Client Profitability Dashboard — Emergent Blueprint

One-line: A PWA-first, light-weight client profitability app for freelancers and small agencies — clear KPIs, fast drillthrough, and themeable gradient UI (Blue primary + 3 gradient presets) to plug into your Freelancer Growth & Income Kit.

---

1. Purpose & Target

- **Purpose:** Help freelancers and small agencies know which clients and projects make money, identify late-paying clients, and prioritise time toward high-margin work.
- **Primary users:** Solo freelancers, small agencies (1–10 people), finance-savvy solopreneurs.

---

2. High-level Feature Set

- Overview KPIs: Total Revenue, Gross Profit, Gross Margin %, Outstanding AR, DSO, Billable Hours
- Client ranking & segmentation (Tier, Profitability buckets)
- AR aging buckets and automated alerts (30/60/90+)
- Drillthrough to invoices & projects per client
- Search & advanced slicers (date range, region, service type, client tier)
- Bookmarks / saved views & shareable links
- CSV import & mapping wizard for quick onboarding
- Exports: PDF report, CSV, PPTX slide (export view)
- PWA features: offline read/write with sync queue, push notifications for overdue AR
- Theme system: primary Blue gradient plus three presets (Green, Purple, Teal), charts aligned to theme

---

3. Recommended Tech Stack (opinionated)

Frontend: React + Vite, Tailwind CSS (utility-first, fast), Headless UI for accessible components, Recharts or Chart.js (or optionally `@nivo` for rich visualisations). Use TypeScript.

PWA tooling: Workbox for service worker + caching strategies, idb-keyval or Dexie for IndexedDB convenience.

Backend: Node.js + Fastify or Express with TypeScript. Use Prisma ORM for a Postgres database.

DB: PostgreSQL (hosted: Render/Heroku/AWS RDS) + Redis for caching.

Auth: Auth0 or NextAuth / JWT for custom. Support single-user (freelancer) and multi-user agencies later.

Hosting / Deployment: Vercel/Netlify for frontend, Render/AWS EC2 or DigitalOcean App Platform for API and DB. Use CI/CD with GitHub Actions.

Reporting / Export: Puppeteer (server-side PDF), pptxgenjs (PPTX), csv-writer for CSV exports.

---

4. Data Model (core tables)

Below are the essential tables and field types. Use normalized schema with indexes on commonly queried fields.

`users`

- `id` UUID PK
- `email` varchar
- `password_hash` varchar (if not using external auth)
- `name` varchar
- `created_at` timestamptz
- `settings` jsonb (theme, bookmarks)

`clients`

- `id` UUID PK
- `client_code` varchar (short id)
- `name` varchar
- `tier` varchar (Enterprise/SMB/Freelance)
- `region` varchar
- `currency` varchar
- `contact_info` jsonb
- `created_at`

`projects`

- `id` UUID
- `client_id` FK -> clients.id
- `name` varchar
- `service_type` varchar (Consulting, Design, Dev, Support)

`invoices`

- `id` UUID
- `invoice_number` varchar
- `client_id` FK
- `project_id` FK
- `invoice_date` date

- `due_date` date
- `invoice_amount` numeric
- `paid_amount` numeric
- `status` varchar (Paid/Partially Paid/Unpaid)
- `payment_date` date
- `hours_billed` numeric
- `hourly_rate` numeric
- `direct_cost` numeric
- `overhead_allocation` numeric
- `currency` varchar
- `created_at`

`payments`

- `id` UUID
- `invoice_id` FK
- `amount` numeric
- `payment_date` date
- `method` varchar

`bookmarks` **(saved views)**

- `id` UUID
- `user_id` FK
- `name` varchar
- `filters` jsonb
- `public` boolean
- `created_at`

---

5. Sample API (REST) endpoints

Secure all endpoints (TLS + JWT). Use pagination for list endpoints.

