








# Architecture Overview

- **Summary:** PomoTo-do is a Next.js App Router front-end delivering an integrated Pomodoro timer, task and project management, and marketing surfaces for a productivity SaaS. Current implementation renders all views client-side with mock data, local storage for preferences, and no persistent backend.
- **Scope:** This document describes the present web application architecture (/app, /components, /hooks, /lib, /styles, /public) and highlights gaps against the PRD in docs/PRD.md. Infrastructure, authentication, payments, and integrations are not yet implemented.
- **Context:** Core user journeys per the PRD  plan the day, run focus sessions, review productivity, personalize settings, and evaluate upgrades  are prototyped fully in the UI. Collaboration, real data persistence, analytics pipelines, and billing are identified as roadmap items.

## Requirements Component Mapping

PRD requirement	Component/Module	Related file/path	Validation method/test
Adjustable Pomodoro cycles with focus mode and quick settings	Pomodoro Timer UI and settings context	components/pomodoro-timer.tsx, components/pomodoro-settings-provider.tsx	Manual timer interactions; inspect state transitions and localStorage writes
Task CRUD with filters, tags, and Pomodoro estimates	Task management workspace	components/task-management.tsx, page wrapper  pp/tasks/page.tsx	Manual UI flow; verify state updates in React component
Project overview with progress, filters, and dialogs	Projects workspace	components/projects-content.tsx,  pp/projects/page.tsx	Manual UI smoke test; confirm mock data rendering
Productivity dashboard with heatmap, streaks, and pro gating	Dashboard content and pro dashboard wrapper	components/dashboard-content.tsx, components/pro-dashboard-content.tsx,  pp/page.tsx	Manual navigation; confirm dev mode toggle gates tabs
Settings for Pomodoro defaults, notifications, themes, profile	Settings experience	components/settings-content.tsx, components/theme-provider.tsx,  pp/settings/page.tsx	Manual settings change; validate localStorage persistence
Upgrade prompts and pricing comparison	Upgrade modal and sidebar CTA	components/upgrade-modal.tsx, components/sidebar.tsx	Trigger modal from CTA; review pro feature list
Marketing landing page with pricing and CTAs	Landing page	components/landing-page.tsx,  pp/landing/page.tsx	Visual inspection; confirm sections align with PRD copy

PRD requirement	Component/Module	Related file/path	Validation method/test
Developer mode unlock for pro features	Dev mode context and toggle	components/theme-provider.tsx, components/sidebar.tsx, components/dashboard-content.tsx	Toggle dev mode; verify pro tabs enabled
Persisted data layer for tasks, projects, sessions	Missing backend/API	❖	Gap❖requires backend service & database
Authentication and authorization for user accounts	Unimplemented	❖	Gap❖needs auth provider, guarded routes
Billing integration for trials and subscriptions	Unimplemented	❖	Gap❖Stripe/Paddle integration and entitlement checks
Analytics telemetry beyond Vercel basics	Minimal analytics	❑pp/layout.tsx (@vercel/analytics/next)	Confirm analytics component inclusion; note lack of custom events

