System Design Document: Worker Time Tracking & Payroll

Author: fnenow & Copilot **Date:** 2025-05-29

1. User Diagrams (Conceptual Use Case Style)

This outlines the main actors and their primary interactions with the system.

Actors:

- **Worker**
- **Administrator**

Use Cases:

- **Worker:**
    - Clock In (selects Project)
    - Clock Out
    - View Own Time Entries
- **Administrator:**
    - Log In
    - View Worker Status Dashboard
    - Manually Clock In/Out Worker
    - Manage Projects (Create, View, Update, Archive)
    - Manage Workers (Create, View, Update worker info)
    - Manage Worker Pay Rates (Add new, view history - ensures old rates are not overwritten)

- Generate Payroll Report (with filters for period, worker, project)
  - *Includes:* Calculate Regular Pay
  - *Includes:* Calculate Daily Overtime Pay
  - *Includes:* Calculate Weekly Overtime Pay
- Update Bill Number for Time Entries (on Payroll Report)
- Update Paid Date for Time Entries (on Payroll Report)
- Log Out

---

2. Folder Structure (Conceptual)

This is a typical way projects might be organized.

A. Frontend (e.g., using plain HTML/CSS/JS or a simple framework)

```
frontend/
├── index.html                  // Worker Clock In/Out page
├── worker_view.html            // Worker's view of their hours
├── admin_login.html            // Admin login page
├── admin/
│   ├── dashboard.html          // Admin dashboard (worker status)
│   ├── projects.html           // Admin project management
│   ├── workers.html            // Admin worker management (info & pay rates)
```

```
|   ├── payroll.html          // Admin payroll report
|   └── admin_layout.html     // Optional: template for common admin page elements
├── css/
|   ├── style.css             // Main stylesheet
|   └── admin_style.css       // Styles specific to admin pages
├── js/
|   ├── main.js               // JS for index.html (worker clocking)
|   ├── worker_view.js        // JS for worker_view.html
|   ├── auth.js               // JS for admin_login.html
|   ├── admin_dashboard.js    // JS for admin/dashboard.html
|   ├── admin_projects.js     // JS for admin/projects.html
|   ├── admin_workers.js      // JS for admin/workers.html
|   ├── admin_payroll.js      // JS for admin/payroll.html
|   └── utils.js              // Common utility functions (API calls, date formatting)
└── assets/
```

```
    └── images/               // Logos, icons, etc.
```

B. Backend (e.g., Node.js with Express.js)

```
backend/
├── src/
│   ├── api/                  // API route definitions
│   │   ├── authRoutes.js     // Handles /api/admin/login
│   │   ├── clockRoutes.js    // Handles /api/clock/*
│   │   ├── projectRoutes.js  // Handles /api/projects/*
│   │   ├── workerRoutes.js   // Handles /api/workers/*, /api/admin/worker-statuses
│   │   └── payrollRoutes.js  // Handles /api/payroll-report, /api/time-entries/*
│   ├── controllers/          // Request handlers (business logic)
│   │   ├── authController.js
│   │   ├── clockController.js
│   │   ├── projectController.js
│   │   ├── workerController.js
│   │   └── payrollController.js // Contains the core overtime logic
```

```
|   ├── models/               // Database interaction logic (e.g., using an ORM or direct queries)
|   |   ├── workerModel.js
|   |   ├── projectModel.js
|   |   ├── payRateModel.js
|   |   └── clockEntryModel.js
|   ├── services/             // More complex business logic, helper services
|   |   └── payrollService.js // Could house the detailed OT calculation steps
|   ├── middleware/           // Express middleware
|   |   ├── authMiddleware.js // Protects admin routes
|   |   └── errorMiddleware.js// Global error handling
|   ├── config/               // Configuration files
|   |   ├── db.js             // Database connection setup
|   |   └── index.js          // Environment variables, etc.
|   ├── utils/                // Utility functions (date calculations, etc.)
|   |   └── timeUtils.js
```

```
|   ├── app.js                // Main Express application setup (middleware, routes)
|   └── server.js             // Starts the HTTP server
├── package.json              // Project dependencies and scripts
├── .env                      // Environment variables (DB_URI, JWT_SECRET, etc.)
└── tests/                    // Unit and integration tests
    └── payroll.test.js       // Example: test payroll calculations
```

