JournalEdge.io - Product Requirements Document

Executive Summary

JournalEdge.io is a trading journal and analytics platform designed to help traders track, analyze, and improve their trading performance. The platform enables users to import trades from Tradovate, visualize performance metrics, tag trades for pattern recognition, and gain insights through comprehensive analytics.

Version: 1.0.0
Last Updated: October 20, 2025
Status: Initial Release

Table of Contents

Product Overview

Vision

Empower traders with actionable insights by providing a comprehensive journaling and analytics platform that transforms raw trade data into strategic intelligence.

Target Audience

- Day traders
- Swing traders
- Futures traders (initial focus on Tradovate users)
- Active retail traders seeking performance improvement

Core Value Propositions

- Simple trade import from Tradovate exports
- Visual calendar-based trade organization
- Deep performance analytics with custom tagging
- TradingView integration for visual trade execution analysis

---

Technical Architecture

Technology Stack

Frontend

- **Framework:** Next.js 14+ (App Router)
- **State Management:** Redux Toolkit
- **Language:** TypeScript 5+
- **Styling:** Tailwind CSS
- **Charts:** Lightweight Charts (primary), D3.js (advanced visualizations)
- **Routing:** Next.js App Router (file-based routing)
- **Form Management:** React Hook Form
- **Date Handling:** date-fns

Backend

- **Runtime:** Node.js 20+
- **Framework:** Fastify 4+
- **Language:** TypeScript 5+
- **API Style:** REST
- **File Upload:** @fastify/multipart
- **Validation:** Zod

Database & Auth

- **Database:** Supabase (PostgreSQL)
- **Authentication:** Supabase Auth
- **Storage:** Supabase Storage (for CSV uploads)
- **Real-time:** Supabase Realtime (future use)

External Services

- **TradingView:** Advanced Charts Widget
- **CSV Parser:** csv-parse

System Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                  Next.js Frontend (App Router)                  │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────────────┐   │
│  │   Server     │  │   Client     │  │  TradingView Widget  │   │
│  │ Components   │  │ Components   │  │                      │   │
│  │              │  │  (Redux)     │  │                      │   │
│  └──────────────┘  └──────────────┘  └──────────────────────┘   │
│  ┌───────────────────────────────────────────────────────────┐  │
│  │        File-based Routing (App Directory)                 │  │
│  │  /dashboard  /trades  /analytics  /settings               │  │
│  └───────────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────────┘
                 │
                 │ HTTPS/REST API
                 ▼
┌─────────────────────────────────────────────────────────────────┐
│                  Fastify Backend API                            │
│  ┌──────────────┐  ┌──────────────┐  ┌────────────────────────┐ │
│  │   Auth       │  │   Trade      │  │     Analytics          │ │
│  │ Middleware   │  │   Parser     │  │      Engine            │ │
│  └──────────────┘  └──────────────┘  └────────────────────────┘ │
└─────────────────────────────────────────────────────────────────┘
                 │
                 │
                 ▼
┌─────────────────────────────────────────────────────────────────┐
│                  Supabase Services                              │
│  ┌──────────────┐  ┌──────────────┐  ┌────────────────────────┐ │
│  │ PostgreSQL   │  │    Auth      │  │      Storage           │ │
│  │  Database    │  │              │  │    (CSV Files)         │ │
│  └──────────────┘  └──────────────┘  └────────────────────────┘ │
└─────────────────────────────────────────────────────────────────┘
```

User Stories

Authentication & Onboarding

- **US-1:** As a new user, I want to sign up with email/password so I can create an account
- **US-2:** As a returning user, I want to log in securely so I can access my trades
- **US-3:** As a user, I want to reset my password if I forget it

Trade Management

- **US-4:** As a trader, I want to upload my Tradovate CSV export so I can import my trades
- **US-5:** As a trader, I want to see my trades in a calendar view so I can understand my trading patterns over time
- **US-6:** As a trader, I want to view detailed information about a specific trade including executions on a chart
- **US-7:** As a trader, I want to add custom tags to my trades so I can categorize them by setup, market condition, or strategy

Analytics & Insights

- **US-8:** As a trader, I want to see my overall win rate so I can measure my success rate
- **US-9:** As a trader, I want to see my profit factor so I can understand my risk-reward performance
- **US-10:** As a trader, I want to see profitability broken down by day of week so I can identify my best trading days
- **US-11:** As a trader, I want to see profitability broken down by hour so I can identify my best trading times
- **US-12:** As a trader, I want to see analytics filtered by custom tags so I can measure strategy performance

---

Feature Requirements

Phase 1 - MVP Features

1. Authentication System

Priority: P0 (Critical)

Requirements:

- Email/password authentication via Supabase Auth
- JWT token-based session management
- Password reset functionality
- Protected routes on frontend
- Auth middleware on backend

Acceptance Criteria:

- User can sign up with email/password
- User receives confirmation email
- User can log in and receive JWT token
- Token is stored securely (httpOnly cookies)
- Protected pages redirect to login if unauthenticated
- User can log out and token is cleared
- Password reset flow sends email with reset link

Technical Notes:

- Use Supabase Auth SDK on both frontend and backend
- Implement auth middleware in Fastify to verify JWT
- Next.js middleware to protect routes
- Store session in secure httpOnly cookies

---

2. Trade Import System

Priority: P0 (Critical)

Requirements:

- File upload component supporting CSV files
- Parse Tradovate CSV export format
- Validate trade data before insertion
- Store raw CSV in Supabase Storage
- Extract and normalize trade data
- Handle duplicate imports gracefully

Tradovate CSV Expected Columns:

```
Contract, Trade Date, Buy/Sell, Qty, Price, T. Price,
P/L Open, P/L, Entry Time, Exit Time, Duration
```

Acceptance Criteria:

- User can upload CSV file (max 10MB)
- System parses Tradovate format correctly
- All trades are imported and associated with user
- Duplicate trades are detected and skipped
- User sees import progress/confirmation
- Invalid CSV shows clear error messages

Technical Notes:

- Use `@fastify/multipart` for file uploads
- Use `csv-parse` for CSV parsing
- Validate CSV structure with Zod schemas
- Transaction-based imports for data integrity

---

3. Calendar View

Priority: P0 (Critical)

Requirements:

- Monthly calendar view showing trading days
- Visual indicators for profitable/losing days
- Day cells show: total P/L, number of trades, win rate
- Click day to view trades for that date
- Navigate between months
- Highlight current day
- Color coding: green (profit), red (loss), gray (no trades)

Acceptance Criteria:

- Calendar displays current month by default
- Each trading day shows summary metrics
- Color coding is intuitive and accessible
- Clicking a day navigates to trade list for that date

- Month navigation works smoothly
- Performance is good with 1000+ trades

Technical Notes:

- Build custom calendar component or use date-fns
- Aggregate trade data by date in backend
- Use React Server Components where possible
- Implement client-side filtering for responsiveness

---

4. Trade Detail View

Priority: P0 (Critical)

Requirements:

- Display all trade information (entry, exit, P/L, duration, etc.)
- Show TradingView Advanced Chart with execution markers
- Display entry and exit prices on chart
- Show trade timeline with execution sequence
- Allow adding/editing custom tags
- Show trade-specific metrics (R-multiple, % gain, etc.)

TradingView Integration:

- Embed TradingView Advanced Charts Widget
- Plot entry executions (buy markers)
- Plot exit executions (sell markers)
- Set chart timeframe based on trade duration
- Support multiple instruments (futures contracts)

Acceptance Criteria:

- Trade details load quickly
- TradingView chart displays with correct symbol
- Entry/exit markers appear accurately on chart
- Tags can be added/removed
- All trade metrics calculate correctly
- Works on mobile devices

Technical Notes:

- Use TradingView Lightweight Charts or Advanced Charts Widget
- Client component for TradingView integration
- Lazy load chart component for performance
- Cache chart data where possible

---

5. Custom Tags System

Priority: P0 (Critical)

Requirements:

- Create custom tags (e.g., "Breakout", "Reversal", "News Event")

- Apply multiple tags to a single trade
- Color-code tags for visual organization
- Search/filter trades by tags
- View analytics per tag
- Manage tags (create, edit, delete)

Acceptance Criteria:

- User can create unlimited tags
- Tags can be assigned to trades during review
- Multiple tags per trade supported
- Tags persist across sessions
- Deleting a tag removes it from all trades
- Tag analytics show in reporting section

Technical Notes:

- Many-to-many relationship (trades ↔ tags)
- Implement tag management UI in settings
- Use tag slugs for URL filtering
- Index tags for query performance

---

6. Analytics Dashboard

Priority: P0 (Critical)

Requirements:

Core Metrics:

- Total P/L (all-time, selected period)
- Win Rate (%)
- Total Trades
- Profit Factor
- Average Win / Average Loss
- Largest Win / Largest Loss
- Average Trade Duration
- Expectancy

Breakdown Charts:

1. **P/L by Day of Week**
   - Bar chart showing total P/L for Mon-Sun
   - Win rate per day
2. **P/L by Hour**
   - Bar chart showing P/L by hour of day (entry time)
   - Identify best trading hours
3. **Cumulative P/L**
   - Line chart showing equity curve over time
   - Drawdown visualization
4. **Win/Loss Distribution**
   - Histogram of trade results
   - Show distribution of wins vs losses
5. **Tag Performance**
   - Table/chart showing P/L per tag

    - Win rate per tag
    - Number of trades per tag

Filters:

- Date range selector
- Tag filter (single or multiple)
- Instrument filter
- Min/Max P/L filter

Acceptance Criteria:

- All metrics calculate accurately
- Charts render smoothly with 1000+ trades
- Filters update metrics in real-time
- Data can be exported to CSV
- Dashboard is responsive on mobile

Technical Notes:

- Use Lightweight Charts for time-series data
- Use D3.js for complex custom visualizations
- Compute analytics in backend for accuracy
- Cache aggregated metrics with Redis (future)
- Use React Query for data fetching

Data Models

Database Schema (PostgreSQL via Supabase)

1. Users Table

sql