High-Level Architecture

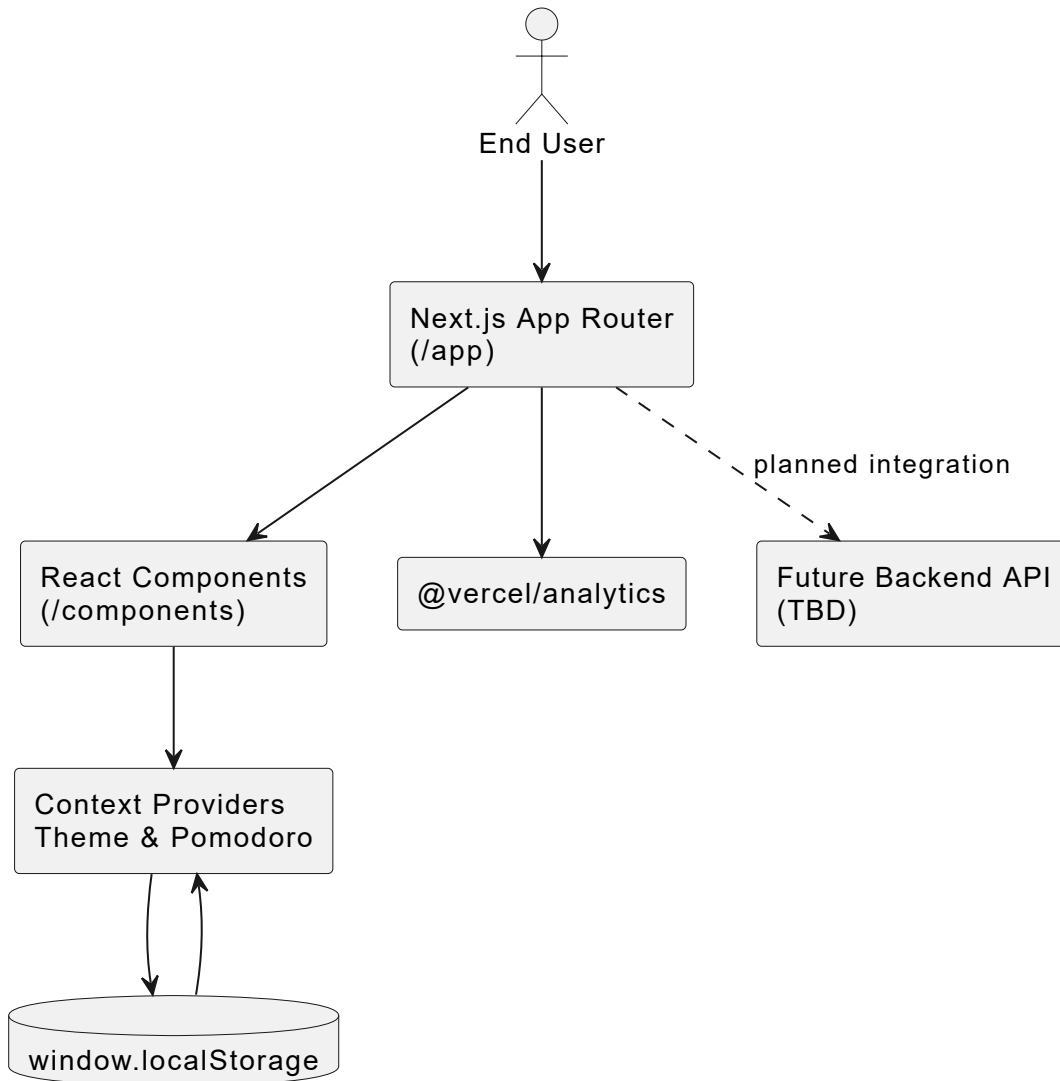
```
`mermaid flowchart LR
  subgraph Browser_UI [Next.js App Router UI (/app)]
    LocalStorage[(Window.localStorage)]
  end

  subgraph Frontend_Runtime [React Client Components /components/**]
    Providers[Context Providers Theme & Pomodoro Settings]
  end

  subgraph External_Services [Analytics @vercel/analytics Managed by Vercel]
    FutureAPI[Future Backend API Tasks/Projects/Auth]
  end

  UI --> Components
  Components --> Providers
  Providers --> LocalStorage
  Providers <-- LocalStorage
  UI --> Analytics_UI[Analytics UI -.planned.-> FutureAPI]
  classDef planned stroke-dasharray: 5 5,stroke:#999,color:#666;
  class FutureAPI planned;

`plantuml
```



## Components and Responsibilities

- **Routing shell (`app/layout.tsx`):** Applies global fonts, wraps pages with `ThemeProvider` and `PomodoroSettingsProvider`, injects Vercel Analytics, and loads global Tailwind styles.
- **Navigation (`components/sidebar.tsx`):** Renders left rail navigation, developer mode toggle, upgrade CTA, and project tree mock. Manages responsive collapse state and dev mode gating.
- **Pomodoro timer (`components/pomodoro-timer.tsx`):** Implements work/break modes, countdown logic, focus mode layout, quick settings dialog, active task sidebar, and upgrade prompts. Persists timer preferences via context.
- **Task management (`components/task-management.tsx`):** Provides in-memory CRUD with dialogs, filters, sorters, tag management, image preview, and Pomodoro estimates using mock data arrays.
- **Projects workspace (`components/projects-content.tsx`):** Displays project cards, filter/search controls, progress indicators, and project creation dialog backed by local mock state.
- **Dashboard (`components/dashboard-content.tsx`):** Renders overview stats, heatmap, leaderboard, pro-gated analytics and social tabs, and dev mode override.
- **Settings (`components/settings-content.tsx`):** Offers profile form, notification toggles, Pomodoro defaults, theme selection, random theme generator, and local persistence via `useCustomTheme`.

