

Análisis Técnico Completo - Aplicación de Tarot

Fecha: 18 de Octubre, 2025

Proyecto: Tarot Flavia

Objetivo: Aplicación web de tarot con funcionalidades básicas, escalable y mantenible

📋 Índice

1. Resumen Ejecutivo
2. Análisis de Historias de Usuario
3. Arquitectura Propuesta
4. Stack Tecnológico
5. Base de Datos
6. Estructura del Monorepo
7. Estrategia de Deployment
8. Roadmap de Desarrollo
9. Consideraciones de Escalabilidad
10. Estimaciones y Prioridades

1. Resumen Ejecutivo

☑ Estado Actual

- **Backend:** NestJS avanzado (~40% completado)
- **Frontend:** Estructura mínima (Vite configurado)
- **Monorepo:** Ya configurado con workspaces de npm
- **Base de Datos:** PostgreSQL (parcialmente configurada)

⌚ Recomendaciones Principales

MANTENER:

- NestJS para el backend (excelente elección)
- PostgreSQL como base de datos
- Estructura de monorepo actual
- TypeScript en todo el stack

CAMBIOS SUGERIDOS:

- Frontend: **Next.js** en lugar de Vite puro (mejor para SEO, routing simple, y desarrollo rápido con genIA)
- Separar módulos de negocio (tarot, oráculo, rituales) para mejor escalabilidad
- Implementar migraciones de base de datos (TypeORM) en lugar de `synchronize: true`
- Agregar validación de variables de entorno más robusta

2. Análisis de Historias de Usuario

2.1 Evaluación por Módulo

➊ Módulo de Autenticación (HU 1-3) - BIEN DEFINIDO

Estado: Backend ~80% implementado

Complejidad: Media

Prioridad: ALTA (requisito base)

Observaciones:

- Registro e inicio de sesión están correctamente especificados
- ⚠ Recuperación de contraseña requiere servicio de email (no contemplado)
- 💡 **Sugerencia:** Agregar HU para verificación de email opcional

Recomendaciones:

- Usar JWT (ya implementado)
- Agregar refresh tokens para mejor UX
- Implementar rate limiting en login
- Email opcional para MVP, mandatorio para producción

➋ Módulo de Interpretación de Cartas (HU 4-6) - BIEN DEFINIDO

Estado: Backend ~60% implementado

Complejidad: Alta (integración IA)

Prioridad: ALTA (funcionalidad core)

Observaciones:

- Flujo claro: selección → interpretación → resultado
- ⚠ Limitación a 3 cartas puede ser restrictiva
- 💡 **Sugerencia:** Permitir diferentes tipos de tiradas (1, 3, 5, 10 cartas)

Recomendaciones:

- Implementar sistema de "spreads" (tiradas predefinidas)
- Cachear interpretaciones frecuentes para reducir costos de IA
- Agregar HU para historial de interpretaciones

➌ Módulo de Consulta de Oráculo (HU 7-9) - NECESITA REFINAMIENTO

Estado: No implementado
Complejidad: Media
Prioridad: MEDIA

Observaciones:

- ⚠ Similar al módulo anterior pero con pregunta abierta
- ⚠ Sin contexto de cartas, ¿cómo funciona el oráculo?
- 💡 **Sugerencia:** Definir si usa cartas, péndulo, o solo IA

Recomendaciones:

- Opción A:** Oráculo con carta única aleatoria + pregunta
 - Opción B:** Respuestas de IA puras (más simple pero menos "místico")
 - Opción C:** Combinar con método de péndulo virtual
 - Limitar preguntas diarias para usuarios gratuitos
-

➊ Módulo de Rituales y Amuletos (HU 10-11) - FALTA DETALLE

Estado: No implementado
Complejidad: Baja
Prioridad: BAJA (puede ser fase 2)

Observaciones:

- ⚠ Contenido estático, requiere CMS o admin panel
- ⚠ No especifica categorías (amor, dinero, protección, etc.)
- 💡 **Sugerencia:** Agregar búsqueda y filtros

Recomendaciones:

- CRUD simple en admin
 - Categorización por tipo y propósito
 - Sistema de favoritos por usuario (requiere auth)
 - Agregar imágenes/iconos (CDN necesario)
-

➋ Módulo de Servicios Pagos (HU 12-13) - REQUIERE EXPANSIÓN

Estado: No implementado
Complejidad: Media-Alta (pagos + email)
Prioridad: MEDIA (monetización)

Observaciones:

- ⚠ Falta definir método de pago (formulario != pago procesado)
- ⚠ No contempla confirmación/coordinación de sesiones
- ⚠ Requiere integración de pasarela de pago
- 💡 **Sugerencia:** Simplificar para MVP

Recomendaciones MVP:

- Fase 1: Solo formulario + email manual (sin pago online)
- Fase 2: Integrar Stripe/MercadoPago
- Agregar HU para:
 - Calendario de disponibilidad
 - Confirmación de citas
 - Sistema de notificaciones

● Módulo de Administración (HU 14-16) - BIEN DEFINIDO

Estado: Parcialmente implementado

Complejidad: Media

Prioridad: MEDIA (puede ser manual al inicio)

