In my Angular project, the `LocationModel` serves as the base model. The company has many houses or locations across the country, but the number of locations is fixed (i.e., 5 or 7 places in the entire country). Every location has a house for this company. That's why I want to build my project based on locations. The company has an Admin who maintains, oversees, or handles everything for all locations.

Example of a Location model:

```
export class LocationModel {
  id!: string; //Comment:  Primary key
  locationName!: string;
  addressLine!: string;
  city!: string;
  state!: string;
  postalCode!: string;
  countryName!: string;
  isActive!: boolean; //Comment:  A boolean flag indicating whether
the location is currently active or inactive
  createdAt!: Date; //Comment:  The timestamp when the location was
created
  updatedAt!: Date; //Comment:  The timestamp when the location was
last updated
  photo?: string; //Comment:  Optional field for the location's photo
}
```

Every house has a fixed number of departments, i.e., 4 or 6, based on the location. Each department has a fixed Department Head.

Example of a Department model:

```
export class DepartmentModel {
  id!: string; //Comment:  Primary Key
  departmentName!: string; //Comment:  Must provide department name
  description?: string;
  numberOfEmployees?: number; //Comment:  Number of employees will be
shown here
  payrollCalculationMethod!: 'Weekly' | 'Monthly';
```

```
  LocationModel!: {
    id: string;
    locationName: string | undefined;
    addressLine: string | undefined;
    city: string | undefined;
    state: string | undefined;
    postalCode: string | undefined;
    countryName: string | undefined;
  };

  UserModel!: {
    id: string;
    firstName: string;
    lastName: string;
    role: 'HR' | 'Employee' | undefined;
  };
}
```

Every department has an unfixed number of employees. Employees come and leave the house, so I want to keep their records.

Example of an Employee model:

```
export class EmployeeModel {
  id!: string; //Comment:  Primary Key

  UserModel!: {
    id: string;
    firstName: string | undefined;
    lastName: string | undefined;
    email: string | undefined;
    role: 'HR' | 'Employee' | undefined;
    profilePhoto: string | undefined;
    gender: 'Male' | 'Female' | 'Other' | undefined;
    contact: string | undefined;
    nidNo: number | undefined;
    joiningDate: Date | undefined;
```

```
    hourlyRate: number | undefined; //Comment:  Hourly rate 150 for
employees and 250 for HR
  };

  DepartmentModel!: {
    id: string;
    departmentName: string | undefined;
    payrollCalculationMethod: 'Weekly' | 'Monthly' | undefined;
  };

  LocationModel!: {
    id: string;
    locationName: string | undefined;
    addressLine: string | undefined;
    city: string | undefined;
    state: string | undefined;
    postalCode: string | undefined;
    countryName: string | undefined;
  };
}
```

All users have a role provided by the Admin. The Admin account is the fixed account in this system. The Admin can create department heads. Employees can't generate or register accounts on this system. Department heads can create employees. I call the department head like an HR. There are three types of users in this project.

Example of a Role-based Authentication Model:

```
export interface AuthResponse {
  token: string; //Comment:  Generated and stored token

  expiresIn?: number; //Comment:  Token expiration time in seconds
  issuedAt?: Date; //Comment:  When the token was issued

  role: 'Admin' | 'HR' | 'Employee';
}
```

I want to create a dynamic registration page with two options. When the Admin logs into their dashboard and wants to create an HR, or when the HR logs into their dashboard and wants to create an employee, one common HTML page will show where a necessary data collection form will be displayed.

Example of a User model:

```
export class UserModel {
  id!: string; //Comment:  Primary Key
  firstName!: string;
  lastName!: string;
  email!: string; //Comment:  Email address, must be unique
  userName!: string; //Comment:  Username for login
  password!: string; //Comment:  Password for authentication
  role!: 'Admin' | 'HR' | 'Employee'; //Comment:  Role of the user,
restricted to specific values
  profilePhoto?: string; //Comment:  Optional profile photo
  gender!: 'Male' | 'Female' | 'Other';
  contact!: string; //Comment:  Contact number
  nidNo!: number; //Comment:  National ID number, must be provided and
unique
  joiningDate!: Date; //Comment:  For payment calculation
  hourlyRate!: number; //Comment:  Salary based on the user's role

  createdAt!: Date; //Comment:  Account creation date
  updatedAt!: Date; //Comment:  Last update date

  DepartmentModel!: {
    id: string;
    departmentName: string | undefined;
    payrollCalculationMethod: 'Weekly' | 'Monthly' | undefined;
  };
}
```

The Admin can perform all tasks (CRUD) from this system. The Admin can handle payroll, performance, grant leave, check attendance, and provide feedback for HR. Similarly, HR can manage payroll, and performance, grant leave, check attendance, and provide feedback for employees.