- **Theme system (components/theme-provider.tsx, components/theme-toggle.tsx):** Extends ext-themes with custom theme registry, local storage persistence, and dev mode state.
- **Upgrade modal (components/upgrade-modal.tsx):** Showcases pricing plans and pro features; invoked from multiple surfaces.
- **UI library (components/ui/)** 🍷 \* Shadcn-based primitives (Button, Card, Dialog, etc.) used across features.
- **Utility modules (lib/utils.ts, hooks/use-mobile.ts, hooks/use-toast.ts):** Provide class name helpers, responsive detection, and toast service stubs.

## Domain Model

- **Bounded contexts:**
  - *Productivity Core:* Tasks, projects, Pomodoro sessions.
  - *Personalization:* Pomodoro settings, themes, developer mode.
  - *Monetization:* Subscription tier, upgrade prompts.
- **Aggregates (current code-level structures):**
  - Task objects (components/task-management.tsx) with fields for priority, due date, Pomodoros.
  - Project objects (components/projects-content.tsx) with status, progress, team members.
  - PomodoroSettings (components/pomodoro-settings-provider.tsx) persisted in local storage.
  - Theme definitions (components/theme-provider.tsx) covering premium presets and custom entries.
  - UserProfile mock (components/settings-content.tsx) for profile editing.
- **Data lifecycle:** Currently all data is transient React state, reset on refresh except for settings/themes stored in local storage (pomodoro-settings, custom-themes). No archival, retention, or deletion policies exist yet.

```

mermaid erDiagram
    USER ||--o{ TASK : "creates"
    USER ||--o{ PROJECT : "owns"
    TASK }o--|| PROJECT : "belongs to"
    TASK ||--o{ POMODORO_SESSION : "tracked by"
    USER ||--|| POMODORO_SETTINGS : "has"
    USER ||--o{ THEME_PRESET : "saves"
    USER ||--o{ SUBSCRIPTION : "subscribes"
  
```

```

USER { string id PK string name string email string avatarUrl string tier }
TASK { string id PK string title string description string priority date dueDate int estimatedPomodoros int completedPomodoros boolean completed }
PROJECT { string id PK string name string status string priority int progressPercent date dueDate }
POMODORO_SESSION { string id PK datetime startedAt int durationMinutes string mode string taskId FK }
POMODORO_SETTINGS { string userId PK int workDuration int shortBreak int longBreak int longBreakInterval boolean autoStartBreaks boolean autoStartPomodoros boolean soundEnabled int volume }
THEME_PRESET { string id PK string name string primaryHex string secondaryHex string accentHex boolean premiumOnly }
SUBSCRIPTION { string id PK string plan datetime startedAt datetime expiresAt boolean trial }
  
```

## Data Flows

- **Focus session loop:** When the user starts a timer, PomodoroTimer transitions modes, updates countdown via useEffect, and increments session counters. Completing a work session increases completedPomodoros for the active task (mock data) and triggers celebratory UI. Preferences persist through PomodoroSettingsProvider writing to local storage.
- **Task lifecycle:** Task creation/edit dialogs collect metadata, update local state arrays, and re-render lists. Filtering and sorting happen in-memory, with computed views for overdue and prioritized items.

