Modelsn and End Points

13 June 2025 16:54

Absolutely bro! Here's the **final set of updated Mongoose models** for your **Employee Leave Tracker (Leotracker)** — 100% aligned with everything we discussed, including:

- Role-based control
- Domain validation logic
- Leave limits, usage, and unpaid leaves
- Employee approval by admin
- Optional profile fields
- Leave application tracking
- Admin comment handling

☑ 1. User Model (Admin + Employee)

```
const mongoose = require("mongoose");
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
    match: /^[\w.+\-]+@[\w\-]+\.[a-z]{2,}$/i,
  },
passwordHash: {
    type: String,
    required: true,
  },
role: {
    type: String,
    enum: ["admin", "employee"],
    default: "employee",
  },
isApproved: {
    type: Boolean,
    default: false, // Only admin can approve
  },
status: {
    type: String,
    enum: ["pending", "active", "rejected"],
    default: "pending",
  },
companyDomain: {
    type: String,
    required: true, // Extracted from email (e.g., ak.com)
  },
phone: {
    type: String,
    default: "", // Optional
  },
address: {
    type: String,
    default: "", // Optional
  },
profilePicture: {
    type: String,
    default: "", // Optional - can be URL or base64 string
  },
leaveBalance: {
    casual: { type: Number, default: 12 },  // Casual & Sick combined
    earned: { type: Number, default: 15 },
    wfh: { type: Number, default: 18 },
    unpaid: { type: Number, default: 0 },   // Always 0 - for display only
  },
leavesUsed: {
    casual: { type: Number, default: 0 },
    earned: { type: Number, default: 0 },
    wfh: { type: Number, default: 0 },
    unpaid: { type: Number, default: 0 },
  },
createdAt: {
    type: Date,
    default: Date.now,
  }
});
module.exports = mongoose.model("User", userSchema);
```

☑ **2. Leave Model**

```
const mongoose = require("mongoose");
const leaveSchema = new mongoose.Schema({
  employeeId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true,
  },
type: {
    type: String,
    enum: ["casual", "earned", "wfh", "unpaid"],
    required: true,
  },
startDate: {
    type: Date,
    required: true,
  },
endDate: {
    type: Date,
    required: true,
  },
isHalfDay: {
    type: Boolean,
    default: false,
  },
comment: {
    type: String,
    default: "",
  },
status: {
    type: String,
    enum: ["pending", "approved", "rejected"],
    default: "pending",
  },
adminComment: {
    type: String,
    default: "",
  },
createdAt: {
    type: Date,
    default: Date.now,
  },
});
module.exports = mongoose.model("Leave", leaveSchema);
```

🛠 3. [Optional] Company Model (For future multi-company support)

If in the future you want to manage multiple companies:

```
const mongoose = require("mongoose");
const companySchema = new mongoose.Schema({
  name: { type: String, required: true },
  domain: { type: String, required: true, unique: true }, // ak.com
  admins: [{ type: mongoose.Schema.Types.ObjectId, ref: "User" }],
});
module.exports = mongoose.model("Company", companySchema);
```

✅ You don't need this right now, but it's future-proofing your platform.

✅ Summary

Model	Purpose
User	Handles employees/admins, role-based access, approval, leave tracking
Leave	Manages each leave request's status, dates, types
Company	Optional: Manages domain-based company info (future-proofing)

Absolutely bro, let's now define the **complete RESTful API endpoints** for your **Leave Tracker System** based on the models we just created (User, Leave, and optional Company).
We'll break this down cleanly into:

✅ 1. Authentication Routes

Method	Endpoint	Description	Auth
POST	/api/auth/register	Employee registers with company email (goes to pending list)	❌
POST	/api/auth/login	Login for both Admin and Employee	❌
GET	/api/auth/me	Get current logged-in user details	✅

✅ 2. Admin Routes

Method	Endpoint	Description
GET	/api/admin/employees	Get all approved employees
GET	/api/admin/employees/pending	Get pending employee registrations
PUT	/api/admin/employees/:id/approve	Approve a pending employee
PUT	/api/admin/employees/:id/reject	Reject a pending employee
PUT	/api/admin/employees/:id/leave-balance	Update leave quota for an employee
GET	/api/admin/leaves	View all leave requests from all employees
PUT	/api/admin/leaves/:id/approve	Approve a leave request
PUT	/api/admin/leaves/:id/reject	Reject a leave request (with reason)
GET	/api/admin/reports	Get leave summary/report by employee, type

🛡 All above routes should be **protected** and **admin-only**.