---

3. User Function Chart (Mapping UI Elements to High-Level Functions)

Screen	**UI Element / Section**	**User Action / Function**	**Actor**
index.html	Project Dropdown	Select Project	Worker
	"Clock In" Button	Initiate Clock-In for selected project (records time, timezone, snapshots pay rate)	Worker
	"Clock Out" Button	Initiate Clock-Out (records time)	Worker
	"View My Hours" Link	Navigate to worker_view.html	Worker

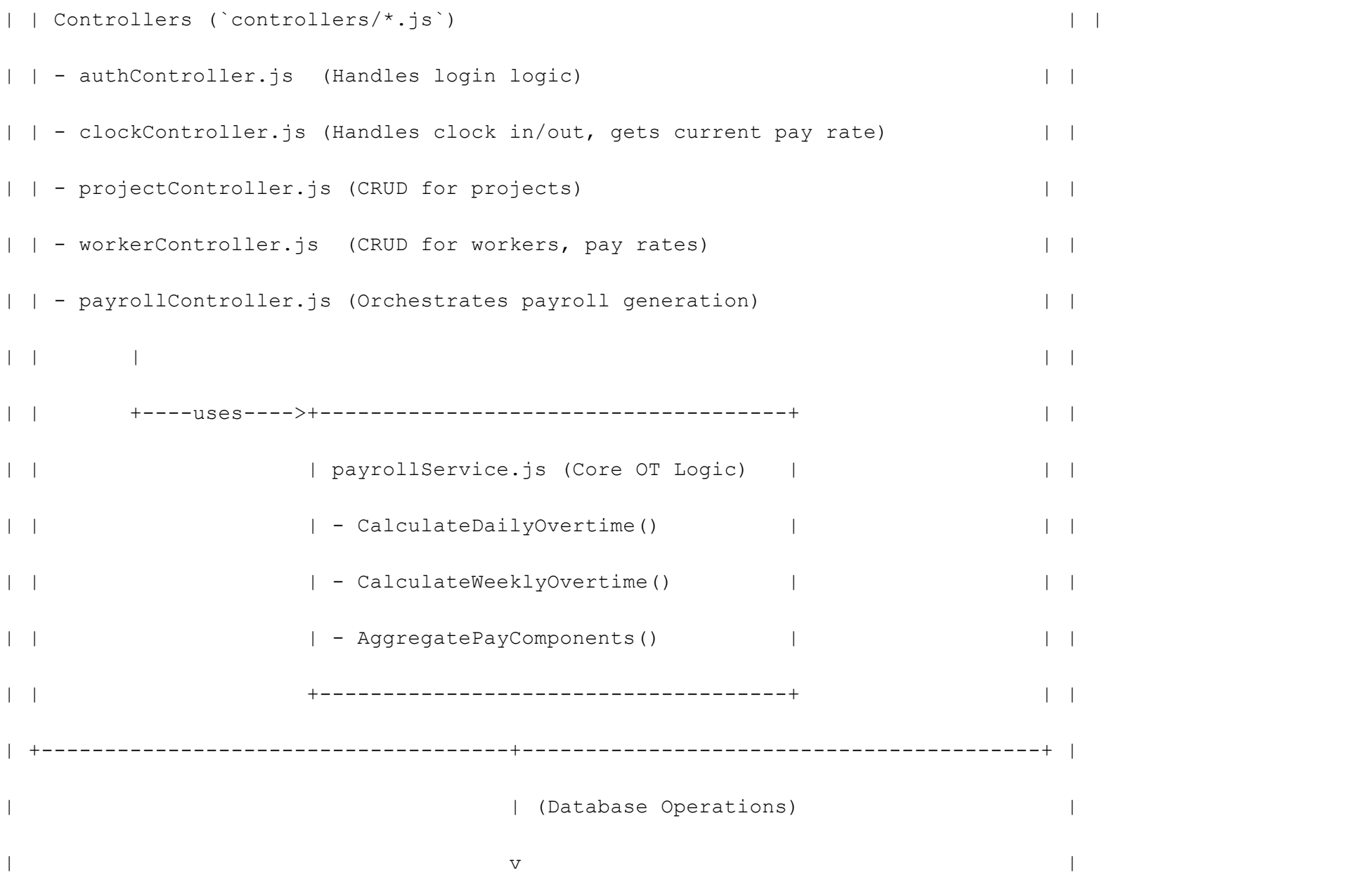
Screen	UI Element / Section	User Action / Function	Actor
worker_view.html	Time Entries Table	View own historical clock entries (date, project, times, duration)	Worker
admin_login.html	Username/Password Fields	Enter Credentials	Admin
	"Login" Button	Submit credentials for authentication	Admin
admin/dashboard.html	Worker Status Table	View real-time status of all workers (In/Out, Project, Duration)	Admin
	"Manual Clock In/Out" Btn	Force clock-in or clock-out for a worker	Admin
admin/projects.html	Project List Table	View all projects (name, description, status)	Admin
	"[+ New Project]" Button	Open form to create a new project	Admin
	Project Form	Enter/Edit project details (name, description)	Admin
	"[Edit]" Button (per project)	Open form to edit existing project details	Admin