Auth - `POST /api/v1/auth/login` -> {email, password} => {token} - `POST /api/v1/auth/signup`

Clients - `GET /api/v1/clients` (filters: q, tier, region) - `GET /api/v1/clients/:id` (detailed) - `POST /api/v1/clients` (create)

Invoices - `GET /api/v1/invoices` (filters: client_id, status, date_from, date_to, page, pageSize) - `POST /api/v1/invoices/import` (CSV upload & mapping) - `GET /api/v1/invoices/:id`

Reports - `GET /api/v1/reports/overview?from=YYYY-MM-DD&to=YYYY-MM-DD` -> aggregated KPIs - `GET /api/v1/reports/client/:id` -> client metrics & invoice list - `GET /api/v1/reports/aging` -> AR aging buckets

Bookmarks - `POST /api/v1/bookmarks` -> store filter set - `GET /api/v1/bookmarks` -> list

Export - `POST /api/v1/exports/pdf` -> {viewId or filters} -> jobId & result URL - `GET /api/v1/exports/:jobId` -> status & download

---

6. Core Calculations & SQL / DAX Snippets

Use server-side aggregation for summary endpoints. Example SQL (Postgres) for core KPIs:

```sql
-- Total Revenue in date range
SELECT SUM(invoice_amount) AS total_revenue
FROM invoices
WHERE invoice_date BETWEEN $1 AND $2;

-- Outstanding AR
SELECT SUM(invoice_amount - COALESCE(paid_amount,0)) AS outstanding
FROM invoices
WHERE (invoice_amount - COALESCE(paid_amount,0)) > 0;

-- Gross Profit
SELECT SUM(invoice_amount - direct_cost - overhead_allocation) AS gross_profit
FROM invoices
WHERE invoice_date BETWEEN $1 AND $2;

-- Top 5 clients by revenue
SELECT c.id, c.name, SUM(i.invoice_amount) AS revenue
FROM clients c
JOIN invoices i ON i.client_id = c.id
GROUP BY c.id, c.name
ORDER BY revenue DESC
LIMIT 5;
```

DAX (Power BI) equivalents — copy-ready for PBIX users:

```
Total Revenue = SUM(Invoices[invoice_amount])
Total Paid = SUM(Invoices[paid_amount])
Outstanding AR = SUMX(FILTER(Invoices, Invoices[invoice_amount] -
COALESCE(Invoices[paid_amount],0) > 0), Invoices[invoice_amount] -
COALESCE(Invoices[paid_amount],0))
Total Cost = SUM(Invoices[direct_cost]) + SUM(Invoices[overhead_allocation])
Gross Profit = [Total Revenue] - [Total Cost]
Gross Margin % = DIVIDE([Gross Profit], [Total Revenue], 0)
Billable Hours = SUM(Invoices[hours_billed])
Profit per Hour = DIVIDE([Gross Profit], [Billable Hours], 0)
```

```
Realization Rate = DIVIDE([Total Revenue], SUMX(Invoices, Invoices[hours_billed]
* Invoices[hourly_rate]), 0)
```

DSO (Days Sales Outstanding) (SQL & DAX variant available on request) — compute weighted by outstanding.

---

7. Frontend Blueprint (Screens & Components)

Each screen lists visuals, fields, interactions and the expected behavior.

Global UI elements (present on every screen)

- Top nav: App logo/name, Global search bar (search clients/invoices/projects), Theme selector (preset swatches), User menu
- Left rail (collapsible): Navigation tabs (Overview, Clients, Invoices, Projects, Settings)
- Right rail (optional): Quick filters & active bookmarks
- Floating action button: `+ New Invoice` / `Import CSV`

Home / Overview (Dashboard)

Layout: 3 rows 1. KPI strip (cards): Total Revenue, Gross Profit, Gross Margin, Outstanding AR, DSO, Billable Hours - Each card: big number, small sparkline (last 12 months), delta vs previous period - Click KPI => drill to filtered view 2. Charts row: - Bar: Revenue vs Hours per Client (top 10) — bars for revenue, line for hours - Line: Revenue & Gross Profit by Month (dual axis optional) - Treemap: Revenue share by Client (interactive, click -> client page) 3. Insights & Alerts: - Overdue clients list (auto-generated) with `Send reminder` quick action - Top 5 profitable clients card + % of total revenue

Interactions: hover tooltips, click to filter, right-click for context menu (e.g., export client data)

Clients Page

- Table / Matrix: ClientName | Tier | Region | Revenue (YTD) | Profit | Margin % | Profit/Hour | Outstanding AR | Last Invoice Date
- Row actions: View client, Export invoices, Send invoice reminder
- Filters: Tier, Region, Client Tier, Service Type, Date range

Invoices & AR Page

- AR Aging chart (stacked column: 0–30, 31–60, 61–90, 90+ by client or total)
- Unpaid Invoices list with pagination & bulk actions (Send reminder, Mark paid)
- Import CSV wizard modal (column mapping + preview + import)

Client Drillthrough (detailed client view)

- Header: Client profile card (see section below)