Observaciones:

- ☑ Funcionalidades estándar de admin
- ⚠ Falta especificar roles y permisos
- 💡 **Sugerencia:** Usar librería de admin panel

Recomendaciones:

- Implementar roles: Admin, Moderador, Usuario
- Para MVP: Panel simple con React Admin o similar
- Para escalabilidad: Implementar RBAC (Role-Based Access Control)

● Implementación de IA (HU 17-19) - BIEN PLANTEADO

Estado: Parcialmente implementado

Complejidad: Alta

Prioridad: ALTA

Observaciones:

- ☑ OpenAI es una buena elección
- ⚠ Costos pueden escalar rápidamente
- 💡 **Sugerencia:** Implementar límites y cache

Recomendaciones:

- Usar GPT-4o-mini para reducir costos (suficiente para el caso de uso)
 - Implementar cache Redis para interpretaciones repetidas
 - Límites por usuario:
 - Gratuito: 3 consultas/día
 - Premium: ilimitado
 - Usar system prompts optimizados para tarot
-

2.2 Historias de Usuario Faltantes (Críticas)

HU Adicionales Recomendadas

Seguridad y Autenticación

20. **Verificación de Email**

Como usuario, quiero verificar mi email para activar mi cuenta.

21. **Cierre de sesión**

Como usuario, quiero cerrar sesión en todos mis dispositivos.

Experiencia de Usuario

22. **Historial de lecturas**

Como usuario, quiero ver todas mis interpretaciones pasadas.

23. **Compartir resultados**

Como usuario, quiero compartir mi lectura en redes sociales.

24. **Perfil de usuario**

Como usuario, quiero editar mi perfil y preferencias.

Límites y Monetización

25. **Límites de uso gratuito**

Como usuario gratuito, quiero saber cuántas consultas me quedan.

26. **Plan premium**

Como usuario, quiero actualizar a un plan premium para consultas ilimitadas.

Administración

27. **Dashboard de estadísticas**

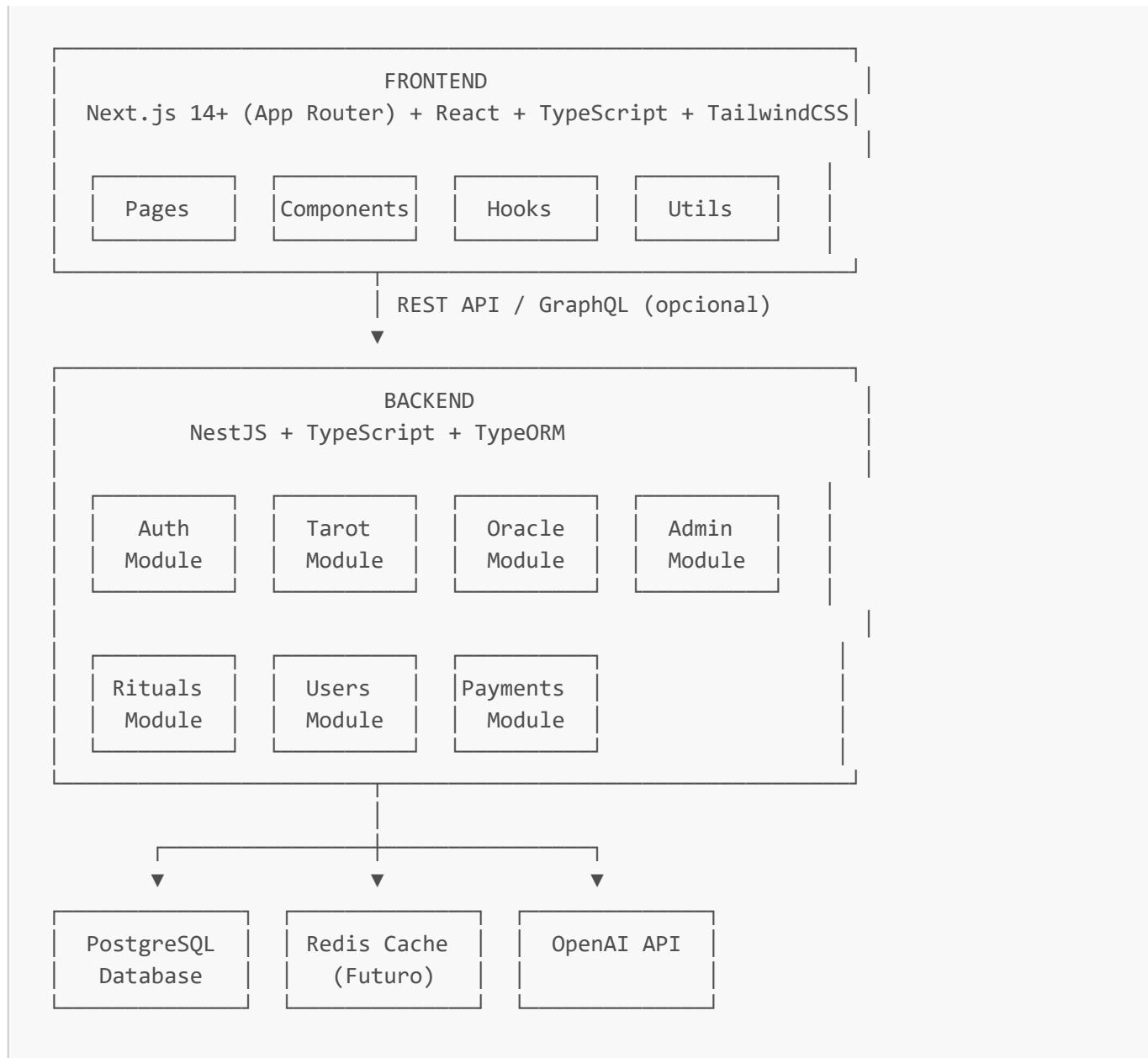
Como admin, quiero ver métricas de uso de la aplicación.

