New Settlements Page - Replit Prompt

Generated: 2025-10-27 08:33

You are editing a full-stack TS project:

- server/: Express + Drizzle (Postgres/Neon). Routes in server/routes.ts, storage in server/storage.ts, schemas in shared/schema.ts.
- client/: React + TS + Vite + shadcn/ui. Router in client/src/App.tsx.
- API helper: client/src/lib/api.ts. Date helpers: client/src/lib/dates.ts.
- Timezone: Asia/Kolkata (IST). **All date math is inclusive** and week windows are **Monday → Sunday**.

==
TASK: Replace old "Settlements" with a NEW **Settlements** page
==
Purpose: Calculate **weekly profit** for every week that exists in Trip Logs.
Each row represents **one week (Mon–Sun)** and shows:

Columns (left → right):
1) **Week (Mon–Sun)** — non-editable label (e.g., 13/10/2025 – 19/10/2025)
2) **Rent** — sum of Trip Log `rent` within the week — **computed, non-editable**
3) **Wallet** — sum of **Weekly Summary wallets** within the same week — **computed, non-editable**
4) **Company Rent** — **editable** integer input (₹)
5) **Company Wallet** — **editable** integer input (₹)
6) **Room Rent** — fixed **₹4,666** — **non-editable**
7) **Profit** — computed only when BOTH Company Rent and Company Wallet are filled:
   Profit = **Rent − Wallet − Company Rent + Company Wallet − Room Rent**
8) **Actions** — Save (upsert company fields) and Delete (clear company fields for that week)

Other rules:
- Weeks are auto-generated from Trip Logs: from the **earliest Monday** that covers `MIN(trip_date)` to the **latest Sunday** that covers `MAX(trip_date)`.
- Only Monday–Sunday windows (inclusive) are shown.
- Use INR formatting (no decimals) in UI; store as integers in DB.
- Use inclusive date filtering with `DATE(col) BETWEEN start::date AND end::date` to avoid tz bugs.

==
DB: New table to store company fields per week
==
Update **shared/schema.ts** to add `weekly_settlements` to persist just the editable values (per week). Profit is always computed at runtime.

```ts
// shared/schema.ts
import { pgTable, date, integer, timestamp, primaryKey } from "drizzle-orm/pg-core";
import { z } from "zod";

export const weeklySettlements = pgTable("weekly_settlements", {
  weekStart: date("week_start").notNull(), // Monday (inclusive)
  weekEnd: date("week_end").notNull(),     // Sunday (inclusive)
  companyRent: integer("company_rent"),    // nullable => "not filled"
  companyWallet: integer("company_wallet"),
  createdAt: timestamp("created_at").notNull().defaultNow(),
  updatedAt: timestamp("updated_at").notNull().defaultNow(),
}, (t) => ({
  pk: primaryKey({ columns: [t.weekStart, t.weekEnd] }),
}));
```

New Settlements Page - Replit Prompt

Generated: 2025-10-27 08:33

````
export const upsertWeeklySettlementSchema = z.object({
  weekStart: z.coerce.date(),
  weekEnd: z.coerce.date(),
  companyRent: z.number().int().min(0).nullable().optional(),
  companyWallet: z.number().int().min(0).nullable().optional(),
});
export type UpsertWeeklySettlementInput = z.infer<typeof upsertWeeklySettlementSchema>;
```

Create migration **drizzle/00xx_create_weekly_settlements.sql**:

```sql
CREATE TABLE IF NOT EXISTS weekly_settlements (
  week_start date NOT NULL,
  week_end date NOT NULL,
  company_rent integer NULL,
  company_wallet integer NULL,
  created_at timestamp NOT NULL DEFAULT now(),
  updated_at timestamp NOT NULL DEFAULT now(),
  PRIMARY KEY (week_start, week_end)
);
```

