1. EVENT MODEL

- _id
- title
- description
- location
- schedule [start, end]
- registrationDeadline
- createdBy
- maxParticipants
- registeredUsers [userId, registeredAt, checkInStatus, role]
- waitlist [userId, joinedAt]
- status
- isPublished
- canceledReason
- notifications [sendReminders, reminderTimes]
- paidEvent
- createdAt

```
const mongoose = require("mongoose");

const EventSchema = new mongoose.Schema({
_id: {type: int, required: true}

title: { type: String, required: true, trim: true },

description: { type: String, required: true },

location: { type: String, required: true },

schedule: {
   start: { type: Date, required: true },
   end: { type: Date, required: true }
},

registrationDeadline: { type: Date, required: true }, // New field

createdBy: { type: mongoose.Schema.Types.ObjectId, ref: "Admin", required: true },

maxParticipants: { type: Number, default: 100 },

registeredUsers: [
   {
     userId: { type: mongoose.Schema.Types.ObjectId, ref: "User" },
     registeredAt: { type: Date, default: Date.now },
```

```
    checkInStatus: { type: Boolean, default: false }, // New field
    role: { type: String, enum: ["participant", "speaker", "VIP"], default: "participant" } //
New field
  }
 ],

waitlist: [
  { userId: { type: mongoose.Schema.Types.ObjectId, ref: "User" }, joinedAt: { type: Date,
default: Date.now } }
],

status: {
  type: String,
  enum: ["draft", "upcoming", "ongoing", "completed", "canceled"],
  default: "draft"
},

isPublished: { type: Boolean, default: false }, // New field
 canceledReason: { type: String, default: null }, // New field

notifications: {
  sendReminders: { type: Boolean, default: true },
  reminderTimes: [{ type: Date }] // Allows multiple reminders
 },

paidEvent: {
  sendReminders: { type: Boolean, default: true },
 },

createdAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model("Event", EventSchema);
```

2.USERS MODEL

- fullName
- Email
- Password
- telNum
- profilePicture
- address [street, city, country]
- role [user, subAdmin, admin]

- status [active, suspended, block]
- authMethod [isEmailVerified, isPhoneVerified, is2faVerified]
- registeredEvents [id, registeredAt]
- waitListEven[id, joinedAt]

User schema : const mongoose = require("mongoose");

```
const UserSchema = new mongoose.Schema({
  fullName: { type: String, required: true, trim: true },
  email: { type: String, required: true, unique: true, lowercase: true },
  password: { type: String, required: true, minlength: 6 },

  phoneNumber: { type: String, unique: true, sparse: true },
  avatar: { type: String, default: "default-avatar.png" }, // URL to profile picture
  bio: { type: String, maxlength: 500 },
  address: {
    street: { type: String },
    city: { type: String },
    country: { type: String }
  },

  role: {
    type: String,
    enum: ["user", "sub-admin", "admin"],
    default: "user"
  },

  status: {
    type: String,
    enum: ["active", "suspended", "blocked"],
    default: "active"
  },

  authentication: {
    isEmailVerified: { type: Boolean, default: false },
    isPhoneVerified: { type: Boolean, default: false },
    twoFactorEnabled: { type: Boolean, default: false }
  },

  registeredEvents: [
    {
      eventId: { type: mongoose.Schema.Types.ObjectId, ref: "Event" },
      registeredAt: { type: Date, default: Date.now },
      checkedIn: { type: Boolean, default: false }
```

```
  }
 ],

 waitlistedEvents: [
  {
   eventId: { type: mongoose.Schema.Types.ObjectId, ref: "Event" },
   joinedAt: { type: Date, default: Date.now }
  }
 ],

 attendedEvents: [
  {
   eventId: { type: mongoose.Schema.Types.ObjectId, ref: "Event" },
   attendedAt: { type: Date }
  }
 ],

 notifications: {
  emailNotifications: { type: Boolean, default: true },
  smsNotifications: { type: Boolean, default: false },
  appNotifications: { type: Boolean, default: true }
 },

 loginHistory: [
  {
   ipAddress: { type: String },
   device: { type: String },
   timestamp: { type: Date, default: Date.now }
  }
 ],

 lastLogin: { type: Date },
 createdAt: { type: Date, default: Date.now },
 updatedAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model("User", UserSchema);
```