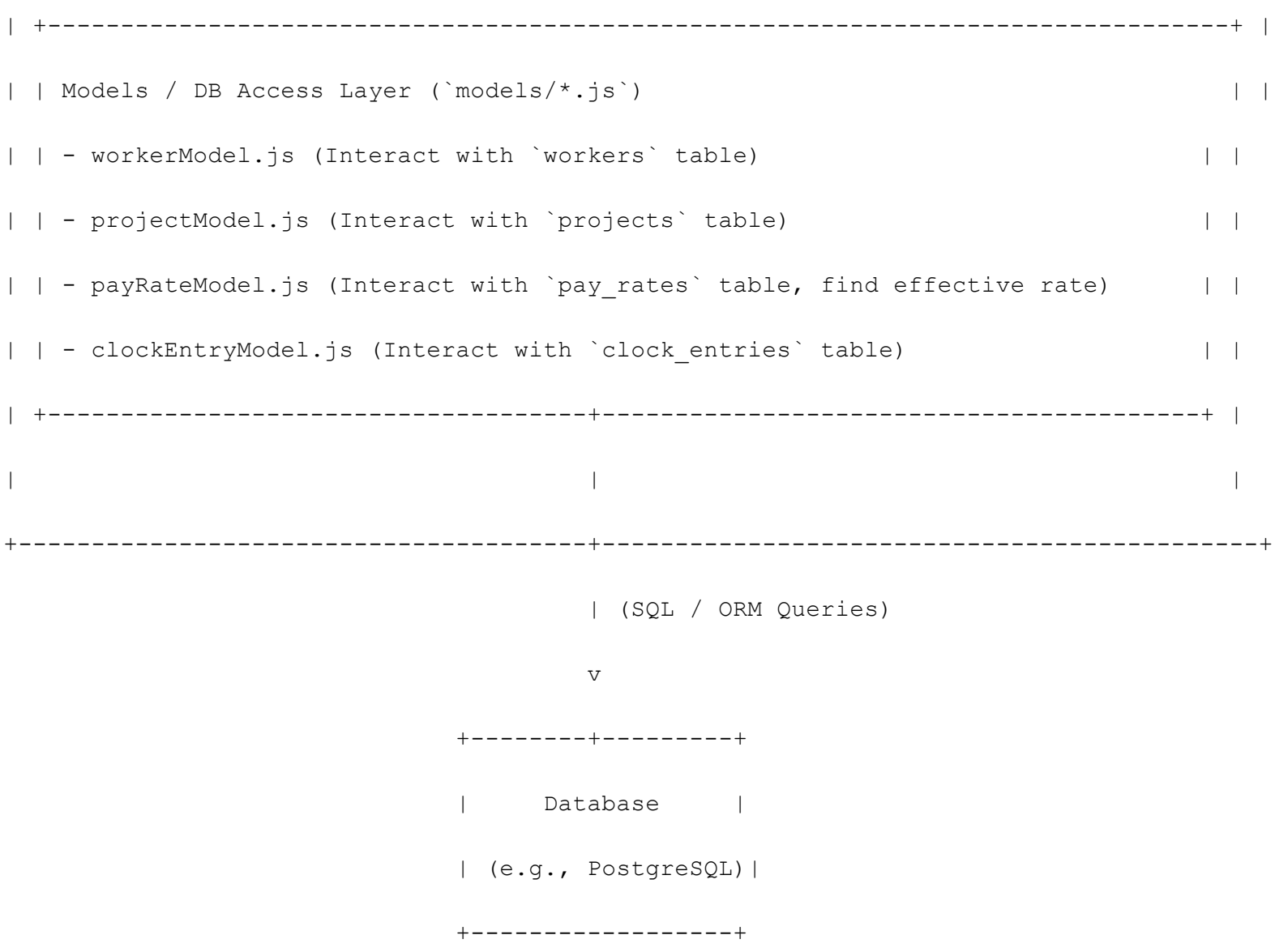
Screen	UI Element / Section	User Action / Function	Actor
admin/workers.html	Worker List Table	View all workers (name, ID, contact)	Admin
	"[+ New Worker]" Button	Open form to add a new worker	Admin
	Worker Info Form	Enter/Edit worker static details	Admin
	"[Edit Info]" Button	Open form to edit worker details	Admin
	"[Manage Pay Rates]" Button	View historical pay rates for a worker; Add new pay rate with effective start date	Admin
admin/payroll.html	Filter Controls	Select Date Period, Worker(s), Project(s)	Admin
	"Generate Report" Button	Trigger payroll calculation (Reg, Daily OT, Weekly OT) & display report	Admin
	Payroll Report Table	View calculated pay breakdown (Reg Hours/Pay, Daily OT H/P, Weekly OT H/P, Total Pay)	Admin

