DeepAgent Best Practices for Genesis Provenance

Complete Guide to Building Full-Stack Applications with DeepAgent

This guide provides battle-tested prompts and strategies for efficiently building and iterating on Genesis Provenance (and similar full-stack applications) using DeepAgent.

Table of Contents

Understanding DeepAgent’s Strengths

✅ What DeepAgent Excels At:

1. **End-to-End Web Applications**
   - Next.js, React, Node.js full-stack apps
   - Marketing sites + authenticated dashboards
   - Database design and integration

2. **Iterative, Phase-Based Development**
   - Breaking large projects into manageable phases
   - Building foundation first, then adding features
   - Testing and refining incrementally

3. **Production-Ready Code**
   - Best practices built-in
   - Security, performance, scalability
   - Deployment configurations

4. **Complex Integrations**
   - Stripe, S3, Auth, APIs

- Database migrations
- Third-party services

⚠️ What to Break Into Smaller Tasks:

1. **Large Feature Sets**
   - Don't ask for "entire app" in one go
   - Split into phases across conversations

2. **Multiple Complex Integrations**
   - One integration per conversation when possible
   - Test each thoroughly before adding next

3. **Significant Refactors**
   - Address architectural changes separately
   - Plan refactors in dedicated conversations

Phase-Based Development Strategy

Recommended Phase Structure for Full-Stack Apps:

Phase 1: Foundation (Conversation 1)

Goal: Deployable MVP with core infrastructure

```
I'm building [APP_NAME] - [ONE_SENTENCE_DESCRIPTION].

Phase 1 Goals:
1. Marketing site with [LIST PAGES]
2. Basic authentication (email/password)
3. Dashboard shell with navigation
4. Database schema for [CORE MODELS]
5. Deployment documentation

Tech Stack:
- Next.js 14 + TypeScript + Tailwind
- PostgreSQL + Prisma
- NextAuth.js
- Hosting: Vercel

Build Phase 1 with production-ready code. We'll add [DEFERRED FEATURES] in future phas
es.
```

Phase 2: Core Features (Conversation 2)

Goal: Main user workflows and data management

```
Continuing Genesis Provenance from previous conversation.

Phase 2 Goals:
1. [FEATURE 1] - e.g., Asset onboarding wizard
2. [FEATURE 2] - e.g., File uploads to S3
3. [FEATURE 3] - e.g., Items list/grid with filters
4. [FEATURE 4] - e.g., Item detail pages

Database changes needed:
- Add tables: [TABLE_NAMES]
- New enums: [ENUM_NAMES]

Build these features and update the database schema accordingly.
```

Phase 3: Integrations (Conversation 3)

Goal: External service integrations

```
Phase 3: Add integrations to Genesis Provenance.

1. n8n webhooks for:
   - POST /api/webhooks/asset-created
   - POST /api/webhooks/provenance-update

2. AI service abstraction layer:
   - lib/ai-service.ts with analyzeAsset()
   - Mock implementation for testing

3. Webhook documentation for n8n setup

Provide working code and integration examples.
```

Phase 4: Payments & Subscriptions (Conversation 4)

```
Phase 4: Add Stripe subscription system.

Plans:
- Collector: $29/month
- Reseller: $99/month
- Enterprise: Custom

Implement:
1. Stripe checkout flow
2. Subscription management UI
3. Webhook handlers
4. Plan gating logic

Provide Stripe test mode setup instructions.
```

Effective Prompt Templates

1. Starting a New Project

```
I'm building [PROJECT_NAME] - [DETAILED_DESCRIPTION].

**Core Features:**
1. [FEATURE 1]
2. [FEATURE 2]
3. [FEATURE 3]

**User Roles:**
- [ROLE 1]: [DESCRIPTION]
- [ROLE 2]: [DESCRIPTION]

**Tech Stack:**
- Frontend: Next.js 14 + TypeScript + Tailwind CSS
- Backend: Next.js API routes
- Database: PostgreSQL + Prisma ORM
- Auth: NextAuth.js (email/password)
- Storage: AWS S3
- Hosting: Vercel

**Phase 1 Scope:**
Build the foundation:
- Marketing site (Home, About, Contact pages)
- Authentication system
- Basic dashboard layout
- Database schema for [MODELS]
- Deployment setup

Defer to later phases:
- [COMPLEX_FEATURE_1]
- [INTEGRATION_1]
- [INTEGRATION_2]

I have intermediate coding skills. Provide production-ready code with clear documenta-
tion.
```