- **Theme personalization:** SettingsContent calls useCustomTheme.addCustomTheme, storing user-generated palettes in local storage and applying CSS variables to document.documentElement.

mermaid sequenceDiagram participant User participant UI as Next.js Client participant Timer as PomodoroTimer participant Settings as PomodoroSettingsProvider participant Storage as window.localStorage

User->>UI: Open /pomodoro UI->>Timer: Render component Timer->>Settings: Read durations (work/short/long) Settings->>Storage: load "pomodoro-settings" User->>Timer: Start Work Session Timer->>Timer: Tick every second (setInterval) Timer->>UI: Update progress bar & remaining time alt Session completes Timer->>Settings: update session counts (in state) Timer->>Storage: Settings persists updates Timer->>UI: Show celebration & next mode prompt end`

## API Design

- **Current state:** No REST/GraphQL/Server Actions are implemented; all data is mock client state.
- **Planned direction (per PRD):** Introduce authenticated APIs for tasks, projects, Pomodoro sessions, analytics events, and subscription management. Recommend designing OpenAPI specs under docs/openapi.yaml (to be created) with well-defined DTOs and pagination.
- **Versioning & compatibility:** Establish semantic versioning for the public API once introduced; default to `1` routes under `/api/v1/`. Support feature flags to guard beta endpoints.

## Security

- **Present gaps:** No authentication, authorization, or secure data handling. Preferences in local storage are unprotected, and developer mode defaults to enabled (components/theme-provider.tsx), exposing pro features in proto builds.
- **Required controls:**
  - Authentication (email/OAuth) and session management with short-lived tokens.
  - Role-based access (free vs pro vs admin vs team) enforced server-side.
  - TLS via hosting provider; encrypt at-rest data in the chosen database.
  - Secrets management for API keys and payment providers (e.g., `.env.local`, vault service).
  - Compliance: align with GDPR/CCPA for export/delete once user data is stored.
- **Near-term mitigation:** Disable default dev mode; guard pro-only UI behind entitlement checks once backend exists.

## Configuration

- 

`ext.config.mjs` sets `distDir` to `dist` and ignores ESLint/TypeScript build errors  risk for production builds.

- `sconfig.json` enables strict mode but allows JS and skips lib checks; includes generated types under `.next` and `/dist`.
- ThemeProvider uses local storage keys `custom-themes` and `isDevMode` to persist personalization.
- Environment variables are not referenced; expect to introduce `.env` for API URLs, Sentry DSNs, Stripe keys, etc.
- Feature flags: currently limited to `isDevMode`. Plan to centralize feature toggles via a provider that checks backend entitlements.

## Observability

- **Logging:** No structured logging beyond browser console; component-level console use is minimal.
- **Metrics:** Only Vercel Analytics (@vercel/analytics/next in `app/layout.tsx`); no custom events or funnels instrumented.
- **Tracing:** None.
- **Recommendations:** Adopt an analytics SDK (Segment, PostHog) to capture PRD-specified events. Introduce error monitoring (Sentry) and performance tracing (OpenTelemetry) once a backend exists. Document dashboards in `/docs/observability.md` when available.

## Scalability

- **Current posture:** Static Next.js front-end can scale via CDN; minimal server load.
- **Constraints:** All state is client-bound; lack of pagination, virtualization, or memoization could impact performance with large datasets.
- **Future strategy:**
  - Build APIs with pagination and caching (CDN or edge caches for read-heavy routes).
  - Offload heavy analytics to background jobs.
  - Consider server components or data fetching hooks to pre-render frequently accessed views.

## Resilience

- **Existing behavior:** Errors fall back to React default boundaries; no retry/backoff logic. Local storage persistence covers only settings, not core data.
- **Needed improvements:**
  - Implement service-level timeouts, retries, and offline handling once network calls exist.
  - Add optimistic UI with reconciliation for task/project mutations.
  - Define disaster recovery objectives (RPO/RTO) for backend data stores.

## Deployment and Operation

- **Build commands:** `pnpm install`, `pnpm lint`, `pnpm build` (produces `/dist`), `pnpm start`.
- **Runtime:** Ready for Vercel/Node hosting; static export not configured due to app router and client components.
- **Environments:** Only development environment implicit. Need staging and production pipelines with environment-specific configs.
- **CI/CD:** None. Recommend GitHub Actions workflow executing lint, type-check (without ignore flags), unit tests, and deploy on main merges. Include artifact upload for build output and preview deployments per PR.
- **Infrastructure:** To be defined ♦ likely Vercel for frontend, plus managed database (PostgreSQL) and serverless functions for APIs.

## Test Strategy

- **Current state:** No automated tests.
- **Proposed coverage:**
  - Unit tests with Vitest/React Testing Library for component logic (timer transitions, form validation).

