

Arquitectura Técnica Completa - Trading Plan System

1. Stack Tecnológico Recomendado

Frontend (Interfaz de Usuario)

FRONTEND

- HTML5 + CSS3 (Responsive Design)
- JavaScript ES6+ (Vanilla)
- WebComponents para modularidad
- LocalStorage para persistencia
- Fetch API para comunicación
- CSS Grid/Flexbox para layouts

Backend (API y Validación)

BACKEND

- Node.js + Express.js
- TypeScript para type safety
- Joi/Yup para validación de esquemas
- Winston para logging
- Rate limiting y CORS
- Swagger para documentación API

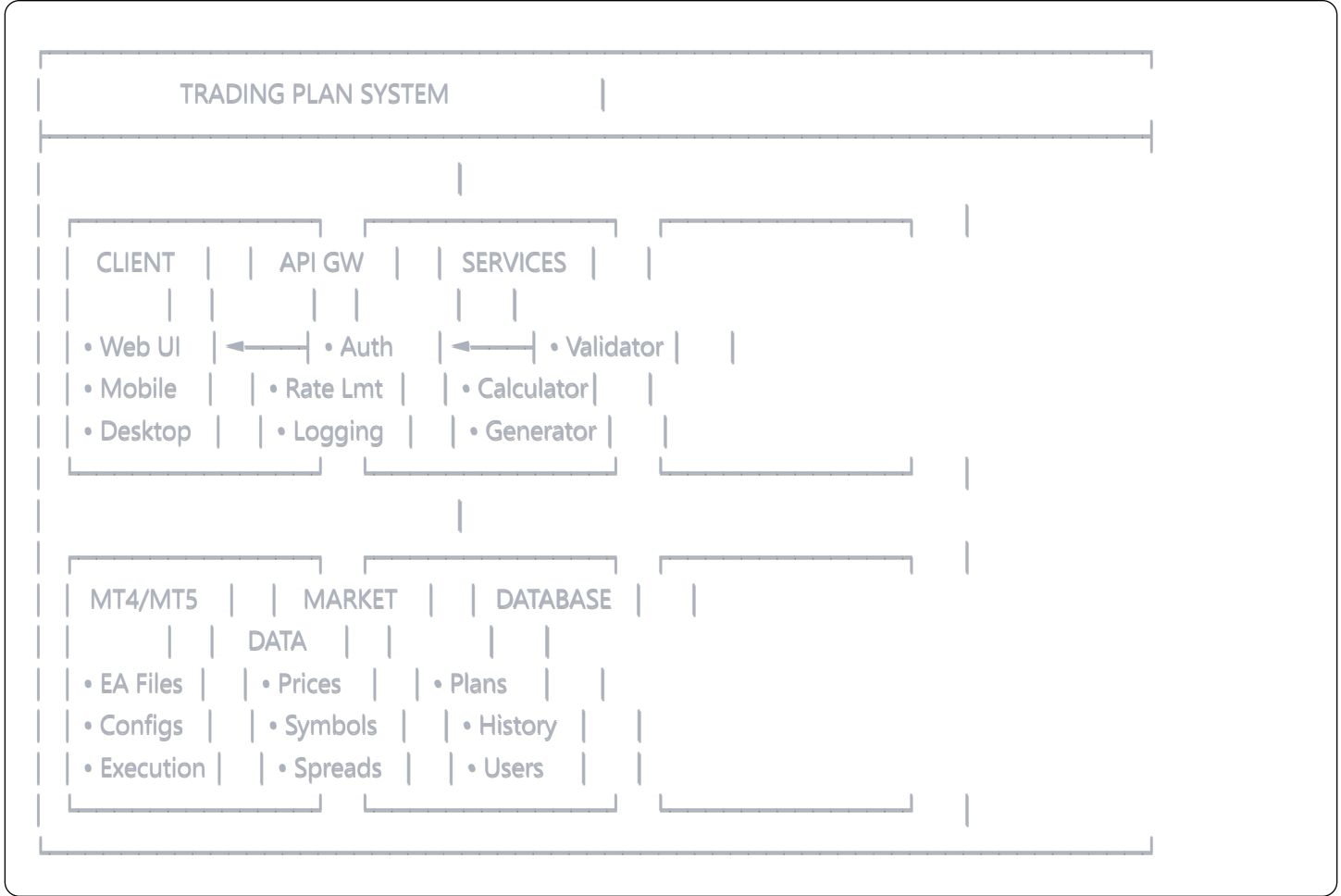
Base de Datos y Cache

DATA LAYER

- MongoDB para trading plans
- Redis para cache de precios
- InfluxDB para métricas de tiempo
- PostgreSQL para datos relacionales