28. **Gestión de contenido de rituales**

Como admin, quiero agregar/editar rituales con editor rico.

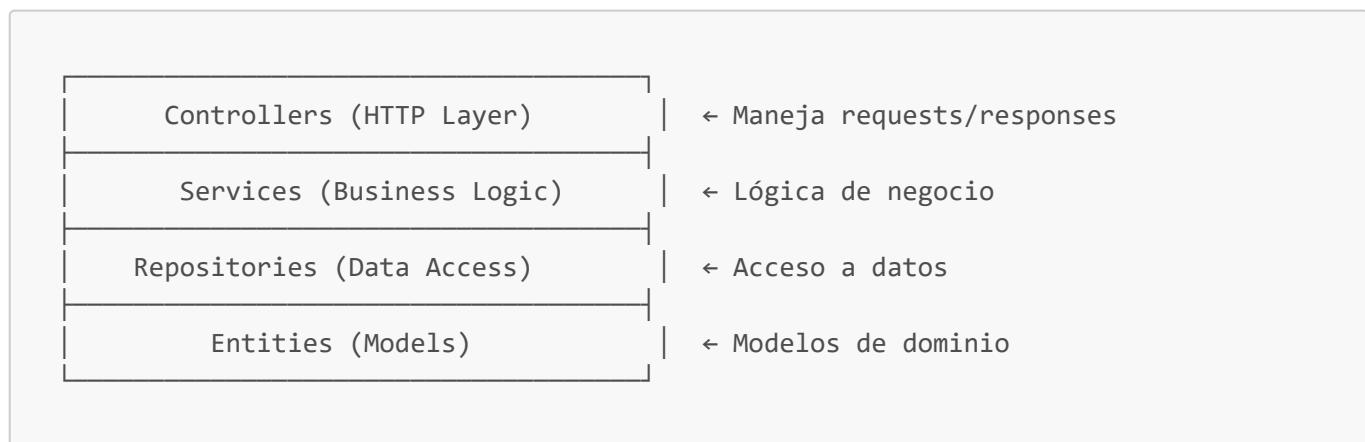
3. Arquitectura Propuesta

3.1 Arquitectura General



3.2 Patrón de Arquitectura

Backend: Arquitectura en Capas (Layered Architecture)



Ventajas para tu caso:

- Separación clara de responsabilidades

- Fácil de testear
 - NestJS lo soporta nativamente
 - Escalable a microservicios si es necesario
-

4. Stack Tecnológico

4.1 Backend (MANTENER con mejoras)

Componente	Tecnología	Justificación
Framework	NestJS 11	<input checked="" type="checkbox"/> Ya implementado, excelente para APIs escalables, TypeScript nativo, modular
Runtime	Node.js 20 LTS	<input checked="" type="checkbox"/> Soporte a largo plazo, mejor rendimiento
Lenguaje	TypeScript 5.7	<input checked="" type="checkbox"/> Type safety, mejor DX con genIA
ORM	TypeORM 0.3	<input checked="" type="checkbox"/> Ya implementado, buena integración con NestJS
Validación	class-validator + class-transformer	<input checked="" type="checkbox"/> Ya implementado, estándar en NestJS
Autenticación	JWT + Passport	<input checked="" type="checkbox"/> Ya implementado, seguro y stateless
API Docs	Swagger/OpenAPI	<input checked="" type="checkbox"/> Ya configurado, esencial para frontend
Testing	Jest + Supertest	<input checked="" type="checkbox"/> Ya configurado, estándar de la industria

4.2 Frontend (CAMBIO RECOMENDADO)

Componente	Tecnología Actual	Recomendación	Justificación
Framework	Vite (vacío)	Next.js 14+ (App Router)	<input checked="" type="checkbox"/> SEO crucial para tarot, <input checked="" type="checkbox"/> SSR/ISR, <input checked="" type="checkbox"/> Routing simple, <input checked="" type="checkbox"/> Fácil para genIA
UI Library	-	React 18	<input checked="" type="checkbox"/> Estándar, muchos componentes disponibles
Lenguaje	-	TypeScript	<input checked="" type="checkbox"/> Consistencia con backend, mejor DX
Styling	-	TailwindCSS	<input checked="" type="checkbox"/> Rápido desarrollo, <input checked="" type="checkbox"/> Fácil para genIA, <input checked="" type="checkbox"/> Utilities-first
UI Components	-	shadcn/ui	<input checked="" type="checkbox"/> Componentes accesibles, <input checked="" type="checkbox"/> Customizables, <input checked="" type="checkbox"/> No es librería (copias el código)
Forms	-	React Hook Form	<input checked="" type="checkbox"/> Mejor rendimiento, validación simple
State	-	Zustand (opcional)	<input checked="" type="checkbox"/> Simple, solo si necesitas estado global complejo
HTTP Client	-	Axios / Fetch	<input checked="" type="checkbox"/> Simple, suficiente para REST