Run migrations: `npm run db:push`

============================================================
SERVER: Weekly ranges + aggregation + upsert + delete
============================================================
In **server/storage.ts** implement helpers. Assumptions:
- Trip logs table = `driver_rent_logs` with (trip_date, rent).
- Weekly Summary table = `weekly_summaries` with (start_date, end_date, total_earnings, cash, refund, expenses).
  Wallet per row = total_earnings − cash + refund − expenses − 100.
  For a given week, sum wallets across **weekly_summaries** rows where start_date==weekStart and end_date==weekEnd.

```ts
// server/storage.ts
import { sql } from "drizzle-orm";
import { driverRentLogs, weeklySummaries, weeklySettlements } from "../shared/schema"; // adjust paths

// 1) Get min/max trip_date
export async function getTripDateBounds(db): Promise<{min:string|null; max:string|null}> {
  const out = await db.execute(sql`SELECT MIN(trip_date)::date AS min_date, MAX(trip_date)::date AS max_date FROM driver_rent_logs`);
  const row = Array.isArray(out) ? out[0] : out.rows?.[0];
  return { min: row?.min_date ? String(row.min_date) : null, max: row?.max_date ? String(row.max_date)
 : null };
}

// 2) Build all Monday→Sunday ranges that cover min..max (inclusive)
export async function listWeeklyWindows(db): Promise<Array<{weekStart:string; weekEnd:string}>> {
  const { min, max } = await getTripDateBounds(db);
````

New Settlements Page - Replit Prompt

Generated: 2025-10-27 08:33

```
  if (!min || !max) return [];
  const first = await db.execute(sql`
    SELECT
      ( min::date - (EXTRACT(ISODOW FROM {min}::date)::int - 1) )::date AS week_start,
      ( max::date - (EXTRACT(ISODOW FROM {max}::date)::int - 1) + 6 )::date AS week_end_all
  `);
  const row = Array.isArray(first) ? first[0] : first.rows?.[0];
  const ws = String(row.week_start);
  const weAll = String(row.week_end_all);

  const seq = await db.execute(sql`
    WITH weeks AS (
      SELECT generate_series(ws::date,{weAll}::date, interval '7 days')::date AS week_start
    )
    SELECT week_start, (week_start + 6)::date AS week_end FROM weeks ORDER BY week_start ASC
  `);
  const rows = Array.isArray(seq) ? seq : seq.rows;
  return rows.map((r:any) => ({ weekStart: String(r.week_start), weekEnd: String(r.week_end) }));
}

// 3) Aggregate one week
export async function aggregateWeek(db, weekStart: string, weekEnd: string) {
  const rentRow = await db.execute(sql`
    SELECT COALESCE(SUM(rent),0)::int AS rent_sum
    FROM driver_rent_logs
    WHERE DATE(trip_date) BETWEEN weekStart::dateAND{weekEnd}::date
  `);
  const rent = Number((Array.isArray(rentRow)?rentRow[0]:rentRow.rows[0]).rent_sum) || 0;

  const wsRows = await db.execute(sql`
    SELECT COALESCE(SUM(total_earnings - cash + refund - expenses - 100), 0)::int AS wallet_sum
    FROM weekly_summaries
    WHERE start_date = weekStart::dateANDend_date = {weekEnd}::date
  `);
  const wallet = Number((Array.isArray(wsRows)?wsRows[0]:wsRows.rows[0]).wallet_sum) || 0;

  const comp = await db.execute(sql`
    SELECT company_rent, company_wallet
    FROM weekly_settlements
    WHERE week_start = weekStart::dateANDweek_end = {weekEnd}::date
  `);
  let companyRent: number|null = null, companyWallet: number|null = null;
  const crow = Array.isArray(comp)?comp[0]:comp.rows?.[0];
  if (crow) {
    companyRent = crow.company_rent === null ? null : Number(crow.company_rent);
    companyWallet = crow.company_wallet === null ? null : Number(crow.company_wallet);
  }
  const ROOM_RENT = 4666;
  const canCalc = companyRent !== null && companyWallet !== null;
  const profit = canCalc ? (rent - wallet - (companyRent||0) + (companyWallet||0) - ROOM_RENT) : null

  return { weekStart, weekEnd, rent, wallet, companyRent, companyWallet, roomRent: ROOM_RENT, profit
};
}
```