```sql
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  email VARCHAR(255) UNIQUE NOT NULL,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

  -- Supabase Auth handles password, so no password field here
  -- This table extends Supabase auth.users
);
```

2. Trades Table

sql

```sql
CREATE TABLE trades (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

  -- Trade Identification
  instrument VARCHAR(100) NOT NULL, -- e.g., "NQH24", "ESZ23"
  trade_date DATE NOT NULL,

  -- Trade Details
  side VARCHAR(10) NOT NULL, -- 'BUY' or 'SELL' (for opening)
  quantity INTEGER NOT NULL,

  -- Entry
  entry_price DECIMAL(12, 4) NOT NULL,
  entry_time TIMESTAMP WITH TIME ZONE NOT NULL,

  -- Exit
  exit_price DECIMAL(12, 4) NOT NULL,
  exit_time TIMESTAMP WITH TIME ZONE NOT NULL,

  -- Performance
  pnl DECIMAL(12, 2) NOT NULL,
  pnl_percent DECIMAL(8, 4),

  -- Additional Metrics
  duration_seconds INTEGER,
  commission DECIMAL(8, 2) DEFAULT 0,

  -- Metadata
  import_id UUID, -- Reference to import batch
  notes TEXT,

  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

  INDEX idx_user_date (user_id, trade_date),
  INDEX idx_user_entry_time (user_id, entry_time),
  INDEX idx_user_instrument (user_id, instrument)
);
```

3. Tags Table

sql