- Charts: Revenue & Profit trend for client, Project breakdown (profit by project), AR Aging for that client
- Invoice table: all invoices with status & quick actions

Settings / Assumptions

- Margin thresholds: low / medium / high default (10% / 20% / 40%)
- Overhead allocation methods: fixed value per invoice or % of revenue
- Currency / locale settings
- Theme presets & custom color selector (advanced)

---

8. UI Components & Interactions (detailed)

- **Global Search:** incremental search (debounced 300ms) hitting `/api/v1/search?q=` returns clients, invoices, projects. Keyboard shortcut `/` to focus.
- **Slicers:** Date range (preset & custom), ServiceType multi-select, Tier select. Slicers persist in bookmarks.
- **Bookmarks / custom views:** `POST /api/v1/bookmarks` stores `{name, filters, page, layout}`. Bookmarks visible in right rail; allow `share` toggles (generate signed short URL with token expiring in X days).
- **Alerts:** Implemented two ways:
- Visual: conditional formatting (rows or KPI cards color) driven by measures.
- Actionable: alert center shows list; pressing an alert may trigger email/SMS (via integrated providers) or mark as acknowledged.
- **Conditional formatting rules:** theme-aware color tokens (see Theming section). Example: if `Margin% < threshold` => use danger color; use `fieldvalue` to dynamically color chart shapes or table backgrounds.

---

9. Theming System (exact tokens + presets)

Use CSS variables as single source of truth. Chart libraries should accept computed palette arrays passed at runtime.

CSS variables structure (base)

```
:root {
  --bg: #ffffff;
  --surface: #f7fafc;
  --text-primary: #0f172a;
  --muted: #64748b;

  /* Accent tokens — replaced per theme */
  --accent-1: #3b82f6; /* primary */
  --accent-2: #60a5fa; /* secondary */
```

```
  --accent-3: #1e40af; /* deep */

  /* Gradient */
  --gradient: linear-gradient(90deg, var(--accent-2), var(--accent-3));

  /* Semantic colors (computed from accent) */
  --good: #16a34a;
  --warn: #f59e0b;
  --bad: #ef4444;
}
```

Preset palettes (examples)

Primary — Blue Gradient - `--accent-1: #60A5FA` (light blue) - `--accent-2: #3B82F6` (blue) - `--accent-3: #1E3A8A` (dark blue) - gradient: `linear-gradient(90deg,#60A5FA 0%,#1E3A8A 100%)`

Green Gradient - `--accent-1: #86EFAC` (mint) - `--accent-2: #34D399` (green) - `--accent-3: #047857` (dark green) - gradient: `linear-gradient(90deg,#86EFAC 0%,#047857 100%)`

Purple Gradient - `--accent-1: #E9D5FF` (lavender) - `--accent-2: #A78BFA` (purple) - `--accent-3: #6D28D9` (deep purple) - gradient: `linear-gradient(90deg,#E9D5FF 0%,#6D28D9 100%)`

Teal Gradient (4th preset) - `--accent-1: #99F6E4` - `--accent-2: #2DD4BF` - `--accent-3: #0F766E` - gradient: `linear-gradient(90deg,#99F6E4 0%,#0F766E 100%)`

> Note: Keep `--bad` as a fixed red (`#ef4444`) across themes for instant recognisability of problems.

Chart color rules

- **Primary series** (bars, main lines) => `--accent-2`
- **Secondary/compare series** => `--accent-1` (lighter)
- **Highlights / deep accents** => `--accent-3`
- **Positive/negative diverging** (margins) => use semantic `--good` / `--bad` + theme-shaded versions (derive by blending). For neutral/amber use `--warn`.

Provide palettes to chart library as arrays e.g., for Chart.js:

```
const chartPalette = [getVar('--accent-2'), getVar('--accent-1'), getVar('--accent-3'), getVar('--warn'), getVar('--good')];
```

Applying theme dynamically

- Persist user choice in `users.settings.theme`.

- At app load, inject root CSS variables (style tag) or toggle classes `theme-blue`, `theme-green` that set variables.
- Re-initialize charts with new palette on theme change (chart.update()).

---

10. Conditional Formatting & Alerts (implementation specifics)

Server-side: compute client-level `margin_pct` and `outstanding_amount` via SQL aggregates. Provide `margin_flag` values (good/okay/poor) in API.