2. Arquitectura del Sistema

Diagrama de Componentes



3. Modelos de Datos

Trading Plan Schema

typescript

```
interface TradingPlan {  
  id: string;  
  userId: string;  
  symbol: SymbolEnum;  
  direction: 'LONG' | 'SHORT';  
  entryType: 'LIMIT' | 'MARKET';  
  limitBehavior?: 'STANDARD' | 'BREAKOUT';  
  entryPrice?: number;  
  stopLoss: number;  
  takeProfits: number[];  
  riskManagement: {  
    type: 'PERCENTAGE' | 'FIXED_VOLUME';  
    value: number;  
  };  
  settings: {  
    slippage: number;  
    comment: string;  
  };  
  validation: ValidationResult;  
  createdAt: Date;  
  updatedAt: Date;  
  status: 'DRAFT' | 'VALIDATED' | 'EXECUTED';  
}
```

```
interface ValidationResult {  
  isValid: boolean;  
  errors: ValidationError[];  
  warnings: ValidationWarning[];  
  calculations: RiskCalculation;  
}
```

```
interface RiskCalculation {  
  lotSize: number;  
  riskAmount: number;  
  riskDistance: number;  
  partialLots?: number;  
  totalExposure: number;  
}
```

4. APIs y Endpoints

RESTful API Design

```
POST /api/v1/trading-plans      # Crear plan
GET  /api/v1/trading-plans      # Listar planes
GET  /api/v1/trading-plans/:id  # Obtener plan específico
PUT  /api/v1/trading-plans/:id  # Actualizar plan
DELETE /api/v1/trading-plans/:id # Eliminar plan

POST /api/v1/trading-plans/:id/validate # Validar plan
POST /api/v1/trading-plans/:id/export/mt4 # Exportar a MT4
POST /api/v1/trading-plans/:id/export/mt5 # Exportar a MT5

GET /api/v1/market-data/symbols # Obtener símbolos disponibles
GET /api/v1/market-data/prices/:symbol # Precios en tiempo real
GET /api/v1/market-data/specs/:symbol # Especificaciones del símbolo

POST /api/v1/risk/calculate # Calcular riesgo
POST /api/v1/validation/rules # Validar reglas de trading

GET /api/v1/user/preferences # Preferencias del usuario
PUT /api/v1/user/preferences # Actualizar preferencias
```

5. Servicios Core

ValidationService

typescript

```
class ValidationService {
  validateTradingPlan(plan: TradingPlan): ValidationResult
  validatePriceRelationships(plan: TradingPlan): boolean
  validateRiskParameters(plan: TradingPlan): boolean
  validateTakeProfitSequence(takeProfits: number[], direction: string): boolean
  validateSymbolAvailability(symbol: string): Promise<boolean>
  validateMarketHours(symbol: string): boolean
}
```

RiskCalculatorService

typescript

```
class RiskCalculatorService {
  calculateLotSize(plan: TradingPlan, accountEquity: number): number
  calculateRiskAmount(plan: TradingPlan, accountEquity: number): number
  calculatePartialLots(totalLots: number, tpCount: number): number
  validateLotSizes(lotSize: number, symbol: string): boolean
  getSymbolSpecifications(symbol: string): SymbolSpecs
}
```

