==========

CLAUDE CODE PROMPT: Build TaniTrack Hatchery Management System

==========

🎯 PROJECT OVERVIEW

You are tasked with building **TaniTrack**, a Progressive Web App (PWA) for managing a Nigerian catfish hatchery operation. This is a production-ready application for Tani Nigeria Ltd to track spawning, production, sales, and financials for a 345,000 fingerlings/year operation.

Application Type: Mobile-first Progressive Web App (PWA)
Target Users: Farm owner and 2-3 farm assistants
Primary Devices: Android phones (primarily)
Network Conditions: Often poor (3G or unstable WiFi)
Location: Abuja, Nigeria

📚 DOCUMENTATION PROVIDED

You have been provided with complete documentation to build this application:

1. Product Requirements Document (PRD)

- **File:** `TaniTrack_PRD.md`
- **What it contains:** Business objectives, user roles, all features and modules, success criteria
- **Use it for:** Understanding the business context, feature priorities, and user workflows

2. Database Schema

- **File:** `TaniTrack_Database_Schema.sql`
- **What it contains:** 30 PostgreSQL tables, relationships, indexes, triggers, sample data
- **Use it for:** Creating the database structure and understanding data relationships

3. API Specifications

- **File:** `TaniTrack_API_Specifications.md`
- **What it contains:** 100+ REST API endpoints with request/response formats
- **Use it for:** Building the backend API exactly as specified

4. Wireframe Descriptions

- **File:** `TaniTrack_Complete_Wireframe_Descriptions.md`
- **What it contains:** 50+ screen layouts, design system, UI components
- **Use it for:** Building the frontend UI exactly as specified

---

🛠️ TECHNOLOGY STACK

Please use the following technologies:

Frontend

- **Framework:** React 18+ with Vite
- **Styling:** TailwindCSS (use their default configuration)
- **State Management:** Zustand (simpler than Redux for this size)
- **API Client:** React Query (with caching for offline support)
- **Routing:** React Router v6
- **Forms:** React Hook Form (with validation)
- **Date Handling:** date-fns (not Moment.js)

- **Charts:** Recharts
- **Icons:** Lucide React
- **UI Components:** Radix UI or HeadlessUI (for accessible primitives)

Backend

- **Runtime:** Node.js 18+ LTS
- **Framework:** Express.js (or Fastify if you prefer performance)
- **Authentication:** JWT (jsonwebtoken library)
- **Validation:** Joi or Zod
- **Database Client:** pg (node-postgres) for PostgreSQL
- **Password Hashing:** bcrypt
- **Environment Variables:** dotenv

Database

- **Database:** PostgreSQL 14+
- **Schema:** Use the provided `TaniTrack_Database_Schema.sql` file exactly as written

Development Tools

- **Linting:** ESLint with React and TypeScript rules
- **Formatting:** Prettier
- **Type Safety:** TypeScript (optional but recommended)

Deployment

- **Frontend:** Can be deployed to Vercel, Netlify, or any static host
- **Backend:** Can be deployed to Railway, Render, or DigitalOcean
- **Database:** PostgreSQL on Railway, Supabase, or DigitalOcean

---

🎨 DESIGN REQUIREMENTS

Mobile-First Approach

- Design for phones FIRST (320px minimum width)
- Then adapt for tablets (768px+) and desktop (1024px+)
- All interactive elements must be at least 44×44px (thumb-friendly)

Design System (from Wireframe Descriptions)

- **Primary Colors:**
  - Blue: `#0066CC` (Trust, water)
  - Green: `#22C55E` (Growth, success)
  - Orange: `#F97316` (Warnings)
  - Red: `#DC2626` (Errors)
- **Typography:**
  - Font: 'Inter' or system font stack
  - Base size: 16px (body text)
  - Headings: 24px, 20px, 18px
- **Spacing:**
  - Use Tailwind's default spacing scale
  - Base unit: 4px (Tailwind's spacing system)