New Settlements Page - Replit Prompt

Generated: 2025-10-27 08:33

```
// 4) List all
export async function listWeeklySettlements(db) {
  const weeks = await listWeeklyWindows(db);
  const rows = [];
  for (const w of weeks) rows.push(await aggregateWeek(db, w.weekStart, w.weekEnd));
  return rows;
}

// 5) Upsert/Delete
export async function upsertWeeklySettlement(db, input: { weekStart:string; weekEnd:string;
companyRent:number|null; companyWallet:number|null; }) {
  const now = new Date();
  await db.insert(weeklySettlements)
    .values({ weekStart: input.weekStart, weekEnd: input.weekEnd, companyRent: input.companyRent,
companyWallet: input.companyWallet, createdAt: now, updatedAt: now })
    .onConflictDoUpdate({
      target: [weeklySettlements.weekStart, weeklySettlements.weekEnd],
      set: { companyRent: input.companyRent, companyWallet: input.companyWallet, updatedAt: now },
    });
}

export async function deleteWeeklySettlement(db, weekStart:string, weekEnd:string) {
  await db.execute(sql`DELETE FROM weekly_settlements WHERE week_start=${weekStart}::date AND
week_end=${weekEnd}::date`);
}
```

Add routes in **server/routes.ts**:

```ts
import { upsertWeeklySettlementSchema } from "../shared/schema";
import { listWeeklySettlements, upsertWeeklySettlement, deleteWeeklySettlement } from "./storage";

app.get("/api/settlements", async (req, res) => {
  const items = await listWeeklySettlements(req.db);
  res.json({ items });
});

app.post("/api/settlements", async (req, res) => {
  const parsed = upsertWeeklySettlementSchema.safeParse(req.body);
  if (!parsed.success) return res.status(400).json({ error: parsed.error.flatten() });
  const payload = {
    ...parsed.data,
    weekStart: parsed.data.weekStart.toISOString().slice(0,10),
    weekEnd: parsed.data.weekEnd.toISOString().slice(0,10),
  };
  await upsertWeeklySettlement(req.db, payload);
  const items = await listWeeklySettlements(req.db);
  res.json({ ok: true, items });
});

app.delete("/api/settlements", async (req, res) => {
  const { weekStart, weekEnd } = req.query as { weekStart?:string; weekEnd?:string };
```

New Settlements Page - Replit Prompt

Generated: 2025-10-27 08:33

```
  if (!weekStart || !weekEnd) return res.status(400).json({ error: "weekStart and weekEnd required"
});
  await deleteWeeklySettlement(req.db, weekStart, weekEnd);
  const items = await listWeeklySettlements(req.db);
  res.json({ ok: true, items });
});
```

```
============================================================
CLIENT: API & Page
============================================================
**client/src/lib/api.ts**

```ts
export type SettlementRow = {
  weekStart: string;
  weekEnd: string;
  rent: number;
  wallet: number;
  companyRent: number | null;
  companyWallet: number | null;
  roomRent: number;
  profit: number | null;
};

export async function getSettlements() {
  const r = await fetch("/api/settlements");
  if (!r.ok) throw new Error("Failed to load settlements");
  return (await r.json()) as { items: SettlementRow[] };
}

export async function saveSettlement(p: { weekStart:string; weekEnd:string; companyRent:number|null;
companyWallet:number|null; }) {
  const r = await fetch("/api/settlements", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(p),
  });
  if (!r.ok) throw new Error("Failed to save settlement");
  return (await r.json()) as { ok: true; items: SettlementRow[] };
}

export async function deleteSettlement(weekStart:string, weekEnd:string) {
  const r = await fetch(`/api/settlements?weekStart=weekStart&weekEnd={weekEnd}`, { method:
"DELETE" });
  if (!r.ok) throw new Error("Failed to delete settlement");
  return (await r.json()) as { ok: true; items: SettlementRow[] };
}
```

client/src/pages/settlements.tsx (replace old page)

```tsx
import { useQuery, useMutation, useQueryClient } from "@tanstack/react-query";
```
```

