Web Dev Runbooks Pack

Streamline Your Web Development Workflow with Cursor

TL;DR Quick Start

1. **Scaffold:**

   ```
   You are my Web Tech Lead. Stack: Next.js + Node + Postgres. Return folder tree + route plan.
   ```

2. **Implement:**

   ```
   Generate pages/components/API for /pricing using SSR and typed endpoints.
   ```

3. **Test:**

   ```
   Create Jest + RTL tests for <PlanCard /> covering tiers, CTAs, edge cases.
   ```

4. **Accessibility:**

   ```
   Audit <PricingPage />; return WCAG fixes (roles, labels, focus order).
   ```

5. **CI:**

   ```
   Create GitHub Actions pipeline: build, lint, test, preview deploy.
   ```

1. Setup & Project Context

Define Project Stack and Constraints:

- Next.js (Frontend)
- Node.js (Backend)
- PostgreSQL (Database)
- SSR only for public pages, 200ms TTFB budget

Mini-Checklist:

- Confirm frameworks and database
- Define rendering strategy (SSR/CSR)
- Set performance goals

Cursor Prompt Example:

```
You are my Web Tech Lead. Stack: Next.js + Node + Postgres. Constraints: SSR only for public pages; 200ms TTFB budget. Return implementation plan with folders/components.
```

Project Folder Structure Example:

```
src/
  ├─ components/
```

```
├─ pages/
├─ utils/
├─ hooks/
```

---

2. Scaffolding the Feature

Define Component Trees & Routes:

- /pricing page → <PlanCard /> component for subscription tiers
- Cursor generates props, states, and route mapping

Cursor Prompt Example:

```
Generate a route map and component tree for '/pricing' with SSR, including <PlanCard /> spec
(props, states).
```

Define Data Models & API Contracts:

- plans table → GET /plans, POST /checkout
- Cursor can generate TypeScript types and OpenAPI stubs

Example OpenAPI Contract:

openapi: 3.0.3
paths:
/plans:
get:
responses:
'200':
description: OK
content:
application/json:
schema:
type: array
items: { $ref: '#/components/schemas/Plan' }
components:
schemas:
Plan:
type: object
required: [id, price]
properties:
id: { type: string }
price: { type: integer, description: "cents" }

---

3. Implement With Inline Edits

Next.js API Handler Example:

// app/api/plans/route.ts
import { NextResponse } from 'next/server';

export async function GET() {
const plans = [{ id: 'basic', price: 0 }, { id: 'pro', price: 2900 }];
return NextResponse.json(plans);
}

PlanCard Component Example:

```
// app/pricing/PlanCard.tsx
export default function PlanCard({ id, price }: { id: string; price: number }) {
const cta = price === 0 ? 'Get Started' : 'Start Trial';
return <button aria-label={`choose-${id}`}>{cta}</button>;
}
```

Mini-Checklist:

- Extract reusable hooks
- Optimize props
- Validate component consistency

4. Testing & Accessibility

Jest + RTL Example:

```
// __tests__/PlanCard.test.tsx
import { render, screen } from '@testing-library/react';
import PlanCard from '../app/pricing/PlanCard';

test('renders correct CTA by price', () => {
render(<PlanCard id="basic" price={0} />);
expect(screen.getByRole('button', { name: /choose-basic/ })).toHaveTextContent('Get Started');
});
```

Accessibility Checklist:

- Run Axe or Lighthouse scans
- Correct ARIA roles & labels
- Ensure keyboard navigation and color contrast

Cursor Prompt Example:

```
Audit <PricingPage /> for WCAG compliance and suggest fixes.
```

---

5. Performance & CI

Mini-Checklist:

- Audit with Lighthouse/PageSpeed Insights
- Optimize images & implement lazy loading
- Prefetch critical resources

GitHub Actions Example:

```
name: web-ci
on: [push, pull_request]
jobs:
build_test:
runs-on: ubuntu-latest
steps:
```

- uses: actions/checkout@v4
- uses: actions/setup-node@v4
with: { node-version: 20 }
- run: npm ci
- run: npm run lint && npm run test -- --ci
- run: npm run build

Proof Table (Before / After):

Metric	Before	After
LCP (Pricing)	2.4s	1.5s
JS Bundle (Pricing)	310 KB	220 KB
Unit Test Coverage	38%	72%

6. Documentation Best Practices

Maintain Clear Project Documentation:

- Concise READMEs for each module
- Include feature purpose, props, API usage, and examples

Cursor Prompt Example:

```
Create README for <PlanCard /> component including props, API usage, and example usage.
```

Use Standardized Templates:

- Sections: Setup, Workflow, Testing, Notes
- Include environment configs & dependencies

Integrate Documentation Inline:

- Update docs during feature implementation
- Review during PRs for accuracy

7. Logging & Observability

Structured Logging Example:

console.info(JSON.stringify({
msg: 'fetch plans',
path: '/api/plans',
ts: new Date().toISOString(),
userId,
}));

Monitor Critical Workflows:

- Log API calls, state changes, user actions
- Alert on repeated failures

Automate Log Analysis:

- Connect logs to monitoring tools (Datadog, Kibana)
- Create dashboards and alerts

Cursor Prompt Example:

```
Generate dashboard configuration for tracking API errors and response times.
```

8. Common Failure Modes

State Bloat & Prop Drilling:

- Analyze heavy state patterns
- Replace props with context/hooks
- Extract reusable logic

API Drift & Contract Issues:

- Validate frontend usage against OpenAPI
- Auto-generate TypeScript types
- Run tests to verify structure