Navigation

- **Mobile:** Bottom navigation bar (5 items: Home, Production, Sales, Reports, More)

- **Tablet/Desktop:** Side drawer + bottom nav
- **Always visible:** Top app bar with title and notifications

Offline Support

- Critical features MUST work offline:
  - View dashboard
  - View lists (batches, tanks, spawns)
  - Log feeding activities (queue for sync)
- Use service workers and local storage
- Show clear online/offline indicator

📋 DEVELOPMENT APPROACH

Phase 1: Foundation & Authentication (Week 1)

Setup:

1. Initialize React project with Vite
2. Set up TailwindCSS
3. Configure folder structure:

```
/frontend
  /src
    /components (reusable UI components)
    /pages (route components)
    /hooks (custom React hooks)
    /lib (utilities, API client)
    /assets (images, icons)
/backend
  /src
    /routes (API endpoints)
    /controllers (business logic)
    /models (database queries)
    /middleware (auth, validation)
    /utils (helpers)
```

Build:

- User authentication (login, logout)
- JWT token management
- Role-based access control (Owner, Senior, Junior)
- Protected routes
- Basic layout (top bar, bottom nav, content area)

Test:

- Can login with test credentials
- Tokens persist (localStorage)
- Protected routes redirect to login
- Logout clears tokens

Phase 2: Dashboard & Core Navigation (Week 2)

Build:

- Dashboard screen with:
  - Quick stats (total fish, active batches, sales, spawns)
  - Today's tasks (dummy data initially)

- Quick action buttons
- Recent activity feed

- Bottom navigation working
- Routing to main sections

Test:

- Dashboard loads quickly (<3 seconds)
- Navigation works smoothly
- Stats display correctly
- Responsive on mobile and desktop

Phase 3: Spawning Management (Week 2-3)
Build:

- Spawn list screen
- Spawn details screen
- Create new spawn form (multi-step)
- Spawn updates/stages
- Link spawns to batches

API Endpoints (from API Specifications):

- `GET /spawns` - List all spawns
- `POST /spawns` - Create new spawn
- `GET /spawns/:id` - Get spawn details
- `PUT /spawns/:id` - Update spawn
- `POST /spawns/:id/updates` - Add spawn update
- `DELETE /spawns/:id` - Delete spawn

Test:

- Can create spawn with broodstock selection
- Spawn stages update correctly
- List view shows all spawns
- Filters work (active, completed, etc.)

Phase 4: Production Tracking (Week 3-4)
Build:

- Batch list screen
- Batch details screen
- Growth sampling form
- Batch movements (transfer between tanks)
- Tank list and details

API Endpoints:

- `GET /batches` - List batches
- `POST /batches` - Create batch (from spawn)
- `GET /batches/:id` - Get batch details
- `POST /batches/:id/samples` - Record growth sample

- `POST /batches/:id/movements` - Transfer to new tank
- `GET /tanks` - List tanks
- `GET /tanks/:id` - Tank details

Test:

- Batches created from spawns
- Growth tracking works
- Tank transfers update correctly
- Survival rates auto-calculate

Phase 5: Sales & Customers (Week 4)
Build:

- Customer list and details
- Sales list screen
- Create sale form
- Sale details screen
- Payment tracking

API Endpoints:

- `GET /customers` - List customers
- `POST /customers` - Add customer
- `GET /sales` - List sales
- `POST /sales` - Create sale
- `PUT /sales/:id/payment` - Record payment

Test:

- Sales reduce batch inventory
- Customer purchase history works
- Payment status updates
- Revenue calculations correct

Phase 6: Feed & Financial (Week 5)
Build:

- Feed inventory screen
- Feeding log form
- Expense tracking
- Financial dashboard
- Batch costing

API Endpoints:

- `GET /feed` - Feed inventory
- `POST /feed/logs` - Log feeding
- `GET /expenses` - List expenses
- `POST /expenses` - Add expense
- `GET /financials/dashboard` - Financial summary