Screen	UI Element / Section	User Action / Function	Actor
	Bill Number Input Field	Enter/Update customer bill number associated with time entries	Admin
	Worker Paid Date Picker	Select/Update date worker was paid for time entries	Admin
	"Update Bill/Paid" Button	Save changes to bill number and paid date	Admin

---

4. Backend Chart (Conceptual Component/Module Interaction)


Controllers (`controllers/*.js`)
- authController.js (Handles login logic)
- clockController.js (Handles clock in/out, gets current pay rate)
- projectController.js (CRUD for projects)
- workerController.js (CRUD for workers, pay rates)
- payrollController.js (Orchestrates payroll generation)
uses
payrollService.js (Core OT Logic)
- CalculateDailyOvertime()
- CalculateWeeklyOvertime()
- AggregatePayComponents()
(Database Operations)


Models / DB Access Layer (`models/*.js`)
- workerModel.js (Interact with `workers` table)
- projectModel.js (Interact with `projects` table)
- payRateModel.js (Interact with `pay_rates` table, find effective rate)
- clockEntryModel.js (Interact with `clock_entries` table)
(SQL / ORM Queries)
Database
(e.g., PostgreSQL)

Key Backend Flows Highlighted in Chart:

- **Request Lifecycle:** Request -> Router -> Auth (if needed) -> Specific Route Handler -> Controller.
- **Controller Responsibilities:** Validate input, call necessary services/models, format response.
- **PayrollService:** Encapsulates the complex overtime and pay calculation logic, making the `payrollController` cleaner.
- **Models:** Abstract database interactions. For example, `payRateModel.getEffectiveRate(workerId, date)` would find the correct historical pay rate.