New Settlements Page - Replit Prompt

Generated: 2025-10-27 08:33

```
import { getSettlements, saveSettlement, deleteSettlement } from "../lib/api";
import { useState } from "react";
import { Card } from "@/components/ui/card";
import { Input } from "@/components/ui/input";
import { Button } from "@/components/ui/button";
import { Table, TableHead, TableHeader, TableRow, TableBody, TableCell } from "@/components/ui/table";

const inr = (n:number)=> new Intl.NumberFormat("en-
IN",{style:"currency",currency:"INR",maximumFractionDigits:0}).format(n||0);
const fmt = (iso:string)=> new Date(iso).toLocaleDateString("en-GB");

export default function SettlementsPage() {
  const qc = useQueryClient();
  const { data, isLoading, error } = useQuery({ queryKey:["settlements"], queryFn: getSettlements });
  const rows = data?.items ?? [];

  const mSave = useMutation({
    mutationFn: saveSettlement,
    onSuccess: () => qc.invalidateQueries({ queryKey:["settlements"] }),
  });
  const mDel = useMutation({
    mutationFn: ({weekStart, weekEnd}:{weekStart:string; weekEnd:string}) =>
deleteSettlement(weekStart, weekEnd),
    onSuccess: () => qc.invalidateQueries({ queryKey:["settlements"] }),
  });

  return (
    <div className="space-y-4">
      <h1 className="text-2xl font-bold">Settlements</h1>

      <Card className="p-0 overflow-hidden">
        <Table>
          <TableHead>
            <TableRow>
              <TableHeader>Week</TableHeader>
              <TableHeader className="text-right">Rent</TableHeader>
              <TableHeader className="text-right">Wallet</TableHeader>
              <TableHeader className="text-right">Company Rent</TableHeader>
              <TableHeader className="text-right">Company Wallet</TableHeader>
              <TableHeader className="text-right">Room Rent</TableHeader>
              <TableHeader className="text-right">Profit</TableHeader>
              <TableHeader className="text-right">Actions</TableHeader>
            </TableRow>
          </TableHead>
          <TableBody>
            {isLoading && <TableRow><TableCell colSpan={8}>Loading…</TableCell></TableRow>}
            {error && <TableRow><TableCell colSpan={8}>Failed to load.</TableCell></TableRow>}
            {!isLoading && rows.length===0 && <TableRow><TableCell colSpan={8}>No
data.</TableCell></TableRow>}
            {rows.map(r => {
              const [cr, setCr] = useState<string>(r.companyRent!=null? String(r.companyRent):"");
              const [cw, setCw] = useState<string>(r.companyWallet!=null? String(r.companyWallet):"");
              const canCalc = cr !== "" && cw !== "";
              return (
```

New Settlements Page - Replit Prompt

Generated: 2025-10-27 08:33

```
                <TableRow key={`r.weekStart—{r.weekEnd}`}>
                  <TableCell>{fmt(r.weekStart)} — {fmt(r.weekEnd)}</TableCell>
                  <TableCell className="text-right">{inr(r.rent)}</TableCell>
                  <TableCell className="text-right">{inr(r.wallet)}</TableCell>
                  <TableCell className="text-right">
                    <Input className="text-right" type="number" min={0} step={1} value={cr}
onChange={e=>setCr(e.target.value)} />
                  </TableCell>
                  <TableCell className="text-right">
                    <Input className="text-right" type="number" min={0} step={1} value={cw}
onChange={e=>setCw(e.target.value)} />
                  </TableCell>
                  <TableCell className="text-right">{inr(r.roomRent)}</TableCell>
                  <TableCell className="text-right">
                    {canCalc ? inr((r.rent - r.wallet - Number(cr||0) + Number(cw||0) - r.roomRent))
 "—"}
                  </TableCell>
                  <TableCell className="text-right space-x-2">
                    <Button size="sm" onClick={()=>mSave.mutate({ weekStart:r.weekStart,
weekEnd:r.weekEnd, companyRent: cr===""?null:Number(cr), companyWallet: cw===""?null:Number(cw)
})}>Save</Button>
                    <Button size="sm" variant="destructive" onClick={()=>mDel.mutate({
weekStart:r.weekStart, weekEnd:r.weekEnd })}>Delete</Button>
                  </TableCell>
                </TableRow>
              );
            })}
          </TableBody>
        </Table>
      </Card>
    </div>
  );
}
```

```
============================================================
ROUTER & NAV
============================================================
- Ensure `/settlements` route imports the new page and update nav label.
- Remove any old Settlements implementation.

============================================================
ACCEPTANCE TESTS
============================================================
- Weeks cover Monday of MIN(trip_date) → Sunday of MAX(trip_date) inclusively.
- Rent equals sum of `driver_rent_logs.rent` within each Mon–Sun window.
- Wallet equals sum over `weekly_summaries` rows for that week of (TE − Cash + Refund − Expenses −
100).
- Company fields persist per week; Delete clears them (weeks still appear).
- Room Rent is fixed at ₹4,666.
- Profit only appears when both company fields are provided; matches: Rent − Wallet − Company Rent +
Company Wallet − 4,666.
- UI shows INR with no decimals. All server filters use inclusive `DATE(...) BETWEEN ...`.
```