
| totalXP | Int | XP total accumulé. |
| level | Int | Niveau utilisateur. |
| impactRank | String? | Classement ou rang d’influence (ex : "Bronze", "Silver"). |
| donationXP | Int | XP provenant des dons. |
| investmentXP | Int | XP provenant des investissements. |
| contentXP | Int | XP provenant de la création de contenu. |
| communityXP | Int | XP provenant de la participation communautaire. |
| createdAt / updatedAt | DateTime | Suivi temporel. |

**Relations**

* user → User

**💡 Cas d’usage**

* Calculer le **niveau d’un utilisateur** pour déverrouiller des fonctionnalités.
* Suivre l’**impact global** sur la plateforme.
* Créer un **tableau de classement** basé sur l’XP.

**### 7.2 Modèle XPEvent**

**📘 Description**

XPEvent représente **chaque action qui modifie l’XP** d’un utilisateur.  
Elle permet de tracer **la source, le type et le delta d’XP**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| userId | String | Utilisateur concerné. |
| type | XPEventType | Type d’événement (READ, SHARE, COMMENT, REVIEW, PLEDGE, WIN, BADGE, SPEND). |
| delta | Int | Variation d’XP (positive ou négative). |
| sourceType | String? | Source de l’événement (ex : "Post", "Project"). |
| sourceId | String? | ID de l’entité source. |
| metadata | Json? | Informations supplémentaires. |
| createdAt | DateTime | Date de l’événement. |

**Relations**

* user → User

**💡 Cas d’usage**

* Un utilisateur **lit un article** → +5 XP.
* Un utilisateur **fait un don** → +20 XP dans donationXP.
* Un utilisateur **reçoit un badge** → +50 XP.
* Historisation complète pour audits ou classements.

**### 7.3 Modèle Badge**

**📘 Description**

Badge représente un **récompense visible** pour un utilisateur, souvent liée à un critère ou accomplissement spécifique.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| name | String | Nom du badge (ex : "Top Donor"). |
| description | String? | Description du badge. |
| image | String? | URL de l’image du badge. |
| criteria | Json? | Conditions pour obtenir le badge (ex : { "donationXP": 100 }). |
| createdAt | DateTime | Date de création. |

**Relations**

* userBadges → Liste de UserBadge

**💡 Cas d’usage**

* Récompenser un **utilisateur qui fait un don supérieur à 100€**.
* Débloquer un **badge pour contribution au contenu**.
* Visuel pour encourager **l’engagement social et la compétition amicale**.

**### 7.4 Modèle UserBadge**

**📘 Description**

UserBadge relie un utilisateur à un badge obtenu, avec une **date d’attribution**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| userId | String | Utilisateur détenteur. |
| badgeId | String | Badge obtenu. |
| earnedAt | DateTime | Date de l’obtention du badge. |

**Relations**

* user → User
* badge → Badge

**💡 Cas d’usage**

* Afficher tous les **badges obtenus par un utilisateur**.
* Vérifier les **conditions d’attribution des badges**.
* Historiser **l’évolution de l’engagement**.

**### 7.5 Enum XPEventType**

| **Valeur** | **Description** |
| --- | --- |
| READ | Lecture de contenu. |
| SHARE | Partage de contenu. |
| COMMENT | Publication d’un commentaire. |
| REVIEW | Réalisation d’une revue (Peer Review). |
| PLEDGE | Engagement financier dans un projet/sprint. |
| WIN | Gain d’un sprint ou compétition. |
| BADGE | Attribution d’un badge. |
| SPEND | Dépense d’XP (ex : échange contre privilèges). |

**💡 Exemple d’usage combiné**

sequenceDiagram

User ->> Post: Lit un article

Post ->> XPEvent: Enregistre +5 XP READ

User ->> ProjectDonation: Fait un don de 50€

ProjectDonation ->> XPEvent: +20 XP donationXP

XPEvent ->> UserXP: Met à jour totalXP et level

System ->> Badge: Vérifie critères (donationXP >=50)

Badge ->> UserBadge: Attribue badge "Top Donor"

UserXP ->> User: Affiche le niveau et badges

Cette architecture **favorise l’engagement utilisateur**, trace toutes les interactions et permet **d’intégrer facilement des récompenses, classements et statistiques d’impact**.