```sql
CREATE TABLE tags (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

  name VARCHAR(50) NOT NULL,
  slug VARCHAR(50) NOT NULL,
  color VARCHAR(7) DEFAULT '#3B82F6', -- Hex color
  description TEXT,

  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

  UNIQUE(user_id, slug),
  INDEX idx_user_tags (user_id)
);
```

4. Trade_Tags Table (Many-to-Many)

sql

```sql
CREATE TABLE trade_tags (
  trade_id UUID NOT NULL REFERENCES trades(id) ON DELETE CASCADE,
  tag_id UUID NOT NULL REFERENCES tags(id) ON DELETE CASCADE,

  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

  PRIMARY KEY (trade_id, tag_id),
  INDEX idx_tag_trades (tag_id)
);
```

5. Imports Table

sql

```
CREATE TABLE imports (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,

  filename VARCHAR(255) NOT NULL,
  file_path VARCHAR(500), -- Supabase Storage path
  file_size INTEGER,

  status VARCHAR(20) DEFAULT 'processing', -- processing, completed, failed
  trades_imported INTEGER DEFAULT 0,
  trades_skipped INTEGER DEFAULT 0,
  error_message TEXT,

  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  completed_at TIMESTAMP WITH TIME ZONE,

  INDEX idx_user_imports (user_id, created_at)
);
```

TypeScript Interfaces

typescript

```
// src/types/trade.ts

export interface Trade {
  id: string;
  userId: string;
  instrument: string;
  tradeDate: string; // ISO date string
  side: 'BUY' | 'SELL';
  quantity: number;
  entryPrice: number;
  entryTime: string; // ISO datetime string
  exitPrice: number;
  exitTime: string;
  pnl: number;
  pnlPercent?: number;
  durationSeconds?: number;
  commission?: number;
  importId?: string;
  notes?: string;
  tags?: Tag[];
  createdAt: string;
  updatedAt: string;
}

export interface Tag {
  id: string;
  userId: string;
  name: string;
  slug: string;
  color: string;
  description?: string;
  createdAt: string;
}

export interface TradeImport {
  id: string;
  userId: string;
  filename: string;
  filePath?: string;
  fileSize?: number;
  status: 'processing' | 'completed' | 'failed';
  tradesImported: number;
  tradesSkipped: number;
  errorMessage?: string;
  createdAt: string;
  completedAt?: string;
}

export interface CalendarDay {
  date: string; // ISO date
  totalPnl: number;
  tradeCount: number;
  winRate: number;
  wins: number;
  losses: number;
}

export interface AnalyticsMetrics {
  totalPnl: number;
  totalTrades: number;
  winningTrades: number;
  losingTrades: number;
  winRate: number;
  profitFactor: number;
  averageWin: number;
  averageLoss: number;
  largestWin: number;
  largestLoss: number;
  averageDuration: number; // seconds
  expectancy: number;
}

export interface DayOfWeekStats {
  day: string; // 'Monday', 'Tuesday', etc.
  totalPnl: number;
  tradeCount: number;
  winRate: number;
}

export interface HourOfDayStats {
  hour: number; // 0-23
  totalPnl: number;
  tradeCount: number;
  winRate: number;
}

export interface TagStats {
  tagId: string;
  tagName: string;
  tagColor: string;
  totalPnl: number;
  tradeCount: number;
  winRate: number;
  profitFactor: number;
}
```

API Specifications

Base URL

```
Development: http://localhost:3001/api
Production: https://api.journaledge.io/api
```

Authentication

All protected endpoints require JWT token in Authorization header:

```
Authorization: Bearer <jwt_token>
```

Endpoints

Authentication

POST /api/auth/register

typescript

```
Request Body:
{
  email: string;
  password: string;
}

Response: 201 Created
{
  user: {
    id: string;
    email: string;
  },
  session: {
    access_token: string;
    refresh_token: string;
  }
}
```

POST /api/auth/login

typescript

