Technical Specification: Clock-In/Clock-Out Module (React + Django)

This document outlines the architecture, database schema, API design, and logic for a production-ready attendance system.

1. System Architecture

The system relies on a **Server-Authoritative Time** model. The frontend (React) is a display layer; it never determines the "official" clock-in time. All timestamps are generated by the Django backend to prevent client-side manipulation.

High-Level Flow

1. **Clock In:** User clicks button -> React captures Location/Device -> POST to Django -> Django records `now()` -> Returns Session.
2. **Timer:** React calculates `Current Time - Server Start Time` to show the duration.
3. **Auto-Clock Out:** A background job (Celery or Cron) runs every 5-15 minutes to check for active sessions exceeding the shift limit.

2. Backend (Django + Django REST Framework)

A. Database Schema (`models.py`)

We need a robust model to store coordinates, device info, and flags for auto-logout.

```
from django.db import models
from django.conf import settings
from django.utils import timezone

class AttendanceLog(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, relate

    # Time Data
    clock_in_time = models.DateTimeField(auto_now_add=True, db_index=True)
    clock_out_time = models.DateTimeField(null=True, blank=True)
    shift_date = models.DateField(default=timezone.now, db_index=True) # Useful for que

    # Location & Device Data (Snapshot at Clock-In)
    latitude_in = models.DecimalField(max_digits=9, decimal_places=6, null=True)
    longitude_in = models.DecimalField(max_digits=9, decimal_places=6, null=True)
    device_agent = models.CharField(max_length=255, help_text="Browser/Device User Ager
    ip_address = models.GenericIPAddressField(null=True, blank=True)

    # Location Data (Snapshot at Clock-Out)
    latitude_out = models.DecimalField(max_digits=9, decimal_places=6, null=True, blank
    longitude_out = models.DecimalField(max_digits=9, decimal_places=6, null=True, blar

    # Status Flags
    is_auto_clocked_out = models.BooleanField(default=False, help_text="True if system

    class Meta:
        ordering = ['-clock_in_time']
        verbose_name = "Attendance Log"

    @property
    def duration(self):
        if self.clock_out_time:
            return self.clock_out_time - self.clock_in_time
        return timezone.now() - self.clock_in_time
```

B. API Endpoints (`views.py`)

We need three primary endpoints.

1. `GET /api/attendance/current-status/`

Returns the user's state for the current day.

- **Response:**

```
{
    "status": "CLOCKED_IN", // or "CLOCKED_OUT", "NOT_STARTED"
    "start_time": "2023-10-27T09:00:00Z",
    "elapsed_seconds": 3600,
    "shift_date": "2023-10-27"
}
```

2. `POST /api/attendance/clock-in/`

- **Payload:** `{ "latitude": 12.34, "longitude": 56.78, "device_info": "Chrome..." }`
- **Logic:**
  1. Check if user already has an active session (`clock_out_time__isnull=True`). If yes, return error.
  2. Create `AttendanceLog` .
  3. Return success with `start_time` .

3. `POST /api/attendance/clock-out/`

- **Payload:** `{ "latitude": ..., "longitude": ... }`
- **Logic:**
  1. Find latest active session for user.
  2. Update `clock_out_time = timezone.now()` .
  3. Save location data.

C. Auto Clock-Out Logic (Cron/Management Command)

Since shifts might end at different times, we run a scheduled task.

File: `management/commands/auto_clockout.py`

```
from django.core.management.base import BaseCommand
from django.utils import timezone
from myapp.models import AttendanceLog
import datetime
```

```
class Command(BaseCommand):
    help = 'Auto clock out users who forgot to sign out'

    def handle(self, *args, **options):
        # Configuration: Max shift length (e.g., 12 hours)
        MAX_SHIFT_HOURS = 12
        threshold_time = timezone.now() - datetime.timedelta(hours=MAX_SHIFT_HOURS)

        # Find active sessions older than 12 hours
        stale_sessions = AttendanceLog.objects.filter(
            clock_out_time__isnull=True,
            clock_in_time__lt=threshold_time
        )

        count = 0
        for session in stale_sessions:
            # Set clock out time to exactly the threshold limit (or shift end time if s
            session.clock_out_time = session.clock_in_time + datetime.timedelta(hours=M
            session.is_auto_clocked_out = True
            session.save()
            count += 1

        self.stdout.write(f"Auto-clocked out {count} users.")
```

- **Production Deployment:** Add a cron job to run this every 30 minutes: `*/30 * * * * python manage.py auto_clockout`

3. Frontend (React)

A. State Management & Hooks

We need a `useAttendance` hook to handle the timer logic.

- **Challenge:** JavaScript `setInterval` drifts over time and pauses when the tab is inactive.
- **Solution:** Do not increment a counter. Instead, store the `startTime` from the server. Every second, calculate: `now = new Date()` `elapsed = now - startTime`

B. Geolocation

Use the `navigator.geolocation.getCurrentPosition` API.

- **Edge Case:** If the user denies permission, you must decide business logic (Block clock-in? Or allow with a "Location Denied" flag?).

C. Security Headers

Ensure `CSRFToken` is sent with every POST request to Django.

```
axios.defaults.headers.common['X-CSRFToken'] = getCookie('csrftoken');
```

4. Production Checklist

1. **Timezones:** Ensure Django `TIME_ZONE` is set correctly (e.g., 'UTC') and convert to user's local time in the React frontend using `Intl.DateTimeFormat` or `date-fns` .
2. **HTTPS:** Geolocation API **only** works over HTTPS in modern browsers.