3. SETTINGS MODEL

- allowWaitListing
- eventApproval
- limit
- Notifications [reminderSchedule, eventEmailNotification]

- Security [enable2FA, passwordLength]
- paymentSettings [enablePaidEvents, currencyOption, paymentGateway{paypal, stripe, none}]
- apiKey
- updatedAt
- updatedBy[user_id]

@masṭęryoda schema for system setting: const mongoose = require("mongoose");

```
const SystemSettingsSchema = new mongoose.Schema({
  general: {
    systemName: { type: String, default: "Event Management System" },
    logo: { type: String, default: "default-logo.png" }, // URL to logo
    contactEmail: { type: String, default: "support@example.com" },
    contactPhone: { type: String, default: "+1234567890" }
  },

  userManagement: {
    allowUserRegistration: { type: Boolean, default: true }, // Can users sign up?
    maxFailedLogins: { type: Number, default: 5 }, // Lock user after X failed logins
    adminRoles: {
      allowSubAdmins: { type: Boolean, default: true }, // Can admins invite sub-admins?
      maxSubAdmins: { type: Number, default: 10 } // Limit number of sub-admins
    }
  },

  eventManagement: {
    maxRegistrationsPerEvent: { type: Number, default: 100 }, // Limit number of
attendees
    allowWaitlist: { type: Boolean, default: true }, // Enable waitlisting
    eventApprovalRequired: { type: Boolean, default: false } // Does an admin need to
approve events?
  },

  notifications: {
    enableEmailNotifications: { type: Boolean, default: true },
    enableSMSNotifications: { type: Boolean, default: false },
    enableAppNotifications: { type: Boolean, default: true },
    eventReminderSchedule: { type: Number, default: 24 } // Hours before event to send
reminders
  },

  security: {
    enableTwoFactorAuthentication: { type: Boolean, default: false },
```

```
    passwordPolicy: {
      minLength: { type: Number, default: 8 },
      requireSpecialCharacter: { type: Boolean, default: true },
      requireUppercase: { type: Boolean, default: true },
      requireNumber: { type: Boolean, default: true }
    }
  },

  paymentSettings: {
    enablePaidEvents: { type: Boolean, default: false },
    supportedCurrencies: { type: [String], default: ["USD"] },
    paymentGateway: {
      provider: { type: String, enum: ["Stripe", "PayPal", "None"], default: "None" },
      apiKey: { type: String, default: "" } // API key for payment processing
    }
  },

  updatedAt: { type: Date, default: Date.now }
});

module.exports = mongoose.model("SystemSettings", SystemSettingsSchema);
```

4. AUDIT MODELS

- action [createAT, updatedAt, deleteAt]
- entryType [entryId, performedBy, changes]
- Field
- oldValue
- newValue
- timeStamp

@masteryoda schema for audit log: const mongoose = require("mongoose");

```
const AuditLogSchema = new mongoose.Schema({
  actionType: {
    type: String,
    enum: ["CREATE", "UPDATE", "DELETE", "LOGIN", "LOGOUT", "ACCESS"],
    required: true
  },

  entityType: { type: String, required: true }, // e.g., "User", "Event", "Settings"
  entityId: { type: mongoose.Schema.Types.ObjectId, required: false }, // ID of the entity
affected
```

```
  performedBy: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true }, // User/Admin who performed the action

  changes: [
    {
      field: { type: String, required: true }, // e.g., "email", "status", "password"
      oldValue: { type: mongoose.Schema.Types.Mixed, default: null },
      newValue: { type: mongoose.Schema.Types.Mixed, default: null }
    }
  ],

  ipAddress: { type: String, default: "" }, // Track user's IP address
  userAgent: { type: String, default: "" }, // Track device/browser details

  timestamp: { type: Date, default: Date.now }
});

module.exports = mongoose.model("AuditLog", AuditLogSchema);
```

5. NOTIFICATION MODEL

- recipient
- type
- message
- event_id
- read