Example of a Payroll model:

```
export class PayrollModel {
  id!: string; //Comment:  Primary Key

  UserModel!: {
    id: string;
    firstName: string | undefined;
    lastName: string | undefined;
    role: 'HR' | 'Employee' | undefined;
    profilePhoto: string | undefined;
    contact: string | undefined;
    nidNo: number | undefined;
    hourlyRate: number | undefined; //Comment:  Hourly rate 150 for
employees and 250 for HR
  };

  EmployeeModel!: {
    id: string | undefined; //Comment:  Using this UserModel, an
employee will be created. After creating a new employee, they have a
unique ID
  };

  performanceBonuses!: number; //Comment:  1* = 200, 2* = 400, 3* =
600, 4* = 800, 5* = 1600
  insurance!: number; //Comment:  1000 for employees, 3000 for HR
monthly
  medicare!: number; //Comment:  5000 for employees, 10000 for HR
  deductions!: number; //Comment:  Deductions (e.g., tax, insurance)

  netPay!: number; //Comment:  Net pay after deductions

  paymentDate!: Date; //Comment:  Date when payment was made

  payPeriodStart!: Date; //Comment:  Start date of the pay period
  payPeriodEnd!: Date; //Comment:  End date after 30 days from the
start of the pay period
```

```
  overtimeExemption!: boolean; //Comment:  Yes, Newcomer or older than
50 years. Or No.

  overtimeHourlyRate!: number; //Comment:  Overtime hourly rate will
add half of their main rate. Assume 150/2 = 75 + 150 = 225 for
employees and 250/2 = 125 + 250 = 375 for HR

  yearlySickDay!: number; //Comment:  10 days reserved

  status!: 'Paid' | 'Pending' | 'Overdue'; //Comment:  Status of the
payroll
  //Comment:  Create a map that returns netPay
  //Comment:  static mapPayroll(payroll: PayrollModel): PayrollModel {
  //Comment:    return {
  //Comment:      ...payroll,
  //Comment:      netPay: payroll.UserModel.salary! - payroll.tax! -
payroll.insurance!,
  //Comment:    };
  //Comment:  }
}
```

Payment will be provided based on attendance, where working hours and working calendar month will be calculated. Performance rating bonuses, overtime, sick day salary, insurance, medicare, and deductions will be considered.

- **Hourly Rate**: 8 hours a day * hourly rate will be calculated. Assume an hourly rate of 150 for employees and 250 for HR.
- **Overtime Rate**: The overtime hourly rate will add half of their main rate. Assume 150/2 = 75 + 150 = 225 for employees and 250/2 = 125 + 250 = 375 for HR.

Example of a Feedback model:

```
export class FeedbackModel {
  id!: string;

  UserModel!: {
    id: string;
    firstName: string | undefined;
    lastName: string | undefined;
```

```
    role: 'HR' | 'Employee' | undefined;
    profilePhoto: string | undefined;
  };

  rating!: number; //Comment:  User rating will be shown here. 1 to 5
scale
  comments!: string; //Comment:  User comments
  feedbackDate!: Date; //Comment:  Feedback creation date
}
```

Example of a Performance model:

```
export class PerformanceModel {
  id!: string; //Comment:  Unique identifier for the performance
record

  goals!: boolean; //Comment:  Check if the employee has achieved the
goal

  achievements!: string; //Comment:  Specific achievements during the
review period

  reviewDate!: Date; //Comment:  Date when the review was conducted

  rating!: number; //Comment:  Example: 1 to 5 scale

  UserModel!: {
    id: string;
    firstName: string | undefined;
    lastName: string | undefined;
    role: 'HR' | 'Employee' | undefined;
    profilePhoto: string | undefined;
  };

  comments!: string; //Comment:  Areas of Improvement based on
comments about the employee's performance
}
```

Example of an Attendance model:

```
export class AttendanceModel {
  id!: string;

  workingDay!: Date; //Comment:  Must be provided to calculate a
user's salary or grant leave on a specific day

  workHours!: number; //Comment:  Example: 8.00

  UserModel!: {
    id: string;
    firstName: string | undefined;
    lastName: string | undefined;
    role: 'HR' | 'Employee' | undefined;
  };

  department!: DepartmentModel;

  inTime!: Date; //Comment:  In time for the day

  outTime!: Date; //Comment:  Out time for the day

  status!: 'Present' | 'Absent' | 'On Leave' | 'Sick' | 'Holiday';
//Comment:  Status of the attendance record
}
```

Example of a Leave model:

```
export class LeaveModel {
  id!: string;

  leaveType!: 'Annual' | 'Sick' | 'Maternity' | 'Paternity' |
'Compassionate' | 'Unpaid'; //Comment:  Type of leave taken

  startDate!: Date; //Comment:  Start date of the leave
```

```
  endDate!: Date; //Comment:  End date of the leave

  totalLeaveDays!: number; //Comment:  The system will calculate the
total number of days between start and end dates

  UserModel!: {
    id: string;
    firstName: string | undefined;
    lastName: string | undefined;
    role: 'HR' | 'Employee' | undefined;
  };

  status!: 'Pending' | 'Approved' | 'Rejected'; //Comment:  Status of
the leave request

  approvalDate!: Date; //Comment:  Date when the leave was approved or
rejected

  comments?: string; //Comment:  Optional comments or reasons for
approval/rejection
}
```