```
Request Body:
{
  email: string;
  password: string;
}

Response: 200 OK
{
  user: { id: string; email: string; },
  session: { access_token: string; refresh_token: string; }
}
```

POST /api/auth/logout

typescript

```
Response: 200 OK
{ message: "Logged out successfully" }
```

Trades

GET /api/trades

typescript

```typescript
Query Params:
  - startDate?: string (ISO date)
  - endDate?: string (ISO date)
  - tags?: string[] (tag IDs)
  - instrument?: string
  - limit?: number (default: 100)
  - offset?: number (default: 0)

Response: 200 OK
{
  trades: Trade[];
  total: number;
  limit: number;
  offset: number;
}
```

GET /api/trades/:id

typescript

```typescript
Response: 200 OK
{
  trade: Trade;
}
```

PUT /api/trades/:id

typescript

```
Request Body:
{
  notes?: string;
  tagIds?: string[];
}

Response: 200 OK
{
  trade: Trade;
}
```

DELETE /api/trades/:id

typescript

```
Response: 204 No Content
```

POST /api/trades/import

typescript

```
Request: multipart/form-data
  - file: CSV file

Response: 202 Accepted
{
  importId: string;
  message: "Import started";
}
```

GET /api/trades/import/:importId

typescript

```
Response: 200 OK
{
  import: TradeImport;
}
```

Calendar

GET /api/calendar

typescript

```
Query Params:
  - month: string (YYYY-MM)
  - tags?: string[]

Response: 200 OK
{
  days: CalendarDay[];
  month: string;
}
```

Analytics

GET /api/analytics/metrics

typescript

```
Query Params:
  - startDate?: string
  - endDate?: string
  - tags?: string[]
  - instrument?: string

Response: 200 OK
{
  metrics: AnalyticsMetrics;
}
```

GET /api/analytics/by-day-of-week

typescript

```typescript
  Query Params: (same as metrics)

  Response: 200 OK
  {
    stats: DayOfWeekStats[];
  }
```

GET /api/analytics/by-hour

typescript

```typescript
  Query Params: (same as metrics)

  Response: 200 OK
  {
    stats: HourOfDayStats[];
  }
```

GET /api/analytics/by-tag

typescript

```typescript
  Query Params:
    - startDate?: string
    - endDate?: string

  Response: 200 OK
  {
    stats: TagStats[];
  }
```

GET /api/analytics/equity-curve

typescript

```typescript
Query Params: (same as metrics)

Response: 200 OK
{
  data: Array<{
    date: string;
    cumulativePnl: number;
  }>;
}
```

Tags

GET /api/tags

typescript

```typescript
Response: 200 OK
{
  tags: Tag[];
}
```

POST /api/tags

typescript

```
  Request Body:
  {
    name: string;
    color?: string;
    description?: string;
  }

  Response: 201 Created
  {
    tag: Tag;
  }
```

PUT /api/tags/:id

typescript

```
  Request Body:
  {
    name?: string;
    color?: string;
    description?: string;
  }