**🛡️ Partie 8 — Conformité, KYC et Entités légales**

**### 8.1 Enum KycStatus**

| **Valeur** | **Description** |
| --- | --- |
| NONE | L’utilisateur n’a pas commencé le processus KYC. |
| PENDING | La vérification est en cours. |
| VERIFIED | L’utilisateur est vérifié. |
| REJECTED | La vérification a été refusée. |

**💡 Cas d’usage**

* Contrôler l’accès aux fonctionnalités financières sensibles (dons importants, investissements).
* Afficher un **statut KYC** sur le profil utilisateur.
* Déclencher des alertes internes pour validation manuelle.

**### 8.2 Modèle User**

* kycStatus: KycStatus — reflète l’état de vérification.
* legalEntities: LegalEntity[] — permet à un utilisateur de posséder ou représenter des entités légales.

💡 **Cas d’usage** :

* Un investisseur professionnel doit avoir kycStatus = VERIFIED pour participer à un projet equity.
* Un créateur de projet peut lier sa LegalEntity pour les aspects fiscaux et contractuels.

**### 8.3 Modèle LegalEntity**

**📘 Description**

LegalEntity représente une **entité légale** (entreprise, association, fondation) possédée par un utilisateur.  
Elle permet de gérer **les obligations légales et fiscales** des projets.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| type | String | Type d’entité (ex : "SAS", "Association"). |
| name | String? | Nom officiel. |
| country | String? | Pays d’enregistrement. |
| registrationNumber | String? | Numéro d’immatriculation. |
| taxId | String? | Numéro fiscal / TVA. |
| contactEmail / contactPhone | String? | Contacts officiels. |
| ownerId | String? | Référence à l’utilisateur propriétaire. |
| projects | Project[] | Projets liés à l’entité. |
| createdAt / updatedAt | DateTime | Suivi temporel. |

**Relations**

* owner → User
* projects → Project[]

**💡 Cas d’usage**

* Une association ou entreprise peut **porter un projet** légalement.
* Permet d’**assurer la conformité fiscale et légale** pour les investissements ou dons.
* Facilite le **reporting légal** et la traçabilité pour audits.

**### 8.4 Modèle FeatureFlag**

**📘 Description**

FeatureFlag permet de **déployer progressivement des fonctionnalités**, utile pour **activer/désactiver des options KYC ou conformité**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| key | String | Nom du flag (ex : "kyc\_required"). |
| enabled | Boolean | Active ou désactive la fonctionnalité. |
| rollout | Json? | Informations de déploiement (pourcentage utilisateurs, groupes). |
| createdAt / updatedAt | DateTime | Suivi temporel. |

**💡 Cas d’usage**

* Activer la **vérification KYC obligatoire** pour les dons > 1000 €.
* Activer certaines **fonctionnalités légales pour certains pays**.
* Faire des **tests progressifs** sans perturber tous les utilisateurs.

**### 8.5 Processus KYC et conformité (exemple)**

sequenceDiagram

User ->> System: Commence le KYC

System ->> Verification: Crée dossier KYC

Verification ->> User: Statut PENDING

Admin ->> Verification: Vérifie documents

Verification ->> User: Statut VERIFIED ou REJECTED

User ->> ProjectDonation: Accès autorisé seulement si VERIFIED

LegalEntity ->> Project: Lie entité au projet pour conformité légale

**💡 Bonnes pratiques**

1. **Toujours vérifier kycStatus avant les transactions sensibles**.
2. **Associer les projets à une entité légale** pour la fiscalité et les contrats.
3. **Historiser les actions KYC et vérifications** pour audit et reporting.
4. **Utiliser les FeatureFlags pour activer progressivement les règles de conformité** selon le type de projet, la région ou le profil utilisateur.