Componente	Tecnología Actual	Recomendación	Justificación
Icons	-	Lucide React	<input checked="" type="checkbox"/> Moderno, tree-shakeable

4.3 Base de Datos

Componente	Tecnología	Justificación
DBMS	PostgreSQL 16	<input checked="" type="checkbox"/> Ya configurado, robusto, JSON support, buenas extensiones
Migraciones	TypeORM Migrations	<input checked="" type="checkbox"/> Control de versiones de DB, deployment seguro
Seeders	TypeORM Seeding	<input checked="" type="checkbox"/> Datos iniciales (cartas de tarot)
Cache (futuro)	Redis	Para interpretaciones y rate limiting

4.4 Integraciones Externas

Servicio	Propósito	Prioridad
OpenAI API (GPT-4o-mini)	Interpretaciones y oráculo	ALTA
Resend / SendGrid	Emails transaccionales	MEDIA
Stripe / MercadoPago	Pagos (servicios premium)	BAJA (Fase 2)
Cloudinary / S3	Almacenamiento de imágenes	BAJA (si se necesitan uploads)

4.5 DevOps & Tooling

Componente	Tecnología	Justificación
Monorepo	npm workspaces	<input checked="" type="checkbox"/> Ya configurado, simple, sin dependencias extras
Linting	ESLint	<input checked="" type="checkbox"/> Ya configurado
Formatting	Prettier	<input checked="" type="checkbox"/> Ya configurado
Git Hooks	Husky + lint-staged	Recomendado para calidad de código
CI/CD	GitHub Actions	<input checked="" type="checkbox"/> Gratis, integrado con GitHub
Containerización	Docker + Docker Compose	<input checked="" type="checkbox"/> Desarrollo local consistente

5. Base de Datos

5.1 Modelo de Datos Propuesto

```
-- USUARIOS Y AUTENTICACIÓN
users
- id (PK)
- email (unique)
- password_hash
- name
- role (enum: user, admin)
- is_verified
- plan (enum: free, premium)
- created_at
- updated_at

-- CARTAS DE TAROT
tarot_cards
- id (PK)
- name
- arcana (enum: major, minor)
- suit (enum: cups, pentacles, swords, wands, null)
- number
- upright_meaning (text)
- reversed_meaning (text)
- image_url
- keywords (text[])
- created_at
- updated_at

-- MAZOS (DECKS)
tarot_decks
- id (PK)
- name
- description
- is_default
- created_at
- updated_at

-- TIPOS DE TIRADAS
tarot_spreads
- id (PK)
- name (ej: "Cruz Celta", "3 Cartas", etc.)
- description
- positions (jsonb) -- [{ position: 1, meaning: "Pasado" }, ...]
- card_count
- is_active
- created_at
- updated_at

-- LECTURAS/INTERPRETACIONES
tarot_readings
- id (PK)
- user_id (FK → users, nullable para anónimos)
- spread_id (FK → tarot_spreads)
- question (text, nullable)
- created_at
```

```
- updated_at
- shared_token (unique, para compartir)

-- CARTAS EN UNA LECTURA
reading_cards
- id (PK)
- reading_id (FK → tarot_readings)
- card_id (FK → tarot_cards)
- position
- is_reversed
- created_at

-- INTERPRETACIONES GENERADAS
tarot_interpretations
- id (PK)
- reading_id (FK → tarot_readings)
- interpretation (text)
- ai_model_used
- tokens_used
- created_at

-- CONSULTAS AL ORÁCULO
oracle_queries
- id (PK)
- user_id (FK → users, nullable)
- question (text)
- answer (text)
- ai_model_used
- created_at

-- RITUALES Y AMULETOS
rituals
- id (PK)
- title
- description (text)
- category (enum: love, money, protection, health, etc.)
- steps (jsonb)
- materials (text[])
- image_url
- created_at
- updated_at

-- FAVORITOS DE RITUALES
user_favorite_rituals
- id (PK)
- user_id (FK → users)
- ritual_id (FK → rituals)
- created_at
- UNIQUE (user_id, ritual_id)

-- SOLICITUDES DE SERVICIOS PAGOS
service_requests
- id (PK)
- user_id (FK → users, nullable)
```

```

- service_type (enum: energy_cleaning, hebrew_pendulum)
- name
- email
- phone
- message (text)
- status (enum: pending, contacted, confirmed, completed, cancelled)
- created_at
- updated_at

-- LÍMITES DE USO (Rate Limiting)
usage_limits
- id (PK)
- user_id (FK → users)
- feature (enum: tarot_reading, oracle_query)
- count
- date (date)
- created_at
- UNIQUE (user_id, feature, date)

