1. Database Design Specification

For this system, we will use a **Referential Document Model**. While MongoDB is NoSQL, the nature of payroll requires strong relationships between employees, their financial records (Salfah/Geza), and their performance data.

1.1 Database Collections & Schema Definitions

Collection	Description	Relationships
`organizations`	Multi-tenant root (if applicable) or company metadata.	1:M with Departments
`departments`	Organizational units (e.g., Sales, Ops).	1:M with Employees
`employees`	Core profile, basic salary, and system roles.	1:M with Tasks, 1:1 with active Debt
`kpi_configs`	Weighted KPI definitions per department.	M:1 with Departments
`tasks`	Individual task assignments and proof of work.	M:1 with Employees (AssignedTo/By)
`performance_reviews`	Periodic scoring and qualitative feedback.	M:1 with Employees
`payroll_slips`	Immutable monthly financial records.	M:1 with Employees
`debts` **(Salfah)**	Loan tracking and installment schedules.	1:1 with active Employee
`penalties` **(Geza)**	Disciplinary deductions.	M:1 with Employees
`audit_logs`	Write-only log of all sensitive data changes.	M:1 with Employees (Actor)

1.2 Detailed Schema Breakdown

A. Employees Collection

JSON

```json
{
  "_id": "ObjectId",
  "fullName": "String",
  "fullNameArabic": "String",
  "nationalId": "String", // Encrypted at App-level
  "email": "String",
  "role": { "type": "String", "enum": ["Employee", "Manager", "HR",
"Finance", "Admin"] },
  "joiningDate": "Date",
  "departmentId": "ObjectId",
  "managerId": "ObjectId",
  "financials": {
    "basicSalary": "Decimal128",
    "allowances": "Decimal128",
    "insuranceSalary": "Decimal128" // Clamped: 2,300 - 14,500
  },
  "status": { "type": "String", "default": "Active" }
}
```

B. PayrollSlips Collection (Financial Snapshot)

Note: We store the "Calculation Snapshot" to ensure historical accuracy even if tax laws change later.

JSON

```
{
  "_id": "ObjectId",
  "employeeId": "ObjectId",
  "period": { "month": "Number", "year": "Number" },
  "earnings": {
    "basic": "Decimal128",
    "overtime": "Decimal128",
    "bonus": "Decimal128",
    "gross": "Decimal128"
  },
  "deductions": {
    "socialInsurance": "Decimal128", // 11% of InsuranceSalary
    "incomeTax": "Decimal128",      // Based on Law 175/2023
    "martyrsFund": "Decimal128",    // 0.05% of Gross
    "salfah": "Decimal128",
    "geza": "Decimal128"
  },
  "netSalary": "Decimal128",
  "isCapped": "Boolean",            // True if 50% cap was triggered
  "status": "String"                // ["Draft", "Approved", "Paid"]
}
```

C. Tasks & Performance

JSON

```
{
  "tasks": {
    "assignedTo": "ObjectId",
    "weight": "Number", // 1-10
    "status": "String",
    "dueDate": "Date",
    "proofUrl": "String"
  },
  "performance_reviews": {
    "employeeId": "ObjectId",
    "overallScore": "Number", // 0-100
    "kpiBreakdown": [
       { "kpiId": "ObjectId", "score": "Number" }
    ],
    "employeeAcknowledged": "Boolean"
  }
}
```

1.3 Entity Relationship Diagram (ERD)

Code snippet

```
erDiagram
    DEPARTMENTS ||--o{ EMPLOYEES : "belongs to"
    EMPLOYEES ||--o{ TASKS : "assigned to"
    EMPLOYEES ||--o{ PERFORMANCE_REVIEWS : "evaluated"
    EMPLOYEES ||--o{ PAYROLL_SLIPS : "paid via"
    EMPLOYEES ||--o{ PENALTIES : "receives"
    EMPLOYEES ||--o{ DEBTS : "borrows"
    DEPARTMENTS ||--o{ KPI_CONFIGS : "defined for"
    KPI_CONFIGS ||--o{ PERFORMANCE_REVIEWS : "measured by"
    EMPLOYEES ||--o{ AUDIT_LOGS : "triggered by"

    EMPLOYEES {
        string fullName
        string nationalId
        decimal basicSalary
        string role
    }
    PAYROLL_SLIPS {
        date period
        decimal grossSalary
        decimal netSalary
        decimal incomeTax
    }
    DEBTS {
        decimal totalAmount
        decimal monthlyInstallment
        string status
    }
```

1.4 Design Rationale for Database

1. **Schema Flexibility:** MongoDB allows us to store the `Tax_Config` as a versioned document. When Egyptian tax law changes (e.g., Law 175/2023), we simply insert a new config document without altering the structure of 7 years of historical data.
2. **Encapsulation:** By using sub-documents for `earnings` and `deductions` within the `PayrollSlips` collection, we can retrieve a full monthly report in a single query, significantly hitting the **< 2s load time** requirement.

