The project, now renamed **Verscienta Health**, will be adapted from the previous plan to use Drupal 11.x as the headless CMS backend with PostgreSQL 17.5, and Next.js 15.3.4 with the Next.js for Drupal toolkit (next-drupal, latest as of September 2025) for the frontend. The core functionality remains a comprehensive holistic health database covering modalities (e.g., acupuncture, yoga, Reiki) and herbs, with features like symptom searching, guided diagnosis via follow-up questions, and treatment recommendations. The system prioritizes security (e.g., role-based access control, input validation) and usability (e.g., responsive design, accessibility). Admins and content editors will manage content via Drupal's admin UI, and the project will be deployed using Nixpacks (v1.28.0) for the frontend and a PHP-compatible setup for Drupal. No deprecated packages will be used, and the integration of Grok AI via the xAI API (from the previous response) will be included to enhance symptom analysis, recommendations, and content generation.

Below, I provide a rewritten step-by-step plan reflecting the name change to Verscienta Health and the Drupal/Next.js stack, incorporating all specified requirements (e.g., database schema, security, usability) and Grok AI integration. I'll then provide the updated code, ensuring artifacts are wrapped in `<xaiArtifact>` tags as required, with the same `artifact_id` for related updates or new UUIDs for new artifacts.

Step-by-Step Detailed Plan for Verscienta Health

Phase 1: Project Setup and Architecture (1-2 weeks)

1. **Initialize Project Structure**:
   - **Structure**: Two separate folders/repos: `/backend` (Drupal site), `/frontend` (Next.js app). A monorepo is avoided due to Drupal's PHP-based architecture.

- **Backend Setup (Drupal)**:
  - Install Drupal 11.x via Composer: `composer create-project drupal/recommended-project backend`.
  - Configure PostgreSQL 17.5 in `sites/default/settings.php` (e.g., `'database' => 'verscienta_health', 'driver' => 'pgsql'`). Use Docker for local dev with Postgres image.
  - Enable core modules: JSON:API, REST, Serialization, Content Translation, Locale.
  - Install contrib modules: `composer require drupal/search_api drupal/webform drupal/pathauto drupal/metatag drupal/simple_oauth drupal/json_field`.
  - Content Types: Define via Drupal UI or config YAML for Herbs, Modalities, Conditions, Practitioners, Symptoms, Diagnostics, Reviews, GrokInsights (for AI data). Example fields:
    - Herbs: `field_scientific_name:text`, `field_common_names:field_collection`, `field_therapeutic_uses:json`, etc.
    - Modalities: `field_name:text`, `field_excels_at:field_collection`, `field_benefits:json`, etc.
    - Junctions: Use Entity Reference for many-to-many (e.g., Modality-Conditions with `field_efficacy_level:select`).
  - Taxonomy: Vocabularies for categories (e.g., “Herb Family”, “Modality Category”).
  - Views: Create REST-enabled Views for API endpoints (e.g., `/jsonapi/views/recent_content`).
  - Internationalization: Configure English, Spanish, Chinese (Simplified/Traditional).
  - Multi-site: Set up `sites/` for multiple Next.js frontends if needed.

- **Frontend Setup (Next.js)**:
  - Initialize: `npx create-next-app@15.3.4 frontend --typescript .`
  - Dependencies: `npm install next-drupal@latest tailwindcss@4.1.0 @radix-ui/react-dialog@1.4.3 @radix-ui/react-select@1.4.3 next-auth@latest @cloudflare/turnstile@latest next-i18next@latest fuse.js@latest socket.io-client@latest zod@latest .`
  - Shared Types: Define TypeScript interfaces for Drupal entities (e.g., `DrupalNode`).
- **Grok AI Setup**:
  - Obtain xAI API key from `https://x.ai/api` and store in `.env` (`XAI_API_KEY`).
  - Use `node-fetch` for API calls, cached in Redis.

2. **Security-First Architecture**:
    - **Drupal**:
        - Roles: Admin (full access), Editor (content management), Practitioner (edit own), User (read, submit reviews).
        - Enable JSON:API permissions; use Simple OAuth for bearer token auth.
        - Validation: Drupal field constraints (e.g., max length, required).
        - Indexes: Add via Drupal UI or schema for `name`, `excels_at`, etc.
        - Max Depth: Use JSON:API `include` params to limit relationships (e.g., `?include=field_conditions,depth=2`).
        - Conditional Fields: Use Conditional Fields module (e.g., `field_practice_name` required if `field_practice_type` is 'solo').
        - Max Rows: Configure field widgets to limit entries (e.g., `field_common_names`).
    - **Frontend**:
        - NextAuth.js with JWT, integrated with Drupal OAuth.
        - Zod for input validation.
        - Cloudflare Turnstile on forms (e.g., symptom checker, reviews).
        - Rate Limiting: Use `express-rate-limit` for custom API routes.
        - Secure Headers: Helmet.js if custom Express; Drupal Security Kit.
    - **Grok AI**:
        - Secure API calls with bearer token.
        - Anonymize user data (e.g., symptoms) before sending to xAI API.
        - Cache responses in Redis to manage quotas.