2. Adding New Features

```
Add [FEATURE_NAME] to existing [PROJECT_NAME] app.

**Current State:**
The app has [EXISTING_FEATURES].

**New Feature Requirements:**
[DETAILED_DESCRIPTION of what the feature should do]

**User Flow:**
1. User [ACTION_1]
2. System [RESPONSE_1]
3. User [ACTION_2]
4. System [RESPONSE_2]

**Database Changes:**
- New tables: [TABLE_NAMES]
- New fields in existing tables: [FIELD_DETAILS]

**UI Components Needed:**
- [COMPONENT_1]
- [COMPONENT_2]

Integrate this seamlessly with the existing codebase.
```

3. Fixing Bugs or Issues

```
**Issue:** [CLEAR_DESCRIPTION of the problem]

**Expected Behavior:**
[What should happen]

**Actual Behavior:**
[What actually happens]

**Steps to Reproduce:**
1. [STEP_1]
2. [STEP_2]
3. [STEP_3]

**Error Messages (if any):**
```

[PASTE_ERROR_LOGS]

```
**Relevant Files:**
- [FILE_PATH_1]
- [FILE_PATH_2]

Fix this issue and explain the root cause.
```

4. Database Schema Changes

```
Extend the database schema for [PROJECT_NAME].

**New Tables Needed:**

1. **[TABLE_NAME_1]**
   - Fields: [FIELD_1, FIELD_2, ...]
   - Relationships: [DESCRIBE_RELATIONSHIPS]
   - Indexes: [INDEX_FIELDS]

2. **[TABLE_NAME_2]**
   - Fields: [FIELD_1, FIELD_2, ...]
   - Relationships: [DESCRIBE_RELATIONSHIPS]

**New Enums:**
- [ENUM_NAME_1]: [VALUE_1, VALUE_2, ...]
- [ENUM_NAME_2]: [VALUE_1, VALUE_2, ...]

**Modifications to Existing Tables:**
- [TABLE_NAME]: Add field [FIELD_NAME] ([TYPE])

Update the Prisma schema and create migrations. Preserve existing data.
```

5. Third-Party Integration

```
Integrate [SERVICE_NAME] into [PROJECT_NAME].

**Integration Goals:**
- [GOAL_1]
- [GOAL_2]

**Required Functionality:**
1. [FUNCTIONALITY_1]
2. [FUNCTIONALITY_2]

**API Credentials:**
- I have: [LIST_WHAT_YOU_HAVE]
- Need help with: [WHAT_YOU_NEED_GUIDANCE_ON]

**User Flow:**
[DESCRIBE how users will interact with this integration]

Provide:
1. Environment variables needed
2. Code implementation
3. Setup instructions
4. Error handling
```

6. Authentication & Authorization

```
Set up authentication for [PROJECT_NAME] with these requirements:

**Auth Methods:**
- [X] Email/Password
- [ ] Google OAuth
- [ ] Magic Link

**User Roles:**
- [ROLE_1]: Can [PERMISSIONS]
- [ROLE_2]: Can [PERMISSIONS]
- [ROLE_3]: Can [PERMISSIONS]

**Protected Routes:**
- /dashboard/* - Requires authentication
- /admin/* - Requires admin role
- /api/items/* - Requires authentication + ownership check

**Security Requirements:**
- Password hashing (bcrypt)
- JWT sessions
- RBAC enforcement
- Multi-tenant data isolation

Implement with NextAuth.js.
```

Database & Schema Management

Best Practices for Prisma/Database:

✅ **DO:**

```
When asking for database changes:

"Update the Prisma schema to add:
1. [SPECIFIC TABLE/FIELD ADDITIONS]
2. [SPECIFIC RELATIONSHIPS]
3. [SPECIFIC INDEXES]

Preserve all existing data and relationships.
Create migrations using 'prisma migrate dev'."
```

❌ **DON’T:**

```
# Too vague:
"Add some tables for user data"

# Better:
"Add a 'user_profiles' table with fields:
- bio (Text, nullable)
- avatarUrl (String, nullable)
- phoneNumber (String, nullable)
Relation: One-to-One with 'users' table."
```

Schema Design Prompt Template:

```
Design a database schema for [FEATURE_NAME].

**Data to Store:**
- [DATA_TYPE_1]: [DESCRIPTION]
- [DATA_TYPE_2]: [DESCRIPTION]

**Relationships:**
- [TABLE_A] has many [TABLE_B]
- [TABLE_C] belongs to [TABLE_D]

**Query Patterns:**
- Frequently filter by: [FIELDS]
- Frequently join with: [TABLES]
- Need full-text search on: [FIELDS]

Create normalized schema with proper indexes.
Use Prisma best practices.
```

Authentication & Authorization

NextAuth.js Setup Prompt:

```
Set up NextAuth.js authentication with:

**Providers:**
- Credentials (email/password with bcrypt)
- [Optional: Google, GitHub, etc.]

**Database:**
- Store users in existing 'users' table
- Fields: id, email, passwordHash, role, createdAt

**Session Strategy:**
- JWT tokens (not database sessions)
- Include in session: userId, email, role

**Protected Routes:**
- All /dashboard/* routes require authentication
- /admin/* requires role='admin'
- Redirect to /auth/login if not authenticated

**Custom Pages:**
- Login: /auth/login
- Signup: /auth/signup
- Error: /auth/error

Provide:
1. lib/auth-options.ts configuration
2. Login/signup page components
3. Middleware for route protection
4. API route protection examples
```

File Uploads & S3 Integration

S3 File Upload Prompt:

~~~
Implement file upload system using AWS S3:

**Requirements:**
- Upload types: Photos (JPG, PNG) and Documents (PDF)
- Multiple file uploads per item
- Direct upload to S3 (not local storage)
- Store S3 keys in database table 'media_assets'

**Flow:**
1. User selects files in browser
2. Files upload to S3 via API route
3. S3 keys saved to database
4. Display signed URLs for viewing/downloading

**Database Schema:**
```prisma
model MediaAsset {
  id               String   @id @default(uuid())
  itemId           String
  cloudStoragePath String   // S3 key
  fileName         String
  fileSize         Int
  mimeType         String
  uploadedAt       DateTime @default(now())
}
~~~

Provide:

1. lib/s3.ts utility functions
2. POST /api/upload API route
3. File upload form component
4. Image gallery/document list component
5. Environment variables needed

~~~
---

## API & Webhook Development

### REST API Endpoint Prompt:

```markdown
Create REST API endpoints for [RESOURCE_NAME]:

**Endpoints:**

1. **POST /api/[resource]**
   - Create new [resource]
   - Auth required: Yes
   - Body: { [FIELDS] }
   - Returns: { id, [fields], createdAt }

2. **GET /api/[resource]**
   - List all [resources] for current user/org
   - Auth required: Yes
   - Query params: page, limit, filter
   - Returns: { items: [], total, page }

3. **GET /api/[resource]/[id]**
   - Get single [resource]
   - Auth required: Yes
   - Ownership check: Yes
   - Returns: { id, [fields], ... }

4. **PUT /api/[resource]/[id]**
   - Update [resource]
   - Auth required: Yes
   - Ownership check: Yes
   - Body: { [updatable_fields] }

5. **DELETE /api/[resource]/[id]**
   - Delete [resource]
   - Auth required: Yes
   - Ownership check: Yes

**Validation:**
- Use Zod schemas for input validation
- Return proper error codes (400, 401, 403, 404, 500)

**Security:**
- Verify user authentication
- Check resource ownership
- Sanitize inputs
- Rate limiting

Provide working Next.js API routes with error handling.
~~~