Test:

- FCR calculations correct

- Feed stock alerts work
- Expense allocation to batches
- Profitability reports accurate

Phase 7: Broodstock & Reports (Week 5-6)
Build:

- Broodstock management
- Health observations
- Basic reports (production, sales, financial)
- Settings and user management

Test:

- Broodstock rest periods enforced
- Health logs link to batches/tanks
- Reports generate correctly
- Export to Excel works

Phase 8: Polish & Testing (Week 6-8)
Build:

- Offline mode (service worker)
- Performance optimization
- Error handling improvements
- Loading states
- Empty states
- Success/error toasts

Test:

- Full user testing with farm staff
- Bug fixes
- Performance audit
- Accessibility check
- Mobile device testing

🔑 CRITICAL IMPLEMENTATION NOTES

1. Nigerian Context

- **Currency:** Always use ₦ (Naira) with comma separators: ₦1,500,000
- **Dates:** DD/MM/YYYY format (02/12/2024)
- **Phone Numbers:** Nigerian format (080XXXXXXXX, 081XXXXXXXX)
- **Time:** 24-hour format (14:30)

2. Data Validation

- Phone numbers: Must start with 070, 080, 081, 090, 091
- Nigerian phone: Exactly 11 digits
- Dates: Cannot be in the future (except for scheduled events)
- Quantities: Must be positive integers
- Weights: Can be decimals (e.g., 1.5kg, 8.2g)

- Prices: Must be positive numbers

3. Automatic Calculations

Implement these calculations client-side AND server-side:

Spawning:

- Estimated eggs = Female weight (kg) × 50,000
- Fertilization rate = (Fertilized / Total) × 100
- Hatch rate = (Hatched / Fertilized) × 100

Production:

- Survival rate = (Current count / Initial count) × 100
- Average daily gain = (Current weight - Previous weight) / Days
- FCR = Total feed given (kg) / Total weight gain (kg)

Financial:

- Total sale amount = Quantity × Price per fish
- Profit = Revenue - Total costs
- Profit margin = (Profit / Revenue) × 100

4. Permission Enforcement

Owner (Full Access):

- Can do everything
- See all financial data
- Manage users
- Delete records

Senior Assistant:

- Create/edit production records
- Create sales
- View financial summaries (not detailed costs)
- Cannot delete or manage users

Junior Assistant:

- Log feeding only
- View tank assignments (read-only)
- Report health observations
- No financial access

Enforce these permissions on BOTH frontend (UI hiding) and backend (API rejection)

5. Error Handling

User-Friendly Errors:

- ❌ Bad: "ECONNREFUSED"
- ✅ Good: "Cannot connect to server. Check your internet connection."

Error Types:

- Network errors: "No internet connection. Your changes will be saved and synced later."
- Validation errors: "Phone number must be 11 digits"
- Permission errors: "You don't have permission to perform this action"
- Not found errors: "Batch not found"

6. Loading States

- Show skeleton loaders for list screens

- Show spinners for button actions
- Disable buttons during API calls
- Show progress for multi-step forms

7. Success Feedback

- Toast notifications for actions: "Spawn created successfully!"
- Navigate to detail view after creation
- Optimistic updates (show change immediately, sync in background)

📊 DATA FLOW EXAMPLE: Creating a Spawn

User Flow:

1. User taps "New Spawn" button on dashboard
2. Navigate to `/production/spawns/new`
3. Multi-step form:
   - Step 1: Date, time, hormone type
   - Step 2: Select 2 females and 2 males (from broodstock)
   - Step 3: Review and confirm
4. On submit:
   - POST to `/api/spawns`
   - Show loading spinner
   - On success: Toast "Spawn created!" → Navigate to spawn details
   - On error: Show error toast, stay on form

Backend Processing:

1. Validate request (auth token, permissions, data format)
2. Check broodstock availability (not already used today)
3. Calculate estimated eggs (female weights × 50,000)
4. Insert into `spawns` table
5. Create spawn_updates entry (stage: "fertilized")
6. Return spawn object with generated ID and code