☑ 3. Employee Routes

Method	Endpoint	Description
POST	/api/leaves	Apply for a new leave
GET	/api/leaves	Get all leaves of current employee
GET	/api/leaves/:id	Get details of one leave
PUT	/api/leaves/:id	Edit pending leave request
DELETE	/api/leaves/:id	Delete a pending leave request
GET	/api/leaves/calendar-view	Get calendar-compatible leave data
GET	/api/employee/leave-balance	Get current leave usage and remaining count
PUT	/api/employee/account	Update personal info (phone, address, etc.)

☑ 4. (Optional) Company Routes (if needed later)

Method	Endpoint	Description
GET	/api/companies	Get all registered companies
POST	/api/companies	Register a new company

🧱 Backend Folder Structure

```
backend/
├── controllers/
│   ├── auth.controller.js
│   ├── admin.controller.js
│   ├── employee.controller.js
│   ├── leave.controller.js
├── models/
│   ├── User.js
│   ├── Leave.js
│   └── Company.js   # Optional
├── routes/
│   ├── auth.routes.js
│   ├── admin.routes.js
│   ├── employee.routes.js
│   ├── leave.routes.js
├── middlewares/
│   ├── auth.middleware.js      # JWT verification
│   ├── role.middleware.js      # Admin/Employee checker
├── utils/
│   └── emailDomainCheck.js
├── config/
│   └── db.js                   # MongoDB connection
├── app.js
└── server.js
```

Here's a clean and scalable **frontend folder structure** suitable for modern frontend apps (especially using **React**, **Vite**, **Parcel**, or **Webpack** setups):

☑ Frontend Folder Structure (React Project Example)

```
/frontend
│
├── public/                 # Static files (index.html, icons, etc.)
│   └── index.html
│
├── src/                    # Source code
```

```
│    ├── assets/              # Images, fonts, logos, etc.
│    │    └── logo.png
│    │
│    ├── components/          # Reusable UI components
│    │    ├── Button.jsx
│    │    └── Navbar.jsx
│    │
│    ├── pages/               # Page-level components (route based)
│    │    ├── Home.jsx
│    │    └── About.jsx
│    │
│    ├── routes/              # All route definitions (optional)
│    │    └── AppRoutes.jsx
│    │
│    ├── layouts/             # Layout components (Header + Footer etc.)
│    │    └── MainLayout.jsx
│    │
│    ├── context/             # React Contexts (theme, auth, etc.)
│    │    └── AuthContext.js
│    │
│    ├── hooks/               # Custom hooks
│    │    └── useAuth.js
│    │
│    ├── services/            # API calls and external services
│    │    └── api.js
│    │
│    ├── utils/               # Utility functions/helpers
│    │    └── formatDate.js
│    │
│    ├── styles/              # Global CSS or Tailwind config
│    │    └── index.css
│    │
│    ├── App.jsx              # Main app component
│    ├── main.jsx             # ReactDOM render logic (entry point)
│    └── config.js            # Constants & environment-specific config
│
├── .env                      # Environment variables
├── package.json              # Dependencies and scripts
├── README.md                 # Project documentation
└── vite.config.js /          # Or webpack.config.js / parcel config
```

📦 **Optional Folders (Based on Project):**

- i18n/ → For translations and localization.
- store/ → If you're using Redux, Zustand, etc.
- tests/ → For unit/integration tests.
- types/ → For TypeScript type definitions.