3. **Data Integrity:** We will implement **Mongoose Middleware** to enforce the **50% deduction cap** and the **5-day Geza limit** at the database layer to prevent invalid data entry.

2. System Architecture

This section describes the high-level technical structure of the **IPPS** platform, focusing on the communication between the **Next.js** frontend and the **Node.js/Express** backend.

2.1 High-Level Architecture Overview

The system follows a **Client-Server Architecture** with a clear separation of concerns to ensure scalability and maintainability.

- **Frontend (Next.js):** Acts as the presentation layer. It utilizes **Server-Side Rendering (SSR)** for protected dashboards to ensure data is fresh and **Client-Side Rendering (CSR)** for interactive elements like task management and real-time form validations.
- **Backend (Node.js/Express):** Functions as the Logic Layer. It hosts the **Egyptian Payroll Engine**, handles authentication via JWT, and enforces business rules (like the 50% deduction cap) before committing data to MongoDB.
- **Communication:** The Frontend communicates with the Backend via a **RESTful API**. Secure data transfer is ensured through **TLS 1.3** encryption, and all requests are authenticated using Bearer tokens.
- **Data Persistence:** MongoDB serves as the primary data store, using Mongoose for schema enforcement and validation.

2.2 Architecture Diagram

Code snippet

```
graph TB
    subgraph "Client Layer (Next.js)"
        A[Web Browser / Mobile Web]
        B[State Management - React Query]
        C[UI Components - Shadcn/Tailwind]
    end

    subgraph "Logic Layer (Node.js/Express)"
        D[API Gateway / Routing]
        E[Auth Middleware - JWT]
        F[Payroll & Tax Engine]
        G[Performance Sync Service]
        H[Notification Service - Email]
    end

    subgraph "Data Layer (MongoDB)"
        I[(MongoDB Cluster)]
        J[GridFS / S3 - Task Attachments]
    end

    A <-->|HTTPS / REST API| D
    D --> E
    E --> F
    E --> G
    F <--> I
    G <--> I
    H -->|SMTP| A
    G --> J
```

---

3. API Specification

The following table outlines the core API endpoints required to fulfill the P0 (Must-Have) requirements of the system.

Endpoint	**Method**	**Description**	**Request Body**	**Success Response (200 OK)**
`/api/auth/login`	`POST`	Authenticates user and returns JWT.	`{email, password}`	`{token, userObj}`
`/api/employees`	`GET`	Retrieves all employees (Admin/HR only).	None	`[{employeeObj}]`
`/api/payroll/calculate`	`POST`	Triggers payroll engine for a specific month.	`{month, year, empIds[]}`	`{summary, slips[]}`
`/api/tasks`	`POST`	Assigns a new task to an employee.	`{title, dueDate, assignedTo, weight}`	`{taskObj}`
`/api/tasks/:id/complete`	`PATCH`	Marks task as done and triggers score sync.	`{proofUrl, comments}`	`{updatedScore}`

Endpoint	Method	Description	Request Body	Success Response (200 OK)
`/api/debts`	`POST`	Records a new "Salfah" for an employee.	`{amount, installments, reason}`	`{debtObj}`
`/api/performance/sync`	`POST`	Recalculates scores based on completed tasks.	`{employeeId, period}`	`{overallScore}`

---

4. Design Rationale for Architecture

1. **Next.js & Node.js (Unified Language):** Using TypeScript across the entire stack reduces context switching for developers and allows for shared type definitions (e.g., the `PayrollSlip` interface), which minimizes integration errors.
2. **Stateless Backend:** By using JWT for authentication, the Node.js server remains stateless. This allows the system to scale horizontally using Docker containers to handle peak loads during the end-of-month payroll window.
3. **Modular Engine:** The **Payroll Engine** is designed as a standalone service within the Express app. This ensures that changes to Egyptian Tax Law (e.g., Law 175/2023) only require updates to one module without affecting the Performance or Task modules.