n8n Webhook Integration:

~~~
Create webhook endpoints for n8n integration:

**Outgoing Webhooks (App → n8n):**

1. **Asset Created Hook**
   - Trigger: When new asset is added
   - URL: process.env.N8N_WEBHOOK_URL + '/asset-created'
   - Payload:
   ```json
   {
     "itemId": "uuid",
     "organizationId": "uuid",
     "category": "watch",
     "brand": "Rolex",
     "model": "Submariner",
     "photos": ["s3-key-1", "s3-key-2"],
     "documents": ["s3-key-1"],
     "timestamp": "2025-01-01T00:00:00Z"
   }
   ```

**Incoming Webhooks (n8n → App):**

1. **POST /api/webhooks/provenance-update**
   - Receives AI analysis results
   - Auth: Verify webhook signature
   - Body:
   ```json
   {
     "itemId": "uuid",
     "status": "verified" | "flagged" | "rejected",
     "riskScore": 85,
     "analysis": "Detailed text...",
     "confidence": 0.95
   }
   ```
   - Action: Update item in database, create provenance event

**Provide:**
1. Utility functions for sending webhooks
2. API routes for receiving webhooks
3. Signature verification
4. Error handling and retries
5. Documentation for n8n workflow setup
~~~

Third-Party Integrations

Stripe Integration Prompt:

```
Integrate Stripe for subscription payments:

**Plans:**
- Collector: $29/month (price_xxx)
- Reseller: $99/month (price_yyy)
- Enterprise: Contact sales

**Features Needed:**

1. **Checkout Flow:**
   - User selects plan on /pricing page
   - Redirects to Stripe Checkout
   - Returns to /dashboard on success

2. **Subscription Management:**
   - Display current plan in settings
   - Cancel subscription button
   - Update payment method

3. **Webhook Handlers:**
   - customer.subscription.created
   - customer.subscription.updated
   - customer.subscription.deleted
   - invoice.payment_succeeded
   - invoice.payment_failed

4. **Database:**
   - Store: stripeCustomerId, stripeSubscriptionId, plan, status
   - Update on webhook events

5. **Access Control:**
   - Gate features by subscription status
   - Show upgrade prompts for inactive subs

**Provide:**
1. Stripe API configuration
2. Checkout session API route
3. Webhook handler API route
4. UI components for plan selection
5. Settings page for subscription management
6. Test mode setup instructions
```

Debugging & Troubleshooting

Effective Debug Prompts:

1. Runtime Errors:

~~~
**Error:** [FULL_ERROR_MESSAGE]

**File:** [FILE_PATH:LINE_NUMBER]

**Context:**
This error occurs when [DESCRIBE_SITUATION].

**Code Snippet:**
```typescript
[PASTE_RELEVANT_CODE]
~~~

Environment:
- Node version: [VERSION]
- Next.js version: [VERSION]
- Database: [TYPE]

What’s causing this and how do I fix it?

~~~
**2. Build Failures:**
```markdown
**Build Error:**
~~~

[PASTE_BUILD_OUTPUT]

```
**What I tried:**
1. [ACTION_1] - [RESULT]
2. [ACTION_2] - [RESULT]

**Recent Changes:**
- [CHANGE_1]
- [CHANGE_2]

Help me resolve this build error.
```

3. Logic Issues:

```
**Expected:** [DESCRIBE_EXPECTED_BEHAVIOR]
**Actual:** [DESCRIBE_ACTUAL_BEHAVIOR]

**Relevant Code:**
- Component: [FILE_PATH]
- API Route: [FILE_PATH]
- Database Query: [DESCRIBE_QUERY]

**Steps to Reproduce:**
1. [STEP_1]
2. [STEP_2]
3. [OBSERVE_ISSUE]

Diagnose the issue and provide a fix.
```