```

5.2 Índices Recomendados

```

-- Performance indexes
CREATE INDEX idx_readings_user_id ON tarot_readings(user_id);
CREATE INDEX idx_readings_created_at ON tarot_readings(created_at DESC);
CREATE INDEX idx_readings_shared_token ON tarot_readings(shared_token);
CREATE INDEX idx_oracle_user_id ON oracle_queries(user_id);
CREATE INDEX idx_usage_limits_lookup ON usage_limits(user_id, feature, date);
CREATE INDEX idx_service_requests_status ON service_requests(status);

```

5.3 Estrategia de Migraciones

```

// Ejemplo de migración
export class InitialSchema1697000000000 implements MigrationInterface {
    public async up(queryRunner: QueryRunner): Promise<void> {
        // Crear tablas
    }

    public async down(queryRunner: QueryRunner): Promise<void> {
        // Rollback
    }
}

```

Comandos:

```
# Generar migración
npm run migration:generate -- -n MigrationName
```

```
# Ejecutar migraciones
npm run migration:run

# Revertir última migración
npm run migration:revert
```

6. Estructura del Monorepo

6.1 Estructura Propuesta

```
tarot/
  └── .github/
      └── workflows/
          ├── backend-ci.yml
          ├── frontend-ci.yml
          └── deploy.yml

  └── backend/
      └── tarot-app/
          ├── src/
              ├── modules/
                  ├── auth/
                  ├── users/
                  ├── tarot/
                      ├── cards/
                      ├── decks/
                      ├── readings/
                      ├── interpretations/
                      └── spreads/
                  ├── oracle/
                  ├── rituals/
                  ├── services/
                  └── admin/
              ├── common/
                  ├── decorators/
                  ├── filters/
                  ├── guards/
                  ├── interceptors/
                  ├── pipes/
                  └── utils/
              ├── config/
              ├── database/
                  ├── migrations/
                  └── seeds/
              ├── app.module.ts
              └── main.ts
          └── test/
          ├── .env.example
          ├── docker-compose.yml
          └── Dockerfile
```

```
    └── package.json

  ├── frontend/
  │   ├── public/
  │   │   └── images/
  │   │       └── cards/
  │   └── src/
  │       ├── app/
  │       │   ├── (auth)/
  │       │   │   ├── login/
  │       │   │   ├── register/
  │       │   │   └── forgot-password/
  │       │   ├── (main)/
  │       │   │   ├── tarot/
  │       │   │   ├── oracle/
  │       │   │   ├── rituals/
  │       │   │   └── profile/
  │       │   ├── (admin)/
  │       │   │   └── dashboard/
  │       │   ├── api/ (optional API routes)
  │       │   └── layout.tsx
  │       └── page.tsx
  ├── components/
  │   ├── ui/ (shadcn/ui)
  │   ├── tarot/
  │   ├── oracle/
  │   └── shared/
  ├── lib/
  │   ├── api/
  │   ├── hooks/
  │   ├── utils/
  │   └── validations/
  ├── types/
  └── styles/

  ├── .env.local.example
  ├── next.config.js
  ├── tailwind.config.js
  └── package.json

  └── shared/ (opcional, para tipos compartidos)
      └── types/
          ├── user.types.ts
          ├── tarot.types.ts
          └── api.types.ts

  └── docs/
      ├── project_backlog.md
      ├── ANALISIS_TECNICO.md (este archivo)
      ├── API_DOCUMENTATION.md
      └── DEPLOYMENT.md

  └── .gitignore
  └── .prettierrc
  └── .eslintrc.js
```

```
└── package.json (root)
└── docker-compose.yml (desarrollo completo)
└── README.md
```

6.2 Scripts del Monorepo

```
// package.json (root)
{
  "name": "tarot-monorepo",
  "private": true,
  "workspaces": [
    "backend/tarot-app",
    "frontend"
  ],
  "scripts": {
    "dev": "concurrently \"npm run dev -w backend/tarot-app\" \"npm run dev -w frontend\"",
    "dev:backend": "npm run start:dev -w backend/tarot-app",
    "dev:frontend": "npm run dev -w frontend",

    "build": "npm run build --workspaces",
    "build:backend": "npm run build -w backend/tarot-app",
    "build:frontend": "npm run build -w frontend",

    "test": "npm run test --workspaces",
    "test:backend": "npm run test -w backend/tarot-app",
    "test:frontend": "npm run test -w frontend",

    "lint": "npm run lint --workspaces",
    "format": "prettier --write \"**/*.{ts,tsx,js,jsx,json,md}\"",

    "db:migrate": "npm run migration:run -w backend/tarot-app",
    "db:seed": "npm run seed -w backend/tarot-app",

    "docker:up": "docker-compose up -d",
    "docker:down": "docker-compose down",
    "docker:logs": "docker-compose logs -f"
  },
  "devDependencies": {
    "concurrently": "^8.2.2",
    "prettier": "^3.4.2"
  }
}
```

7. Estrategia de Deployment

7.1 Opciones de Hosting

Opción A: RECOMENDADA (Fácil y Económica)