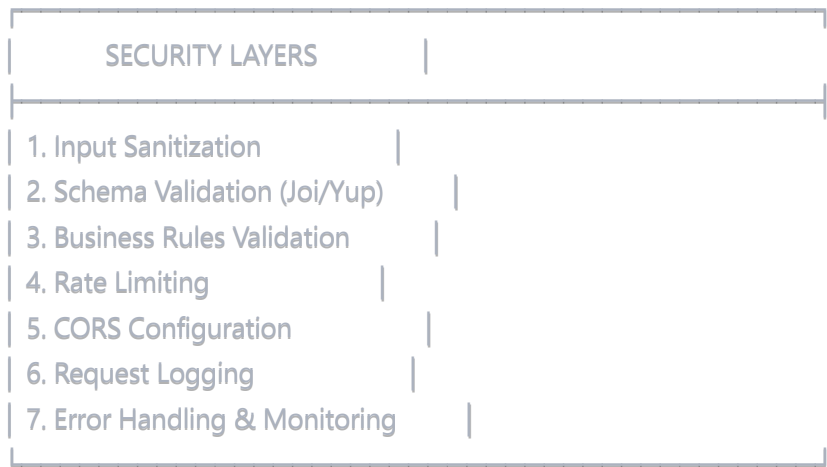
MT4GeneratorService

typescript

```
class MT4GeneratorService {
  generateInputParameters(plan: TradingPlan): string
  generateEAConfiguration(plan: TradingPlan): string
  generateCommentHeader(plan: TradingPlan): string
  validateMT4Compatibility(plan: TradingPlan): boolean
  exportToFile(content: string, format: 'mq4' | 'set'): Buffer
}
```