Common Pitfalls to Avoid

❌ Avoid These Prompt Patterns:

1. Too Vague:

```
# Bad:
"Build a dashboard for my app"

# Good:
"Build a dashboard with:
- Summary cards showing total items, pending reviews, flagged items
- Recent activity list
- Quick action buttons for 'Add Item' and 'View Reports'
- Role-based navigation (admin sees extra menu items)"
```

2. Too Much at Once:

```
# Bad:
"Build the entire app with all features, integrations,
Stripe, S3, n8n, AI, admin panel, and reports"

# Good:
"Build Phase 1: Foundation (marketing + auth + dashboard)"
"Build Phase 2: Core features (items CRUD + file uploads)"
"Build Phase 3: Integrations (Stripe + n8n webhooks)"
```

3. Unclear Requirements:

```
# Bad:
"Add authentication"

# Good:
"Add NextAuth.js authentication with:
- Email/password login
- User roles: admin, user, guest
- Protected routes: /dashboard/* requires auth
- Session expires after 7 days"
```

4. Missing Context:

```
# Bad:
"Fix the bug"

# Good:
"Fix the authentication bug:
- Error: 'Cannot read property id of undefined'
- Occurs: When user logs out and tries to access /dashboard
- File: app/(dashboard)/layout.tsx line 42
- Expected: Redirect to /auth/login"
```

✅ Best Practices Summary:

1. **Be Specific:** Provide exact requirements, not vague goals
2. **Break It Down:** Split large projects into phases

3. **Include Context:** Share relevant code, errors, and environment details
4. **Test Iteratively:** Build, test, refine, repeat
5. **Document as You Go:** Ask for clear documentation with each feature
6. **Plan Database First:** Design schema before building features
7. **Security First:** Always ask for authentication, validation, RBAC
8. **Think Production:** Request error handling, logging, monitoring

---

Complete Example: Full Project Prompt

Conversation 1: Phase 1 Foundation

```
I'm building Genesis Provenance - an AI-powered provenance vault for luxury assets
(watches, handbags, jewelry, art, collectibles).

**Business Model:**
- Collectors register and document luxury items
- Resellers/dealers authenticate inventory
- Partners (insurers, lenders) access provenance data

**Phase 1 Goals - Build deployable foundation:**

1. **Marketing Site** (genesisprovenance.com)
   - Pages: Home, Product, How It Works, Pricing, Use Cases, Security, About, Contact
   - Professional design with navy/gold color scheme
   - Contact form saves to database

2. **Authentication**
   - NextAuth.js with email/password
   - Roles: collector, reseller, partner, admin
   - Multi-tenant (users belong to organizations)

3. **Dashboard**
   - Protected at app.genesisprovenance.com
   - Sidebar navigation
   - Dashboard home with summary cards (placeholder data)
   - Settings page (profile + password change)
   - Admin console (users/orgs list)

4. **Database Schema**
   - Tables: users, organizations, items, item_categories, contact_submissions
   - Proper relations, indexes
   - Seed with default admin user and categories

5. **Deployment Setup**
   - README with setup instructions
   - .env.example
   - Vercel deployment guide
   - DNS setup for GoDaddy

**Tech Stack:**
- Next.js 14 App Router + TypeScript + Tailwind CSS
- PostgreSQL + Prisma ORM
- NextAuth.js
- Vercel hosting

**Defer to Phase 2:**
- Asset onboarding wizard
- File uploads (S3)
- Item detail pages
- Provenance timeline

**Defer to Phase 3:**
- n8n webhooks
- AI integration
- Certificate generation

**Defer to Phase 4:**
- Stripe subscriptions

Build Phase 1 with production-ready code, proper error handling, and comprehensive
documentation. I have intermediate coding skills.
```