  Response: 200 OK
  {
    tag: Tag;
  }
```

DELETE /api/tags/:id

typescript

```
  Response: 204 No Content
```

UI/UX Requirements

Design System

Color Palette:

- Primary: Blue (`#3B82F6`)
- Success/Profit: Green (`#10B981`)
- Error/Loss: Red (`#EF4444`)
- Warning: Yellow (`#F59E0B`)
- Neutral: Gray scale

Typography:

- Font: Inter or system font stack
- Headings: Bold, larger sizes
- Body: Regular weight
- Monospace: For numbers and financial data

Page Layouts

1. Dashboard (/)

- Overview metrics cards (P/L, Win Rate, Total Trades, Profit Factor)
- Recent trades list (last 10)
- Quick stats charts
- Import trade button (prominent)

2. Calendar View (/calendar)

- Month selector
- Calendar grid with daily summaries
- Filters (tags, date range)
- Legend for color coding

3. Trade List (/trades)

- Searchable/filterable table
- Columns: Date, Instrument, Side, P/L, Duration, Tags
- Pagination
- Export to CSV button
- Click row to view detail

4. Trade Detail (/trades/:id)

- Trade header with key metrics
- TradingView chart with executions
- Trade timeline
- Tag management
- Notes section
- Edit/delete actions

5. Analytics (/analytics)

- Metrics overview
- Multiple chart sections:
  - P/L by day of week
  - P/L by hour
  - Equity curve
  - Win/loss distribution
  - Tag performance table
- Date range picker
- Tag filters
- Export reports button

6. Tags Management (/settings/tags)

- List of all tags
- Create new tag form
- Edit tag inline
- Delete with confirmation
- Color picker for each tag

7. Settings (/settings)

- Profile information
- Import history
- Data export
- Account settings
- Theme preferences (future)

Responsive Design

- Mobile-first approach
- Breakpoints: 640px (sm), 768px (md), 1024px (lg), 1280px (xl)
- Touch-friendly buttons and controls
- Collapsible navigation on mobile

Security & Authentication

Authentication Flow

1. User registers/logs in via Supabase Auth
2. Supabase returns JWT access token and refresh token
3. Frontend stores tokens in httpOnly cookies
4. All API requests include JWT in Authorization header
5. Fastify middleware verifies JWT with Supabase
6. Refresh token used to get new access token when expired

Security Measures

- **Password Requirements:** Min 8 characters, mix of letters and numbers
- **Rate Limiting:** Implement on login/register endpoints
- **SQL Injection Prevention:** Use parameterized queries
- **XSS Prevention:** Sanitize user inputs, use Content Security Policy
- **CSRF Protection:** Use CSRF tokens for state-changing operations

- **File Upload Security:**
  - Validate file type (CSV only)
  - Limit file size (10MB)
  - Scan for malicious content
  - Store in isolated Supabase Storage bucket
- **Row-Level Security (RLS):** Enable on Supabase tables to ensure users can only access their own data

Supabase RLS Policies

sql

```sql
-- Users can only read their own data
CREATE POLICY "Users can view own trades"
  ON trades FOR SELECT
  USING (auth.uid() = user_id);

CREATE POLICY "Users can insert own trades"
  ON trades FOR INSERT
  WITH CHECK (auth.uid() = user_id);

CREATE POLICY "Users can update own trades"
  ON trades FOR UPDATE
  USING (auth.uid() = user_id);

CREATE POLICY "Users can delete own trades"
  ON trades FOR DELETE
  USING (auth.uid() = user_id);

-- Similar policies for tags and trade_tags tables
```

Analytics & Reporting

Key Calculations

Win Rate:

```
Win Rate = (Winning Trades / Total Trades) × 100
```

Profit Factor:

```
Profit Factor = Gross Profit / Gross Loss
```

Expectancy:

```
Expectancy = (Win Rate × Avg Win) - (Loss Rate × Avg Loss)
```

Average Duration:

```
Average Duration = Sum of all trade durations / Total trades
```

Performance Considerations

- Pre-calculate analytics for large datasets
- Use database aggregation functions
- Implement caching for frequently accessed metrics
- Paginate large result sets
- Use database indexes on commonly queried columns

Export Functionality

Users can export:

- Trade list (CSV)
- Analytics report (PDF - future)
- Custom date range data

Future Enhancements (Post-MVP)

Phase 2

- **Automated Import:** Direct API integration with Tradovate
- **Real-time Trade Tracking:** Live P/L updates
- **Advanced Charting:** More chart types and indicators
- **Trade Replay:** Replay trades step-by-step
- **Screenshots:** Upload trade screenshots
- **Mobile App:** React Native iOS/Android apps

Phase 3

- **AI Insights:** ML-powered pattern recognition
- **Social Features:** Share trades with community
- **Strategy Backtesting:** Test strategies on historical data
- **Risk Management:** Position sizing calculators
- **Multi-broker Support:** Import from multiple sources
- **Team Accounts:** Share workspace with trading team

Phase 4

- **Live Trading Signals:** Real-time alerts
- **Automated Execution:** Connect directly to brokers
- **Video Journaling:** Record trade thought process
- **Mentorship Platform:** Connect with expert traders

Technical Implementation Notes

Frontend Structure (Next.js App Router)