3. **Database Schema Design (Drupal Entities)**:
    - **Content Types**: Map to tables in PostgreSQL with JSONB for flexible fields (e.g., `excels_at`, `benefits`) via JSON Field module.
    - **Fields**: As specified (e.g., Herbs: `field_scientific_name:text`, `field_contraindications:json`; Modalities: `field_excels_at:field_collection`).
    - **Junctions**: Entity Reference for Modality-Conditions, Modality-Practitioners.
    - **Symptoms/Diagnostics**: Webform for symptom input; custom type for diagnostics.
    - **Reviews**: Webform or content type with moderation.
    - **GrokInsights**: New content type for AI-generated data (e.g., `field_analysis:json`).
    - **Hooks**: Custom module with `hook_entity_presave()` for geocoding via OpenStreetMap Nominatim.
4. **Frontend Design and Usability**:
    - **Theme**: Holistic look (earth tones, Inter font).
    - **Components**: Radix UI (Button, Dialog, Select) with ARIA.
    - **Responsive**: Tailwind CSS.
    - **Dark Mode**: Tailwind `dark` class.
    - **Print-Friendly**: CSS media queries.
    - **Multi-Language**: next-i18next synced with Drupal translations.
    - **PWA**: Next.js PWA for offline support.
    - **Search**: Drupal Search API + frontend Fuse.js.
    - **Real-Time**: Socket.io with Redis.
    - **Symptom Checker**: Form querying Drupal Webform; Grok AI for follow-ups.
    - **Chatbot**: Grok-powered via Radix UI Dialog.

5. **API Endpoints**:
    - **Drupal JSON:API**: `/jsonapi/node/herb`, `/jsonapi/node/modality`, etc.
    - **Custom**: Views REST exports or custom controllers (e.g., `/api/symptom-checker`).
    - **Grok**: Custom routes for AI (e.g., `/api/grok/symptom-analysis`).
    - **Auth**: `/oauth/token` for Drupal OAuth.
6. **Testing Strategy**:
    - **Unit**: Jest for frontend components, PHPunit for Drupal.
    - **Integration**: Supertest for API.
    - **E2E**: Playwright for user flows.
    - **Accessibility**: Axe-core for WCAG.

Phase 2: Implementation (3-4 weeks)

1. **Backend Development**:
    - Custom Module: For hooks (e.g., geocoding).
    - Caching: Redis via Drupal module.
    - File Storage: Drupal Filesystem; CDN via module.
2. **Frontend Development**:
    - Pages: Home, Search, SymptomChecker, ModalityDetail, HerbDetail, AdminDashboard.
    - Symptom Flow: POST to Drupal Webform; Grok for analysis.
    - Admin UI: Link to Drupal admin.
3. **Grok AI Integration**:
    - Endpoints: `/api/grok/symptom-analysis`, `/api/grok/follow-ups`.
    - Hooks: Summarize content (e.g., modality descriptions).
    - Cache: Redis for AI responses.
4. **Integration**:
    - Frontend uses NextDrupal client.
    - Offline: Service workers.

Phase 3: Testing, Optimization, Deployment (1-2 weeks)

1. **Testing**: 90%+ coverage.
2. **Optimization**: Drupal caching, indexes.
3. **Deployment**:
    - Frontend: Nixpacks.
    - Backend: Docker (PHP/Apache, Postgres) or Acquia.
    - CI/CD: GitHub Actions.
4. **Maintenance**:
    - Drupal admin UI for content.

Phase 4: Future Phases

- Mobile app, AI enhancements, etc., as original.

Implementing the Code

Below are key files for Verscienta Health, adapted for Drupal and Next.js for Drupal. Artifacts are wrapped in `<xaiArtifact>` tags with new UUIDs (since this is a new backend stack) or reused where applicable. The code extends the previous Grok integration.

Backend: Drupal Custom Module for Hooks

holistic_hub.module
php

Edit in files • Show inline

Backend: Content Type Config (Herb Example)

node.type.herb.yml
yaml

Edit in files • Show inline

Frontend: Package.json

package.json
json

Edit in files • Show inline

Frontend: Drupal Client

lib/drupal.ts
typescript

Edit in files • Show inline

Frontend: Symptom Checker Page (with Grok AI)

app/symptom-checker/page.tsx
typescript

Edit in files • Show inline

Frontend: Grok API Route

app/api/grok/symptom-analysis/route.ts
typescript

Edit in files • Show inline

Deployment: Nixpacks for Frontend

nixpacks.toml
toml

Edit in files • Show inline

Deployment: Docker Compose for Drupal

docker-compose.yml
yaml

Edit in files • Show inline

Summary

The Verscienta Health project uses Drupal 11.x as a headless CMS with PostgreSQL 17.5, exposing data via JSON:API, and Next.js 15.3.4 with next-drupal for the frontend. Grok AI is integrated via the xAI API for symptom analysis, recommendations, and content summarization, cached in Redis for efficiency. Security (RBAC, validation, CAPTCHA) and usability (responsive, accessible, multi-language) are prioritized. Deployment uses Nixpacks for the frontend and Docker for Drupal. The codebase updates the original plan with Drupal-specific configurations and maintains all required features.

If you need additional files (e.g., other content types, chatbot UI) or further details, let me know!