Frontend Display:

1. Navigate to `/production/spawns/:id`
2. Fetch spawn details: GET `/api/spawns/:id`
3. Show spawn timeline
4. Show broodstock used
5. Show estimated eggs
6. Next action button: "Update Spawn Status"

🚀 GETTING STARTED CHECKLIST

Before you start coding:

- [ ] Read the PRD completely (understand the business)
- [ ] Review the Database Schema (understand data relationships)
- [ ] Scan the API Specifications (know what endpoints to build)
- [ ] Look at the Wireframe Descriptions (see what the UI should look like)

Initial setup:

- [ ] Create frontend project: `npm create vite@latest tanitrack-frontend -- --template react`
- [ ] Create backend project: `npm init` in a separate folder
- [ ] Install dependencies (listed in Technology Stack section)
- [ ] Set up PostgreSQL database
- [ ] Run the Database Schema SQL file
- [ ] Configure environment variables (.env files)

First code to write:

- [ ] Backend: Database connection
- [ ] Backend: Authentication endpoints (login, verify token)
- [ ] Frontend: API client setup (axios or fetch wrapper)
- [ ] Frontend: Login screen
- [ ] Frontend: Protected route wrapper
- [ ] Frontend: Basic layout (top bar, bottom nav)

Suggested prompt to Claude Code:

```
I need to build TaniTrack, a Progressive Web App for catfish hatchery management.

I'm providing you with four documents:
1. Product Requirements (business context and features)
2. Database Schema (PostgreSQL tables)
3. API Specifications (all endpoints)
4. Wireframe Descriptions (UI designs)

Let's start with the authentication system. Please:
1. Set up the backend with Express and PostgreSQL
2. Create /auth/login and /auth/verify endpoints
3. Implement JWT token generation
4. Set up the frontend with React + Vite + TailwindCSS
5. Build the login screen (from Wireframe Descriptions)
6. Create an API client for making authenticated requests

Use the Database Schema for the users table structure.
Use the API Specifications for the exact request/response formats.
Use the Wireframe Descriptions for the login screen layout.

Let me know when this is ready and I'll test it before we move to the next module.
```

🎯 DEVELOPMENT BEST PRACTICES

Code Organization

- Keep components small and focused (single responsibility)
- Extract reusable logic into custom hooks
- Use proper TypeScript types (or PropTypes if using JS)
- Comment complex business logic
- Follow consistent naming conventions

Performance

- Lazy load routes: `const Dashboard = lazy(() => import('./Dashboard'))`
- Memoize expensive calculations: `useMemo`, `useCallback`
- Debounce search inputs
- Paginate long lists (20-50 items per page)
- Compress images before upload

Testing

- Test each module before moving to the next
- Manual testing on actual mobile devices

- Test with poor network conditions (Chrome DevTools throttling)
- Test with different screen sizes
- Test all user roles (Owner, Senior, Junior)

Git Workflow

- Commit frequently with clear messages
- Use branches for features: `feature/spawning-module`
- Tag releases: `v1.0.0-mvp`
- Write a clear README

📦 DELIVERABLES

At the end of development, provide:

1. **Source Code**
   - Frontend repository (or folder)
   - Backend repository (or folder)
   - Clear folder structure
   - .env.example files (without secrets)
2. **Documentation**
   - README.md with setup instructions
   - How to run locally
   - How to deploy
   - Environment variables needed
3. **Database**
   - SQL schema file (already provided)
   - Seed data script (optional)
   - Migration scripts (if used)
4. **Deployment Guide**
   - Where to deploy frontend (Vercel/Netlify)
   - Where to deploy backend (Railway/Render)
   - How to set up PostgreSQL
   - Environment variables for production
5. **Testing Guide**
   - Test credentials for each role
   - Key workflows to test
   - Known issues or limitations