6. Seguridad y Validación

Capas de Seguridad



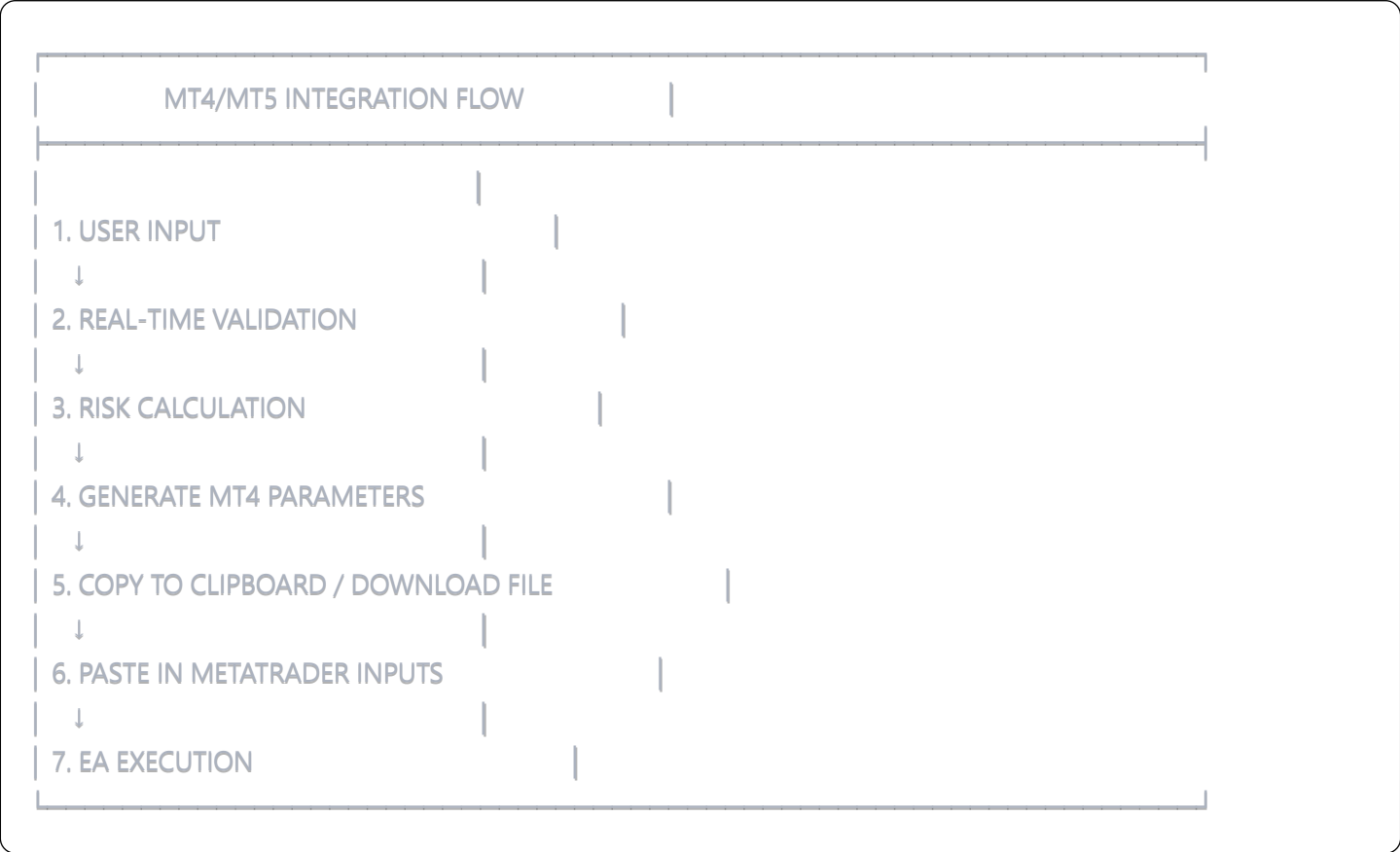
Reglas de Validación

typescript

```
const tradingPlanValidation = {
  symbol: Joi.string().valid(...VALID_SYMBOLS).required(),
  direction: Joi.string().valid('LONG', 'SHORT').required(),
  entryPrice: Joi.when('entryType', {
    is: 'LIMIT',
    then: Joi.number().positive().required(),
    otherwise: Joi.number().optional()
  }),
  stopLoss: Joi.number().positive().required(),
  riskPercent: Joi.number().min(0.1).max(100).when('totalVolume', {
    is: 0,
    then: Joi.required(),
    otherwise: Joi.optional()
  }),
  takeProfits: Joi.array().items(
    Joi.number().positive().optional()
  ).max(5)
};
```

7. Integración con MetaTrader

Flujo de Exportación



8. Monitoreo y Logging

Logging Strategy

```
typescript

interface LogEntry {
  timestamp: Date;
  level: 'INFO' | 'WARN' | 'ERROR' | 'DEBUG';
  service: string;
  function: string;
  line: number;
  message: string;
  metadata?: object;
  userId?: string;
  tradingPlanId?: string;
}

// Similar al sistema de logging del EA original
class Logger {
  info(message: string, metadata?: object): void
  warn(message: string, metadata?: object): void
  error(message: string, metadata?: object): void
  debug(message: string, metadata?: object): void
}
```

9. Performance y Escalabilidad

Optimizaciones

- **Client-side caching** de símbolos y especificaciones
- **Server-side caching** de precios de mercado con Redis
- **Lazy loading** de componentes UI
- **Debouncing** en validación en tiempo real
- **Connection pooling** para base de datos
- **CDN** para assets estáticos

Métricas de Performance

- Tiempo de respuesta de validación < 100ms
- Tiempo de generación de MT4 config < 50ms
- Disponibilidad del sistema > 99.9%
- Throughput: 1000+ validaciones por segundo

10. Deployment y DevOps

Containerización

dockerfile

Frontend Container

FROM nginx:alpine

COPY dist/ /usr/share/nginx/html/

COPY nginx.conf /etc/nginx/nginx.conf

Backend Container

FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm ci --only=production

COPY . .

EXPOSE 3000

CMD ["npm", "start"]

CI/CD Pipeline





yaml

stages:





- lint
- test
- build
- security-scan
- deploy-staging
- e2e-tests
- deploy-production
- monitoring

11. Roadmap de Implementación





Fase 1: MVP (2-3 semanas)

-  Interfaz web básica
-  Validación en tiempo real
-  Generación de parámetros MT4
-  Calculadora de riesgo





Fase 2: Backend API (2-3 semanas)

-  API REST completa
-  Base de datos persistente
-  Sistema de usuarios
-  Logging avanzado

Fase 3: Características Avanzadas (3-4 semanas)

-  Integración con datos de mercado en tiempo real
-  Templates de trading plans
-  Historial y analytics
-  Mobile app (React Native)

Fase 4: Enterprise Features (4-5 semanas)

-  Multi-broker support
-  Advanced risk management
-  Team collaboration
-  API para terceros

12. Costos Estimados

Desarrollo

- **Frontend Developer (Senior):** 40 hrs/semana × 8 semanas = \$12,800
- **Backend Developer (Senior):** 40 hrs/semana × 8 semanas = \$12,800
- **DevOps Engineer:** 20 hrs/semana × 4 semanas = \$4,800
- **QA Tester:** 20 hrs/semana × 6 semanas = \$3,600

Total Desarrollo: ~\$34,000

Infraestructura (Mensual)

- **Cloud Hosting (AWS/GCP):** \$200-500
- **Database (MongoDB Atlas):** \$100-300
- **CDN (CloudFlare):** \$50-100
- **Monitoring (DataDog):** \$100-200

Total Mensual: \$450-1,100

ROI Esperado

- **Tiempo ahorrado por trader:** 15-30 min por plan
- **Reducción de errores:** 80-90%
- **Incremento en eficiencia:** 200-300%
- **Escalabilidad:** Soporta 1000+ usuarios concurrentes