Componente	Plataforma	Costo Aprox.	Justificación
Frontend	Vercel	\$0 - \$20/mes	<input checked="" type="checkbox"/> Gratis para hobby, <input checked="" type="checkbox"/> Next.js optimizado, <input checked="" type="checkbox"/> Deploy automático, <input checked="" type="checkbox"/> CDN global
Backend	Railway / Render	\$5 - \$20/mes	<input checked="" type="checkbox"/> Fácil setup, <input checked="" type="checkbox"/> Postgres incluido, <input checked="" type="checkbox"/> Auto-deploy
Database	Supabase / Railway	\$0 - \$10/mes	<input checked="" type="checkbox"/> Postgres gestionado, <input checked="" type="checkbox"/> Backups automáticos
Total	-	\$5 - \$50/mes	Escalable, fácil de mantener

Pros:

- Deploy en minutos
- Escalamiento automático
- HTTPS incluido
- Monitoreo básico
- Ideal para MVP y crecimiento inicial

Contras:

- Vendor lock-in parcial
- Costos suben con tráfico alto

Opción B: Media (Más Control)

Componente	Plataforma	Costo Aprox.
Frontend	Vercel / Netlify	\$0 - \$20/mes
Backend	DigitalOcean Droplet	\$12 - \$24/mes
Database	DigitalOcean Managed DB	\$15/mes
Total	-	\$27 - \$59/mes

Opción C: Avanzada (Máximo Control)

Componente	Plataforma	Costo Aprox.
Todo	AWS (ECS + RDS + CloudFront)	\$30 - \$100+/mes

Solo recomendado si:

- Tráfico muy alto esperado (>100k usuarios/mes)
- Requisitos de compliance específicos
- Necesitas control total de infraestructura

7.2 Estrategia de Deploy Recomendada

```
# .github/workflows/deploy.yml
name: Deploy

on:
  push:
    branches: [main, develop]

jobs:
  backend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '20'
      - run: npm ci
      - run: npm run build -w backend/tarot-app
      - run: npm run test -w backend/tarot-app
      # Deploy a Railway/Render

  frontend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '20'
      - run: npm ci
      - run: npm run build -w frontend
      # Vercel auto-deploy configurado
```

7.3 Variables de Entorno

```
# Backend (.env)
NODE_ENV=production
PORT=3000

# Database
POSTGRES_HOST=
POSTGRES_PORT=5432
POSTGRES_USER=
POSTGRES_PASSWORD=
POSTGRES_DB=tarot_production

# JWT
JWT_SECRET=
JWT_EXPIRES_IN=7d
```

```
# OpenAI
OPENAI_API_KEY=

# Email (opcional)
EMAIL_HOST=
EMAIL_PORT=
EMAIL_USER=
EMAIL_PASSWORD=

# Frontend
CORS_ORIGINS=https://tu-dominio.com

# Rate Limiting
RATE_LIMIT_TTL=60
RATE_LIMIT_MAX=100
```

```
# Frontend (.env.local)
NEXT_PUBLIC_API_URL=https://api.tu-dominio.com
NEXT_PUBLIC_APP_URL=https://tu-dominio.com
```

8. Roadmap de Desarrollo

8.1 Fase 1: MVP (4-6 semanas)

Sprint 1: Fundamentos (2 semanas)

- **Backend:**
 - Refactorizar estructura de módulos
 - Implementar migraciones de TypeORM
 - Completar módulo de autenticación (con refresh tokens)
 - Seeders para cartas de tarot (78 cartas estándar)
 - Sistema de validación robusto
- **Frontend:**
 - Setup de Next.js + TailwindCSS + shadcn/ui
 - Páginas de autenticación (login, register)
 - Layout principal
 - Configurar API client (axios)
- **DevOps:**
 - Docker Compose para desarrollo local
 - CI básico (linting + tests)

Sprint 2: Funcionalidad Core (2 semanas)

- **Backend:**
 - Módulo de lecturas (tarot readings)

- Integración con OpenAI para interpretaciones
- Sistema de spreads (tiradas predefinidas)
- Rate limiting básico
- **Frontend:**
 - Página de selección de tirada
 - Interfaz de selección de cartas (drag & drop o click)
 - Página de resultados/interpretación
 - Loader durante generación de IA
- **Testing:**
 - Tests unitarios críticos
 - Tests E2E básicos

Sprint 3: Oráculo y Pulido (1-2 semanas)

- **Backend:**
 - Módulo de oráculo
 - Sistema de límites de uso (free vs premium)
 - Historial de lecturas
- **Frontend:**
 - Página de oráculo
 - Historial de lecturas del usuario
 - Página de perfil básica
 - Responsive design
- **Deploy:**
 - Deploy a staging (Vercel + Railway)
 - Testing en producción

Entregables Fase 1:

- Autenticación funcional
- Lecturas de tarot con interpretación IA (al menos 2 spreads)
- Consulta de oráculo
- Historial de lecturas
- Límites de uso
- Aplicación deployada y accesible

8.2 Fase 2: Monetización y Contenido (3-4 semanas)

Sprint 4: Rituales y Admin (2 semanas)