⚠️ IMPORTANT REMINDERS

What Makes This Project Unique

1. Mobile-First is Critical

- Farm staff use phones, not laptops
- Design for 375px width screens
- Large touch targets (44×44px minimum)
- Works in bright sunlight (high contrast)

2. Offline is Essential

- Farm often has poor internet
- Critical features must work offline
- Queue actions and sync later
- Clear online/offline indicator

3. Nigerian Context Matters

- Currency: ₦ (Naira)
- Date format: DD/MM/YYYY
- Phone format: 080XXXXXXXX
- Local terminology (fingerlings, not juveniles)

4. Data Integrity is Critical

- This is a business-critical system
- Data loss would be devastating
- Implement proper validation
- Use database constraints
- Regular automated backups

5. User Adoption Depends on Ease of Use

- Farm staff have varying tech skills
- Make it OBVIOUS how to use
- Minimize text input (use dropdowns)
- Clear error messages
- Instant feedback

Common Pitfalls to Avoid

❌ **Don't:**

- Overcomplicate the UI with too many options
- Require internet for basic viewing
- Use small touch targets (<44px)
- Show technical error messages to users
- Build desktop-first (mobile is primary)
- Add features not in the PRD (scope creep)
- Skip input validation
- Hardcode values (use constants)

✅ **Do:**

- Keep the UI clean and simple
- Make key features work offline
- Use large, thumb-friendly buttons
- Show friendly error messages
- Start with mobile design
- Stick to the MVP scope
- Validate all inputs
- Use environment variables

🎓 LEARNING RESOURCES

If you need to learn or reference any technologies:

React + Vite:

- React docs: https://react.dev
- Vite docs: https://vitejs.dev

TailwindCSS:

- Docs: https://tailwindcss.com/docs
- Cheat sheet: https://nerdcave.com/tailwind-cheat-sheet

React Query:

- Docs: https://tanstack.com/query/latest/docs/react/overview

Express.js:

- Getting started: https://expressjs.com/en/starter/installing.html

PostgreSQL:

- Node.js client: https://node-postgres.com

JWT:

- Introduction: https://jwt.io/introduction

🎯 SUCCESS CRITERIA

You'll know you've succeeded when:

1. **Functional:**
   - All 10 core modules work as specified
   - Users can complete all key workflows
   - Data persists correctly in the database
   - Reports calculate accurately
2. **Usable:**
   - Farm staff can use it without constant help
   - Works smoothly on mobile devices
   - Offline mode functions for critical features
   - Loads fast even on 3G
3. **Reliable:**
   - No data loss
   - Proper error handling
   - Validation prevents bad data
   - Backups work automatically
4. **Professional:**
   - Clean, modern UI matching wireframes
   - Consistent design throughout
   - No bugs in critical flows
   - Ready for production use

📞 SUPPORT & QUESTIONS

If you need clarification:

- Refer back to the four documentation files

- PRD has business context and priorities
- Database Schema shows data relationships
- API Specifications define exact endpoints
- Wireframes show UI layouts

If something is unclear:

- Ask specific questions with context
- Reference the relevant document and section
- Suggest an approach and ask for confirmation

🚀 LET'S BUILD TANITRACK!

You now have everything you need:

- ✅ Complete Product Requirements (PRD)
- ✅ Database Schema (30 tables)
- ✅ API Specifications (100+ endpoints)
- ✅ Wireframe Descriptions (50+ screens)
- ✅ Technology stack defined
- ✅ Development roadmap (8-week plan)
- ✅ This comprehensive prompt

Start with Phase 1 (Authentication & Foundation) and build module by module.

Remember: The goal is to create a production-ready application that helps Tani Nigeria Ltd manage their catfish hatchery more efficiently. This is a real business tool, not a demo project.

Good luck, and happy coding! 🐟💻

END OF CLAUDE CODE PROMPT

Document Version: 1.0
Created: December 2024
For: Claude Code Development
Project: TaniTrack v1.0 MVP