```
src/
├── app/
│   ├── (auth)/
│   │   ├── login/
│   │   │   └── page.tsx
│   │   └── register/
│   │      └── page.tsx
│   ├── (dashboard)/
│   │   ├── layout.tsx
│   │   ├── page.tsx (dashboard)
│   │   ├── calendar/
│   │   │   └── page.tsx
│   │   ├── trades/
│   │   │   ├── page.tsx (list)
│   │   │   └── [id]/
│   │   │      └── page.tsx (detail)
│   │   ├── analytics/
│   │   │   └── page.tsx
│   │   └── settings/
│   │      ├── page.tsx
│   │      └── tags/
│   │         └── page.tsx
│   ├── api/ (optional - or use Fastify only)
│   └── layout.tsx (root)
├── components/
│   ├── ui/ (shadcn components)
│   ├── charts/
│   ├── calendar/
│   ├── trades/
│   └── analytics/
├── lib/
│   ├── supabase/
│   ├── api-client.ts
│   └── utils.ts
├── store/ (Redux)
│   ├── slices/
│   └── store.ts
└── types/
   └── index.ts
```

Backend Structure (Fastify)

```
src/
├── index.ts (entry point)
├── app.ts (Fastify app setup)
├── config/
│   ├── database.ts
│   └── supabase.ts
├── middleware/
│   ├── auth.ts
│   └── error-handler.ts
├── routes/
│   ├── auth.ts
│   ├── trades.ts
│   ├── calendar.ts
│   ├── analytics.ts
│   └── tags.ts
├── services/
│   ├── trade-parser.ts
│   ├── analytics.ts
│   └── import.ts
├── models/
│   └── (types/interfaces)
└── utils/
    ├── validation.ts
    └── calculations.ts
```

Environment Variables

bash

```
# Frontend (.env.local)
NEXT_PUBLIC_SUPABASE_URL=
NEXT_PUBLIC_SUPABASE_ANON_KEY=
NEXT_PUBLIC_API_URL=http://localhost:3001
NEXT_PUBLIC_TRADINGVIEW_API_KEY=

# Backend (.env)
SUPABASE_URL=
SUPABASE_SERVICE_KEY=
JWT_SECRET=
PORT=3001
NODE_ENV=development
```

Development Workflow

1. Set up Supabase project and get credentials
2. Run database migrations for schema
3. Set up Fastify backend with auth middleware
4. Create Next.js frontend with App Router structure
5. Implement authentication flow
6. Build trade import functionality
7. Create calendar view
8. Implement trade detail with TradingView
9. Build analytics dashboard
10. Add tag management
11. Testing and optimization

Success Metrics

Technical KPIs

- Page load time < 2 seconds
- API response time < 500ms (p95)
- CSV import processing < 30 seconds for 1000 trades
- Chart rendering < 1 second
- 99.9% uptime

User Engagement KPIs

- Daily active users
- Trades imported per user
- Tags created per user
- Time spent in analytics
- Feature adoption rate

Business KPIs

- User registration rate
- User retention (30-day)
- Conversion rate (free to paid - future)
- Churn rate

Conclusion

JournalEdge.io MVP focuses on delivering core functionality that provides immediate value to traders: easy trade import, visual organization, and actionable analytics. The architecture is scalable and built with modern technologies to support future enhancements while maintaining performance and security.

Next Steps:

1. Design mockups and user flows
2. Set up development environment
3. Initialize repositories (frontend & backend)
4. Begin sprint planning
5. Start with authentication and database setup