# Technical Design

- **Architecture Overview:** The app follows a modern web architecture. A **React/Next.js** frontend runs as a Single Page App (hosted on a CDN/hosting like Vercel) providing the dashboard and forms. It communicates via HTTPS/JSON APIs to a backend service (Node.js/Express or serverless functions) that handles business logic. The backend uses a **PostgreSQL** database (hosted on AWS RDS or via Supabase) to store Contractors, Workers, Attendance, Payments, and SentReceipts.
  - *User Flow:* A contractor logs in ➔ interacts with the React UI ➔ front-end calls REST APIs ➔ backend performs DB operations ➔ (on payment, triggers sending of SMS via Twilio) ➔ returns confirmation.
  - *External Services:* Twilio (or an equivalent SMS/WhatsApp API) is used to send messages. A PDF generation service (using a Node.js library like PDFKit or a headless-browser like Puppeteer) creates receipts. Analytics (Google Analytics or Mixpanel) track usage.
  - *Diagram (conceptual):* Frontend ↔ Backend API ↔ Database; Backend ↔ Twilio/SMS; Backend ↔ PDF service.
-

  *Figure: Example of a clean admin dashboard (payments/attendance metrics).*
- **Data Models:** Key entities include:
  - **Contractor:** { id, name, email, phone, (optionally company info) }.
  - **Worker:** { id, contractorId, name, contactPhone, wageRate, bankAccount(optional) }.
  - **Attendance:** { id, contractorId, workerId, date, hoursWorked, status }.
  - **Payment:** { id, contractorId, workerId, datePaid, amount, notes }.
  - **SMSReceipt:** { id, paymentId, sentTo (phone/email), sentAt, successFlag, messageSid }.
    Each model relates by IDs (Attendance/Payment link to Worker; Worker link to Contractor). We will use foreign keys and possibly an ORM (e.g. Prisma or TypeORM) for data integrity.
- **Security Practices:** We will implement HTTPS for all traffic (SSL/TLS), encrypt sensitive fields (e.g. personal phone numbers, bank details) in the database, and never store plaintext credentials. Authentication tokens (JWT) will be signed with secure keys. Use OWASP guidelines to prevent injection and XSS. Store logs of all changes (who did what and when). Follow corpsoft's recommendations: "Data encryption (in transit and at rest), role-based access control, secure authentication (OAuth2, 2FA), and audit logs".
- **Scalability:** Deploy on a cloud platform (AWS/GCP/Azure). Use Docker containers or serverless functions for the backend. Configure CI/CD pipelines for easy updates. Separate services (frontend, API, DB) enable independent scaling.