Client-side: apply style classes based on `margin_flag`. Example rule: - `margin_pct < 0.10` => `.flag-bad` (use `--bad` border / background) - `0.10 <= margin_pct < 0.20` => `.flag-warn` - `>= 0.20` => `.flag-good`

AR alerts: server periodically (or on-demand) returns list of alerts: `{invoice_id, client_id, days_overdue, amount, severity}`. UI shows badge counts and an alert center with quick actions.

---

11. Bookmarks & Custom Views

Schema: store `{userId, name, filters: {dateRange, tiers, serviceTypes}, layoutState, pinned}`

Capabilities: - Save current UI state and filters as a bookmark. - Mark bookmark `public` to produce a short-lived share link `https://app/.../view/<token>`. - Allow export of bookmark as JSON for teammates.

Implementation notes: - Save on backend (POST /bookmarks) and in localStorage for instant restore when offline. - On load, hydrate filters and request new report data.

---

12. Mobile Responsive Behavior

Principles: vertical stack, simplified charts, large tap targets, hide complex visuals.

Desktop: 3-column layout on wide screens (kpis + charts + insights) **Tablet:** 2-column layout **Mobile (<480px):** single column — KPI cards become collapsible accordions

Examples: - Bar chart on desktop => show top 10 clients; mobile => show top 5 and a `See more` button. - Treemap => replaced by simple donut + list for mobile. - Floating bottom nav with 4 icons: Overview, Clients, Invoices, More

Accessibility: ensure touch targets >=44px, font scaling with `rem`, and keyboard nav for modals.

---

13. PWA & Offline Strategy

Manifest & Service-worker: - Use `manifest.json` with icons, theme_color pointing to `--accent-2`, start_url. - Service worker (Workbox): precache shell assets; runtime cache API responses with `StaleWhileRevalidate` for lists; `NetworkFirst` for submit endpoints.

Offline data: - Store user data & imported CSV in IndexedDB (Dexie). Provide queue for write operations (create invoice / mark paid) while offline. - On reconnect, run `syncQueue` to flush operations to API; implement optimistic UI but show sync status per item.

Conflict resolution: - If server state changed, show merge modal (server v local) and allow user to choose or merge fields.

Push notifications: - Use Push API (opt-in) to send reminders for overdue invoices or completed imports.

---

14. Export & Sharing

Exports supported: PDF (full dashboard or filtered view), CSV (invoices / clients), PPTX (single-slide summary), shareable view link.

Implementation: - PDF: Server-side Puppeteer job that renders the dashboard HTML (snapshot mode) with auth token. - CSV: On demand from backend -> stream CSV. - PPTX: pptxgenjs client-side for simple single-page export, or server-side generation for styled slides. - Share link: create short-lived token stored in DB mapping to bookmark filters; viewers can open read-only view.

---

15. Security, Privacy & Compliance

- TLS (HTTPS) everywhere
- Hash passwords (bcrypt) or use third-party auth
- Encrypt sensitive fields at rest if needed
- Audit logs for exports
- Allow account-level data export & deletion (GDPR)
- Rate-limit export endpoints to prevent misuse

---

16. Performance & Scaling Notes

- Use server-side aggregation for heavy KPIs; expose cached endpoints for overview (`/reports/overview`) with TTL (e.g., 1–5 minutes).
- Redis cache for expensive queries (top clients / aging buckets)
- Pagination for lists; infinite scroll optional
- Use CDN for static assets

---

17. Tests & QA

- Unit tests for calculations (SQL/DAX equivalence)
- Integration tests for imports & CSV mapping
- E2E tests (Cypress) for user flows (import -> dashboard -> export)
- Accessibility testing (axe-core)

---

18. Handoff Checklist for Emergent

Provide these assets to the dev team: 1. This blueprint (current doc) 2. Sample CSV dataset (300 rows) with realistic invoices, clients, payments 3. Figma mockups for key screens (Overview, Client detail, Import wizard) 4. OpenAPI spec for endpoints 5. Storybook for components (buttons, cards, table rows) 6. Test cases for import mapping and KPI checks 7. Theme token file (JSON) with color hexes above

---

19. Next Steps (recommended)

1. Create Figma mockups for the three highest-value screens.
2. Provide sample dataset and run a quick prototype build for the Overview page (React + Tailwind + Recharts).
3. Implement CSV import wizard and server import route first — quick win for users.

If you want, I can:
- generate the **Figma-ready spec** (exportable tokens + spacing system) next,
- OR scaffold a **React + Tailwind** prototype for the Overview screen (single file) that Emergent can extend.

Which of those should I produce now?