End of Document

SQL CREATE DATABASES

```
-- ------------------------------------------------------
-- Table workers
-- ------------------------------------------------------
CREATE TABLE IF NOT EXISTS workers (
  worker_id SERIAL PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  employee_id_number VARCHAR(100) UNIQUE,
  phone_number VARCHAR(50),
  address TEXT,
  email VARCHAR(255) UNIQUE, -- Added for potential login/notifications
  is_active BOOLEAN DEFAULT TRUE, -- To deactivate workers instead of deleting
  -- password_hash VARCHAR(255), -- For authentication if you implement it
  created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

-- Trigger to automatically update updated_at timestamp
CREATE OR REPLACE FUNCTION trigger_set_timestamp()
RETURNS TRIGGER AS $$
BEGIN
  NEW.updated_at = NOW();
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER set_workers_timestamp
BEFORE UPDATE ON workers
FOR EACH ROW
EXECUTE FUNCTION trigger_set_timestamp();
```

```
-- -----------------------------------------------------
-- Table projects
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS projects (
  project_id SERIAL PRIMARY KEY,
  project_name VARCHAR(255) NOT NULL UNIQUE,
  description TEXT,
  is_active BOOLEAN DEFAULT TRUE,
  created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

CREATE TRIGGER set_projects_timestamp
BEFORE UPDATE ON projects
FOR EACH ROW
EXECUTE FUNCTION trigger_set_timestamp();

-- -----------------------------------------------------
-- Table pay_rates
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS pay_rates (
  pay_rate_id SERIAL PRIMARY KEY,
  worker_id INT NOT NULL,
  rate_amount DECIMAL(10, 2) NOT NULL CHECK (rate_amount > 0),
  effective_start_date DATE NOT NULL,
  effective_end_date DATE, -- Null means it's the current/ongoing rate
  created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT uq_worker_effective_start UNIQUE (worker_id, effective_start_date), -- A worker can only have
one rate starting on a specific day
```

```
  CONSTRAINT chk_dates CHECK (effective_end_date IS NULL OR effective_end_date > effective_start_date)
);

CREATE TRIGGER set_pay_rates_timestamp
BEFORE UPDATE ON pay_rates
FOR EACH ROW
EXECUTE FUNCTION trigger_set_timestamp();

-- ----------------------------------------------------
-- Table clock_entries
-- ----------------------------------------------------
CREATE TABLE IF NOT EXISTS clock_entries (
  entry_id BIGSERIAL PRIMARY KEY,
  worker_id INT NOT NULL,
  project_id INT NOT NULL,
  clock_in_time TIMESTAMPTZ NOT NULL,
  clock_out_time TIMESTAMPTZ,
  original_timezone VARCHAR(100), -- Timezone from worker's device at clock-in
  duration_minutes INT, -- Calculated on clock-out
  recorded_pay_rate DECIMAL(10, 2), -- Pay rate at the time of clock-in
  notes TEXT,
  bill_number VARCHAR(100), -- For invoicing/billing reference
  worker_paid_date DATE, -- Date the worker was paid for this entry
  is_manually_adjusted BOOLEAN DEFAULT FALSE, -- If entry was edited by admin
  created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);

CREATE TRIGGER set_clock_entries_timestamp
BEFORE UPDATE ON clock_entries
FOR EACH ROW
```

```
EXECUTE FUNCTION trigger_set_timestamp();

-- ------------------------------------------------------
-- Add Foreign Key Constraints
-- ------------------------------------------------------
ALTER TABLE pay_rates
  ADD CONSTRAINT fk_pay_rates_worker
  FOREIGN KEY (worker_id)
  REFERENCES workers (worker_id)
  ON DELETE RESTRICT; -- Or CASCADE if you want pay rates deleted when worker is deleted

ALTER TABLE clock_entries
  ADD CONSTRAINT fk_clock_entries_worker
  FOREIGN KEY (worker_id)
  REFERENCES workers (worker_id)
  ON DELETE RESTRICT; -- Prevent deleting worker if they have time entries

ALTER TABLE clock_entries
  ADD CONSTRAINT fk_clock_entries_project
  FOREIGN KEY (project_id)
  REFERENCES projects (project_id)
  ON DELETE RESTRICT; -- Prevent deleting project if it has time entries

-- ------------------------------------------------------
-- Add Indexes for Performance
-- ------------------------------------------------------
CREATE INDEX IF NOT EXISTS idx_pay_rates_worker_id ON pay_rates(worker_id);
CREATE INDEX IF NOT EXISTS idx_clock_entries_worker_id ON clock_entries(worker_id);
CREATE INDEX IF NOT EXISTS idx_clock_entries_project_id ON clock_entries(project_id);
CREATE INDEX IF NOT EXISTS idx_clock_entries_clock_in_time ON clock_entries(clock_in_time);
CREATE INDEX IF NOT EXISTS idx_workers_employee_id_number ON workers(employee_id_number);
```

```
CREATE INDEX IF NOT EXISTS idx_projects_project_name ON projects(project_name);

COMMENT ON COLUMN clock_entries.duration_minutes IS 'Total duration of the work shift in minutes,
calculated upon clock-out.';
COMMENT ON COLUMN clock_entries.recorded_pay_rate IS 'The hourly pay rate effective for this worker at the
moment of clock-in.';
COMMENT ON COLUMN pay_rates.effective_end_date IS 'If NULL, this pay rate is currently active until a new
one supersedes it.';

COMMIT; -- Commit all changes
```