4. Component Design

This section breaks down the system into modular, reusable units for both the Frontend and Backend to ensure the code follows the DRY (Don't Repeat Yourself) principle.

4.1 Frontend Components (Next.js + Shadcn UI)

The frontend is organized into atomic components and higher-order layouts to handle the bilingual (Arabic/English) requirements and complex financial data.

- **Core Layout Components:**
  - `DashboardLayout`: Handles the sidebar navigation, top bar with language switcher (AR/EN), and user profile dropdown.

  - `AuthGuard`: A High-Order Component (HOC) that checks JWT validity and Role-Based Access Control (RBAC) before rendering protected pages.
- **Shared UI Components (Shadcn/Tailwind):**
  - `StatCard`: Displays high-level metrics (e.g., Performance Score, Next Payslip amount) with trend indicators.
  - `DataTable`: A sortable/filterable table component optimized for large payroll sheets, including horizontal scrolling for mobile responsiveness.
  - `TaskCard`: A visual representation of a task with status badges (Pending, Overdue) and action buttons.
- **Module-Specific Components:**
  - `PayrollCalculator`: A complex form with real-time validation for bulk payroll processing.
  - `KPIWeightConfig`: An interactive interface for HR to adjust KPI weightages per department.
  - `PayslipPDF`: A server-side component to generate and format the Egyptian-compliant payslip for printing.

4.2 Backend Modules & Services (Node.js/Express)

The backend is structured as a service-oriented architecture where each core domain has its own logic controller.

- **`AuthService`**: Manages JWT signing, verification, and password hashing (using Bcrypt).
- **`PayrollEngine`**: The heart of the system. It contains the logic for Social Insurance caps (2,300–14,500 EGP), progressive Income Tax (Law 175/2023), and the 50% deduction cap.
- **`PerformanceService`**: Handles the sync between completed tasks and employee overall scores based on department-specific KPI weightages.
- **`AuditService`**: An immutable logging service that records every financial change, fulfilling the 7-year Egyptian tax audit requirement.
- **`NotificationService`**: A wrapper for sending email reminders for overdue tasks and salary slip notifications.

---

5. Infrastructure & DevOps

This section details how to package and deploy the IPPS platform in a production-ready environment.

5.1 Docker Configuration

`Dockerfile` (Backend/Node.js):

Dockerfile

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install --production
COPY . .
EXPOSE 5000
CMD ["node", "dist/index.js"]
```

`docker-compose.yml`:

YAML

```yaml
version: '3.8'
services:
  backend:
    build: ./backend
    ports:
      - "5000:5000"
    environment:
      - MONGO_URI=mongodb://mongo:27017/ipps
      - JWT_SECRET=${JWT_SECRET}
    depends_on:
      - mongo
  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
  mongo:
    image: mongo:latest
    ports:
      - "27017:27017"
    volumes:
      - mongo_data:/data/db
volumes:
  mongo_data:
```

5.2 Deployment Strategy

- **Frontend Deployment**: Deployed to **Vercel** for optimal Next.js performance and global CDN edge caching.
- **Backend & DB Deployment**: Deployed using **Docker containers** on a cloud provider with a Middle East region (e.g., AWS Bahrain or Azure UAE) to minimize latency for Egyptian users and address data sovereignty concerns.
- **CI/CD Pipeline**: GitHub Actions will be used to run automated tests for the Payroll Engine before any production deployment to ensure zero calculation regressions.

---

6. Design Rationale

1. **Next.js + Node.js (Full-stack TypeScript)**: Ensures type safety across the entire application, preventing "undefined" errors in critical financial calculations.
2. **MongoDB for Financial Logs**: While payroll is structured, the "Audit Log" and "Performance History" can vary in depth. MongoDB’s document model allows us to store complex snapshots of calculations without rigid migrations.
3. **Shadcn UI**: Provides high-quality, accessible components that natively support RTL (Right-to-Left) layouts, which is essential for the Egyptian market's Arabic requirement.
4. **Service-Oriented Logic**: By isolating the `PayrollEngine`, the system can be updated instantly when the Egyptian government announces new tax brackets without taking the entire HR portal offline.