Cette partie garantit **la sécurité juridique et financière** de la plateforme, et protège à la fois **les utilisateurs** et **l’organisation** contre les risques légaux.

**⚡ Partie 9 — Sprints et Pledges**

**### 9.1 Enum SprintStatus**

| **Valeur** | **Description** |
| --- | --- |
| DRAFT | Sprint créé mais non actif. |
| UPCOMING | Sprint planifié et à venir. |
| RUNNING | Sprint en cours. |
| SETTLED | Sprint terminé et résultats enregistrés. |
| CANCELLED | Sprint annulé. |

💡 **Cas d’usage** :

* Filtrer les sprints actifs (RUNNING) pour l’affichage public.
* Empêcher de modifier un sprint terminé (SETTLED).

**### 9.2 Modèle Sprint**

**📘 Description**

Sprint représente une **campagne d’impact ou challenge collaboratif**.  
Il regroupe **plusieurs projets participants** et les **engagements financiers (pledges)** des utilisateurs.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| name | String | Nom du sprint. |
| slug | String | Identifiant lisible pour URL. |
| theme | String? | Thème ou objectif du sprint. |
| startAt / endAt | DateTime? | Dates de début et fin. |
| status | SprintStatus | Statut actuel du sprint. |
| winnerProjectId | String? | ID du projet gagnant. |
| sprintProjects | SprintProject[] | Projets participant au sprint. |
| pledges | Pledge[] | Engagements financiers des utilisateurs. |
| createdAt / updatedAt | DateTime | Suivi temporel. |

**Relations**

* winnerProject → Project?
* sprintProjects → SprintProject[]
* pledges → Pledge[]

**💡 Cas d’usage**

* Organiser un **challenge écologique** avec 5 projets.
* Suivre la **participation des utilisateurs** via les pledges.
* Déterminer le **projet gagnant** selon les votes ou engagements.

**### 9.3 Modèle SprintProject**

**📘 Description**

SprintProject relie **un projet à un sprint**, avec un ordre ou classement éventuel.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| sprintId | String | Référence au sprint. |
| projectId | String | Référence au projet. |
| position | Int? | Classement ou ordre dans le sprint. |

**Relations**

* sprint → Sprint
* project → Project

**💡 Cas d’usage**

* Déterminer **l’ordre de présentation** des projets dans un sprint.
* Mettre à jour **le projet gagnant** après le vote ou le financement.

**### 9.4 Enum PledgeStatus**

| **Valeur** | **Description** |
| --- | --- |
| PROMISED | Engagement enregistré mais non encore débité. |
| SETTLED | Engagement débité et confirmé. |
| CANCELLED | Engagement annulé par l’utilisateur. |
| EXPIRED | Engagement non honoré avant la fin du sprint. |

💡 **Cas d’usage** :

* Filtrer les pledges actifs pour calculer les fonds disponibles.
* Gérer le remboursement automatique si CANCELLED ou EXPIRED.

**### 9.5 Modèle Pledge**

**📘 Description**

Pledge représente **l’engagement financier d’un utilisateur** dans un sprint.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| sprintId | String | Sprint concerné. |
| userId | String | Utilisateur ayant fait le pledge. |
| amount | Float | Montant engagé. |
| status | PledgeStatus | Statut de l’engagement. |
| settledProjectId | String? | Projet bénéficiaire après settlement. |
| pledgeSelections | PledgeSelection[] | Sélection des projets favoris par l’utilisateur. |
| createdAt | DateTime | Date de création. |

**Relations**

* sprint → Sprint
* user → User
* pledgeSelections → PledgeSelection[]

**💡 Cas d’usage**

* Un utilisateur **s’engage à financer 100€** dans un sprint.
* Après la fin du sprint, le pledge est **settled** pour le projet choisi.
* Permet de **calculer les contributions totales par projet et sprint**.