- Integration tests for context providers and flows (settings persistence).
- End-to-end tests with Playwright covering key journeys (Pomodoro session, task CRUD, upgrade modal).
- Contract tests once APIs are introduced.
- Accessibility audits using Axe/Storybook.
- Document manual QA steps per release in /docs/testing.md (to be created).

## Dependencies

- **Framework & tooling:** ext@14.2.16, eact@18, yepscript@5, pnpm.
- **UI & design system:** Shadcn UI components (components/ui/\*\*), @radix-ui primitives, ailwindcss@4.
- **State & forms:** eact-hook-form, zod, class-variance-authority, clsx.
- **Visualization:** echarts (planned usage), custom heatmap logic in components/dashboard-content.tsx.
- **Utilities:** date-fns, ext-themes, @vercel/analytics.
- **Icons:** lucide-react.
- Monitor for updates and track licensing for bundled assets in public/.

## Risks & Trade-offs

- **Ignored build checks:** ext.config.mjs disables ESLint/TypeScript errors, risking production regressions.
- **Mock-only data:** All critical features rely on in-memory data; no persistence or multi-user capabilities, diverging from PRD goals.
- **Developer mode default:** Pro features are effectively free in current builds, potentially confusing stakeholders and masking entitlement bugs.
- **Accessibility gaps:** Custom themes and random generator may produce low-contrast palettes; WCAG compliance unverified.
- **Analytics debt:** Lack of telemetry prevents measuring PRD success metrics (activation, engagement).
- **Performance considerations:** Large client components (> 500 lines) risk re-render costs; no code splitting or virtualization.

## Future Evolution Plan

1. **Introduce backend persistence:** Define schema, implement CRUD APIs, and connect UI via async data fetching. Consider Next.js Server Actions or dedicated backend service.
2. **Add authentication & authorization:** Integrate with NextAuth or custom OAuth, secure routes, and attach user context to data.
3. **Implement billing workflow:** Embed Stripe checkout, manage trials, and enforce entitlements in both backend and UI.
4. **Telemetry & observability:** Instrument key events, add error tracking, and build dashboards for PRD metrics.
5. **Refine pro gating:** Replace dev-mode shortcuts with role checks and toggle experiments via feature flags.
6. **Hardening & QA:** Re-enable lint/type checks, add automated tests, and introduce accessibility audits.
7. **Collaboration roadmap:** Design team project sharing, role management, and real-time sync in line with stretch persona (Jordan).

## Assumptions

- Backend services will be implemented as Next.js API routes or serverless functions hosted alongside the frontend.
- PostgreSQL (or equivalent) will back tasks, projects, sessions, and users to meet persistence requirements.
- Stripe will handle billing, given its alignment with SaaS subscription needs.
- Email/OAuth authentication will be chosen; magic link is assumed for MVP unless specified otherwise.
- Observability stack (logging, metrics) will piggyback on Vercel-compatible tooling.

## Open Questions

- Which authentication method (magic link, OAuth providers, username/password) should we prioritize for MVP?
- What free-tier limits (tasks/projects/history) trigger upsell flows, and how are they enforced?
- Do we need real-time synchronization across sessions at launch, or is periodic refresh acceptable?
- What notification channels (browser push, email, mobile) must ship initially?
- Which analytics platform (Segment, PostHog, custom) fits budget and compliance targets?
- How will team collaboration handle ownership, sharing, and permissions?
- Where will backend services and databases be hosted (Vercel, AWS, Supabase)?
- What SLAs or performance targets (latency, uptime) must we commit to for paid customers?

## Glossary of Terms

- **Pomodoro:** Time-management technique involving focused work intervals separated by breaks.
- **Dev Mode:** Internal toggle (sidebar card) that unlocks pro-only UI without billing.
- **Pro Tier:** Paid subscription unlocking analytics, premium themes, and social features.
- **Focus Mode:** Full-screen timer experience minimizing distractions.
- **Contribution Heatmap:** Grid visualization of Pomodoro sessions over time on the dashboard.
- **Theme Preset:** Named collection of CSS custom properties defining visual palette.
- **Upgrade Modal:** Dialog presenting pricing plans and feature highlights.

## Change Log

- **2025-09-17:** Initial architecture baseline authored by Codex (GPT-5) based on repo snapshot and docs/PRD.md.