- Módulo de rituales (CRUD completo)
- Panel de administración básico
- Sistema de favoritos
- Búsqueda y filtros de rituales

Sprint 5: Servicios Pagos (1-2 semanas)

- Formulario de solicitud de servicios
- Integración de email (Resend/SendGrid)
- Panel de admin para gestionar solicitudes
- (Opcional) Integración de pagos (Stripe/MercadoPago)

Entregables Fase 2:

- Sistema de rituales completo
 - Panel de admin funcional
 - Sistema de solicitud de servicios
 - Notificaciones por email
-

8.3 Fase 3: Optimización y Escala (2-3 semanas)

Sprint 6: Performance

- Implementar Redis para cache
- Optimizar queries de database
- CDN para imágenes
- Lazy loading en frontend
- Optimización de prompts de IA

Sprint 7: Features Premium

- Sistema de planes (free/premium)
- Dashboard de estadísticas
- Compartir lecturas (links públicos)
- Exportar lecturas a PDF

Entregables Fase 3:

- Aplicación optimizada (< 3s carga)
 - Sistema de planes implementado
 - Métricas y analytics
-

9. Consideraciones de Escalabilidad

9.1 Backend

Escalabilidad Horizontal

```
// Preparar para múltiples instancias
// 1. Sesiones stateless (JWT)
// 2. Cache compartido (Redis)
// 3. Database connection pooling

// config/database.ts
poolSize: parseInt(process.env.DB_POOL_SIZE || '10'),
```

Rate Limiting

```
// Implementar throttler
@ThrottlerGuard()
export class AppModule {}

// Límites por endpoint
@Throttle({ default: { limit: 3, ttl: 60000 } })
@Post('interpret')
```

Caching Strategy

```
// 1. Cache de interpretaciones frecuentes
// 2. Cache de cartas (raramente cambian)
// 3. Cache de rituales

@Injectable()
export class CacheService {
    async getOrSet(key: string, factory: () => Promise<any>, ttl = 3600) {
        // Redis implementation
    }
}
```

9.2 Frontend

Code Splitting

```
// next.config.js
const nextConfig = {
    experimental: {
        optimizeCss: true,
    },
};
```

Image Optimization

```
// Usar Next.js Image
<Image
    src="/cards/fool.jpg"
    width={300}
    height={500}
    alt="The Fool"
```

```
loading="lazy"
/>>
```

Static Generation

```
// Para páginas que no cambian
export async function generateStaticParams() {
  const rituals = await getAllRituals();
  return rituals.map(r => ({ id: r.id }));
}
```

9.3 Base de Datos

Índices

- Ya definidos en sección 5.2

Particionamiento (Futuro)

```
-- Partitionar lecturas por fecha si crecen mucho
CREATE TABLE tarot_readings_2025_q1 PARTITION OF tarot_readings
FOR VALUES FROM ('2025-01-01') TO ('2025-04-01');
```

Read Replicas

- Para reportes y analytics
- Separar lectura de escritura

9.4 Costos de IA

Estrategia de Optimización

```
// 1. Cache de interpretaciones idénticas
const cacheKey = `interpretation:${cardIds.sort().join('-')}:${spreadId}`;

// 2. Usar modelo más barato (GPT-4o-mini)
model: 'gpt-4o-mini',

// 3. Limitar tokens de respuesta
max_tokens: 500,

// 4. System prompt optimizado
const systemPrompt = `Eres un experto en tarot.
Responde en 3 párrafos máximo,
sin introducción ni despedida.`;
```

Estimación de costos:

- GPT-4o-mini: ~\$0.15 / 1M tokens input, ~\$0.60 / 1M tokens output
- Interpretación típica: ~800 tokens (input + output) = \$0.0006
- 1000 interpretaciones/mes = \$0.60
- 10,000 interpretaciones/mes = \$6

Con cache (80% hit rate):

- 10,000 interpretaciones = \$1.20 (ahorro del 80%)
-

10. Estimaciones y Prioridades

10.1 Priorización de Features

Prioridad CRÍTICA (MVP mínimo)

1. Autenticación (login, register)
2. Selección de cartas (3 cartas básicas)
3. Interpretación con IA
4. Visualización de resultados

Prioridad ALTA (MVP completo)

5. Múltiples tipos de tiradas
6. Oráculo
7. Historial de lecturas
8. Límites de uso (rate limiting)
9. Recuperación de contraseña

Prioridad MEDIA (Fase 2)

10. Rituales y amuletos
11. Sistema de favoritos
12. Solicitud de servicios pagos
13. Panel de administración
14. Compartir lecturas

Prioridad BAJA (Fase 3+)

15. Planes premium con pagos
16. Estadísticas y analytics
17. Notificaciones push
18. App móvil nativa
19. Múltiples idiomas

10.2 Estimación de Esfuerzo

Feature	Complejidad	Tiempo Estimado	Dependencias
Auth completo	Media	3-5 días	-
Módulo Tarot	Alta	5-7 días	Auth
Integración IA	Media	2-3 días	Tarot
Frontend Next.js	Media	5-7 días	-
UI Lecturas	Media	3-5 días	Frontend, Tarot
Oráculo	Baja	2-3 días	IA
Rituales	Baja	2-3 días	Auth
Admin Panel	Media	3-5 días	Auth
Servicios Pagos	Media	3-4 días	Email
Optimizaciones	Media	3-5 días	Todo lo anterior