**### 9.6 Modèle PledgeSelection**

**📘 Description**

PledgeSelection relie **un pledge à un ou plusieurs projets**.  
Cela permet à un utilisateur de **diviser son engagement sur plusieurs projets**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| pledgeId | String | Référence au pledge. |
| projectId | String | Projet sélectionné. |

**Relations**

* pledge → Pledge
* project → Project

**💡 Cas d’usage**

* Un utilisateur fait un **pledge de 200€**, réparti sur **2 projets du sprint**.
* Chaque projet reçoit **la part correspondante** du pledge lors du settlement.
* Facilite **le suivi des allocations et rapports de contribution**.

**### 9.7 Exemple d’usage combiné**

sequenceDiagram

User ->> Sprint: Consulte sprint "Eco-Challenge"

User ->> Pledge: Fait un pledge de 150€

Pledge ->> PledgeSelection: Sélectionne 2 projets

Sprint ->> SprintProject: Enregistre projets participants

Sprint ->> Pledge: Met à jour statut SETTLED après fin du sprint

Pledge ->> Project: Transfert montant aux projets sélectionnés

System ->> Wallet: Mets à jour wallets utilisateurs et projets

Cette architecture **permet de gérer des campagnes collaboratives complexes**, de suivre les **engagements financiers et leur allocation**, et d’assurer **la traçabilité complète des pledges**.

**💰 Partie 10 — Gestion des Portefeuilles et Transactions**

**### 10.1 Enum WalletKind**

| **Valeur** | **Description** |
| --- | --- |
| USER | Portefeuille appartenant à un utilisateur. |
| CIRCLE | Portefeuille appartenant à un cercle. |
| PROJECT | Portefeuille appartenant à un projet. |

💡 **Cas d’usage** :

* Identifier le type de propriétaire du portefeuille pour appliquer des règles spécifiques.
* Filtrer les transactions par type (ex : toutes les transactions PROJECT pour un audit financier).

**### 10.2 Enum WalletLedgerType**

| **Valeur** | **Description** |
| --- | --- |
| DEPOSIT | Dépôt sur le wallet. |
| WITHDRAW | Retrait du wallet. |
| PROMISE\_LOCK | Verrouillage de fonds pour un pledge. |
| PROMISE\_RELEASE | Libération d’un pledge non utilisé. |
| PLEDGE\_SETTLE | Paiement final d’un pledge. |
| REFUND | Remboursement à l’utilisateur. |
| SUBSCRIPTION\_FEE | Paiement de frais d’abonnement. |
| ALLOCATION | Allocation de fonds à un projet via un cercle. |
| TRANSFER\_IN / TRANSFER\_OUT | Transfert entre wallets. |

💡 **Cas d’usage** :

* Suivi précis du **mouvement des fonds**.
* Historisation pour les audits et la conformité légale.

**### 10.3 Modèle Wallet**

**📘 Description**

Wallet représente le **portefeuille d’un utilisateur, cercle ou projet**.  
Il gère **les soldes disponibles et bloqués**, ainsi que **l’historique des opérations** via le WalletLedger.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| userId / circleId / projectId | String? | Référence au propriétaire. |
| ownerKind | WalletKind | Type de propriétaire. |
| currency | String | Devise (ex : EUR). |
| chain | String? | Chaîne blockchain si applicable. |
| asset | String? | Type d’actif (ex : token, crypto). |
| address | String? | Adresse du wallet externe. |
| available | Float | Solde disponible pour transactions. |
| locked | Float | Fonds verrouillés (ex : pledges non réglés). |
| ledger | WalletLedger[] | Historique détaillé des opérations. |
| createdAt / updatedAt | DateTime | Suivi temporel. |

**💡 Cas d’usage**

* Un utilisateur fait un **dépôt de 100€** : available augmente et un WalletLedger de type DEPOSIT est créé.
* Un pledge est créé : locked augmente et un WalletLedger PROMISE\_LOCK est ajouté.
* Paiement final d’un pledge : locked diminue, available diminue si nécessaire, PLEDGE\_SETTLE est enregistré.