Conversation 2: Phase 2 Core Features

```
Continuing Genesis Provenance from Phase 1.

Phase 2 Goals:

1. **S3 Cloud Storage**
   - Initialize S3 for file uploads
   - Create lib/s3.ts utilities (upload, download, delete)
   - Environment variables for AWS credentials

2. **Database Extensions**
   - Add tables: media_assets, provenance_events, certificates
   - New enums for media types, event types
   - Proper relations to items table

3. **Asset Onboarding Wizard**
   - Multi-step form (4 steps):
     * Step 1: Basic details (category, brand, model, serial, date)
     * Step 2: Upload photos (multiple)
     * Step 3: Upload documents (optional)
     * Step 4: Review and submit
   - Files upload to S3
   - Save item + media to database
   - Redirect to item detail page

4. **Items List Page** (/vault)
   - Grid/list view toggle
   - Filter by category, status
   - Search by brand/model
   - Pagination (20 per page)
   - "Add Asset" button

5. **Item Detail Page** (/vault/[id])
   - Display all metadata
   - Photo gallery (lightbox)
   - Documents list with download links
   - Provenance timeline
   - Status badge
   - Edit/Delete buttons

Update existing "My Vault" page to show actual items list.
Provide S3 setup instructions and test the full flow.
```

Conversation 3: Phase 3 Integrations

```
Phase 3: Add n8n and AI integrations.

1. **n8n Webhooks**

   **Outgoing (App → n8n):**
   - Trigger: When asset created
   - URL: N8N_WEBHOOK_URL/asset-created
   - Payload: item metadata, S3 keys for photos/docs

   **Incoming (n8n → App):**
   - POST /api/webhooks/provenance-update
   - Body: itemId, status, riskScore, analysis
   - Updates item status and creates provenance event

2. **AI Service Abstraction**
   - lib/ai-service.ts with:
     * analyzeAsset(itemId): Fetches item + media, prepares payload
     * Mock implementation for testing
     * Clear TODO for actual API integration
   - Expected input/output JSON schemas

3. **Certificate Generation**
   - Generate HTML certificate view
   - Shareable public link with secure token
   - Display: item details, provenance summary, risk score
   - TODO: PDF generation (Phase 4)

4. **Virtual Roles Config**
   - Create /config/agents/ folder
   - Agent files: CEO, CTO, ProvenanceAnalyst, FraudAgent
   - Each file: system prompt, input schema, output schema, examples

5. **Documentation**
   - n8n workflow setup guide
   - AI API integration guide
   - Webhook testing with curl examples

Provide working webhook code and test with mock payloads.
```

Quick Reference: Common Commands

Database Commands:

```
# Generate Prisma client
yarn prisma generate

# Create migration
yarn prisma migrate dev --name description_of_changes

# Push schema without migration
yarn prisma db push

# Seed database
yarn prisma db seed

# Open Prisma Studio
yarn prisma studio

# Reset database (CAUTION)
yarn prisma migrate reset
```

Development:

```
# Install dependencies
yarn install

# Run dev server
yarn dev

# Build for production
yarn build

# Start production server
yarn start

# Type check
yarn tsc --noEmit

# Lint
yarn lint
```

Vercel Deployment:

```
# Install Vercel CLI
npm i -g vercel

# Login
vercel login

# Deploy
vercel

# Deploy to production
vercel --prod

# Pull environment variables
vercel env pull .env.local
```

Final Tips for Success

1. **Start Simple:** Get Phase 1 working perfectly before adding complexity
2. **Test Frequently:** Deploy and test after each major feature
3. **Document Everything:** Ask for clear README, setup guides, API docs
4. **Security First:** Authentication, validation, RBAC from day one
5. **Plan Ahead:** Think about Phase 2-4 requirements while building Phase 1
6. **Git Commits:** Commit after each successful feature addition
7. **Environment Variables:** Keep secrets in .env, never in code
8. **Error Handling:** Always ask for try/catch, error messages, logging
9. **User Experience:** Loading states, error messages, empty states
10. **Production Ready:** Build for scale, not just MVP

Remember: DeepAgent works best with clear, specific, phased requirements. Break big projects into conversations, test iteratively, and build production-ready code from the start.

Last Updated: November 29, 2025
For: Genesis Provenance Development