Total MVP (Fase 1): ~30-40 días de desarrollo **Total Fase 2:** ~10-15 días adicionales **Total Fase 3:** ~10-15 días adicionales

10.3 Riesgos y Mitigaciones

Riesgo	Probabilidad	Impacto	Mitigación
Costos de IA altos	Media	Alto	Cache agresivo, límites estrictos, monitoreo
Complejidad del frontend	Baja	Medio	Next.js simplifica, shadcn/ui acelera
Problemas de performance	Media	Medio	Monitoreo temprano, optimización progresiva
Scope creep	Alta	Alto	MVP estricto, fases bien definidas
Email delivery issues	Media	Bajo	Usar proveedor confiable (Resend)
Integración de pagos compleja	Media	Medio	Posponer a Fase 2, usar Stripe (bien documentado)

11. Recomendaciones Finales

Hacer (DO)

1. Empezar con MVP mínimo:

- Auth + 1 tipo de tirada + Interpretación IA
- Deploy rápido, iterar basado en feedback

2. Usar Next.js para frontend:

- SEO crítico para discoverability
- Desarrollo más rápido que Vite puro

- Mejor para trabajar con genIA

3. Implementar migraciones desde el inicio:

- `synchronize: false` en producción
- Control de versiones de DB

4. Monitorear costos de IA:

- Implementar logging de tokens
- Dashboard de uso por usuario
- Alertas de costos

5. Testing automatizado:

- Al menos tests unitarios de servicios críticos
- E2E de flujo principal (auth + lectura)

6. Documentación:

- Swagger para API
- README con setup instructions
- Architecture Decision Records (ADRs)

✗ No Hacer (DON'T)

1. No sobre-ingenierizar:

- No microservicios (al menos no al inicio)
- No GraphQL (REST es suficiente)
- No arquitecturas complejas "por si acaso"

2. No implementar todas las HU a la vez:

- Priorizar ruthlessly
- MVP primero, features después

3. No ignorar seguridad:

- Validación de inputs siempre
- Rate limiting desde el inicio
- Variables de entorno nunca en el código

4. No usar `synchronize: true` en producción:

- Usar migraciones
- Evitar pérdida de datos

5. No optimizar prematuramente:

- Primero que funcione, luego optimiza
- Pero: monitorea desde el inicio

12. Checklist de Inicio

Semana 1: Setup

- Revisar y aprobar este análisis técnico
- Decidir stack de frontend (recomendación: Next.js)
- Configurar Next.js en [/frontend](#)
- Refactorizar backend según estructura propuesta
- Configurar migraciones de TypeORM
- Crear seeders de cartas de tarot (78 cartas)
- Setup de Docker Compose para desarrollo
- Configurar variables de entorno
- Setup de CI/CD básico
- Crear cuentas en:
 - Vercel (frontend)
 - Railway/Render (backend)
 - OpenAI (API key)

Semana 2: Desarrollo MVP

- Completar módulo de autenticación
- Implementar selección de cartas
- Integrar OpenAI para interpretaciones
- Crear UI de lectura de tarot
- Implementar rate limiting

Semana 3-4: Testing y Deploy

- Tests críticos
- Deploy a staging
- Corrección de bugs
- Deploy a producción
- Documentación básica

13. Conclusión

Resumen de Decisiones Clave

Decisión	Recomendación	Justificación
Frontend Framework	Next.js 14+	SEO, routing simple, fácil desarrollo con genIA
UI Library	shadcn/ui + TailwindCSS	Rápido, customizable, moderno
Backend	NestJS (mantener)	<input checked="" type="checkbox"/> Ya avanzado, excelente arquitectura
Base de Datos	PostgreSQL (mantener)	<input checked="" type="checkbox"/> Ya configurado, robusto
Hosting	Vercel + Railway	Fácil, económico, escalable

Decisión	Recomendación	Justificación
IA	OpenAI GPT-4o-mini	Balance costo/calidad
Monorepo	npm workspaces (mantener)	Simple, suficiente

Próximos Pasos Inmediatos

1. **Discutir y aprobar** este análisis
2. **Decidir sobre frontend:** ¿Next.js o mantener Vite?
3. **Priorizar features:** ¿MVP es suficiente con Auth + Tarot?
4. **Setup de desarrollo:** Docker Compose + migraciones
5. **Empezar Sprint 1** del roadmap

Métricas de Éxito del MVP

- Usuario puede registrarse e iniciar sesión
- Usuario puede hacer una lectura de tarot (al menos 1 tipo)
- Interpretación IA se genera en < 10 segundos
- UI es responsive y funciona en móvil
- Aplicación deployada y accesible públicamente
- Costos de IA < \$10/mes para primeros 100 usuarios

Documento creado: 18 de Octubre, 2025

Versión: 1.0

Autor: Análisis técnico generado para proyecto Tarot Flavia