**### 10.4 Modèle WalletLedger**

**📘 Description**

WalletLedger garde **l’historique de toutes les opérations sur un wallet**, garantissant **traçabilité et auditabilité**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| walletId | String | Référence au wallet concerné. |
| type | WalletLedgerType | Type d’opération. |
| amount | Float | Montant de l’opération. |
| balanceAfter | Float | Solde après l’opération. |
| referenceType | String? | Source de la transaction (ex : Pledge, Subscription). |
| referenceId | String? | ID de l’objet source. |
| metadata | Json? | Informations additionnelles. |
| createdAt | DateTime | Date de l’opération. |

**💡 Cas d’usage**

* Traçabilité complète pour les audits financiers.
* Historisation des **dépôts, retraits, transferts et pledges**.
* Permet de reconstruire le **solde exact à tout instant**.

**### 10.5 Modèle Transaction**

**📘 Description**

Transaction représente **une opération financière globale**, souvent déclenchée par un utilisateur ou un cercle, et pouvant être allouée à **projets ou cercles**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| userId | String | Utilisateur initiateur. |
| type | String | Type de transaction (ex : donation, pledge, subscription). |
| amount | Float | Montant de la transaction. |
| currency | String | Devise. |
| status | String | État (pending, completed, etc.). |
| description | String? | Détail de la transaction. |
| projectId / circleId | String? | Références pour allocation des fonds. |
| metadata | Json? | Informations supplémentaires. |
| allocations | Allocation[] | Détail de répartition des fonds. |
| createdAt / updatedAt | DateTime | Suivi temporel. |

**💡 Cas d’usage**

* Une **donation à un projet** crée une transaction et met à jour le wallet du projet.
* Une **allocation de fonds d’un cercle** vers plusieurs projets crée plusieurs allocations attachées à la transaction.
* Le statut permet de gérer **les paiements en attente** ou **échoués**.

**### 10.6 Modèle Allocation**

**📘 Description**

Allocation représente **la répartition d’une transaction de cercle vers des projets**.  
Cela permet un **suivi précis des fonds collectés et distribués**.

**🧩 Champs**

| **Champ** | **Type** | **Description** |
| --- | --- | --- |
| id | String | Identifiant unique. |
| circleId | String | Cercle source. |
| transactionId | String | Transaction associée. |
| operatingAmount | Float | Montant conservé pour les frais internes. |
| projectAmount | Float | Montant transféré aux projets. |
| period | String? | Période d’allocation (ex : mois, trimestre). |
| createdAt | DateTime | Date de l’allocation. |

**Relations**

* circle → Circle
* transaction → Transaction

**💡 Cas d’usage**

* Un cercle collecte **500€ de ses membres** : 450€ vont aux projets, 50€ restent pour frais internes.
* Chaque allocation est tracée individuellement pour **audit et reporting**.

**### 10.7 Exemple d’usage combiné**

sequenceDiagram

User ->> Wallet: Dépose 200€

Wallet ->> WalletLedger: DEPOSIT 200€, balanceAfter 200€

User ->> Pledge: Engage 100€ sur sprint

Wallet ->> WalletLedger: PROMISE\_LOCK 100€, locked 100€

Sprint ->> PledgeSelection: Sélectionne projets A et B

Sprint ->> Wallet: PLEDGE\_SETTLE, débite wallet, crédite projets

Circle ->> Transaction: Collecte 500€

Transaction ->> Allocation: 450€ projets, 50€ frais internes

WalletLedger ->> Wallet: Enregistre chaque mouvement

Cette partie assure que **toutes les opérations financières sont tracées**, les **soldes sont cohérents** et la plateforme peut **garantir la sécurité et la transparence** des fonds.