그리드 트레이딩 봇 백엔드 API 명세서

📋 프로젝트 개요

프로젝트명: Grid Trading Bot Backend API **목적:** 암호화폐 그리드 트레이딩 자동화 봇 관리 시스템 **지원 거래소:** Upbit, Binance **프론트엔드 포트:** 3009 **백엔드 포트 (권장):** 3000

🛠 기술 스택 (권장)

Core

- **Runtime:** Node.js 18+
- **Framework:** Express.js 4.x
- **Language:** TypeScript 5.x
- **Database:** MongoDB (NoSQL - 유연한 스키마) 또는 PostgreSQL (관계형)
- **ORM/ODM:** Mongoose (MongoDB) 또는 Prisma (PostgreSQL)

Authentication & Security

- **인증:** JWT (JSON Web Tokens)
- **암호화:** bcrypt (비밀번호), crypto (API 키 암호화)
- **환경변수:** dotenv
- **보안 헤더:** helmet
- **Rate Limiting:** express-rate-limit

Exchange Integration

- **Upbit:** ccxt 라이브러리 또는 직접 API 연동
- **Binance:** ccxt 라이브러리 또는 binance-api-node
- **WebSocket:** ws (실시간 가격 데이터)

Background Jobs

- **스케줄러:** node-cron 또는 bull (Redis 기반 큐)
- **프로세스 관리:** PM2

Utilities

- **Validation:** joi 또는 zod
- **Logging:** winston
- **CORS:** cors
- **HTTP Client:** axios

🗂 프로젝트 구조 (권장)

```
v0-grid-transaction-backend/
├── src/
│   ├── config/              # 설정 파일
│   │   ├── database.ts
│   │   ├── exchange.ts
│   │   └── constants.ts
│   ├── models/              # 데이터 모델
│   │   ├── User.ts
│   │   ├── Bot.ts
│   │   ├── Trade.ts
│   │   ├── GridLevel.ts
│   │   └── Credential.ts
│   ├── controllers/         # 컨트롤러 (비즈니스 로직)
│   │   ├── authController.ts
│   │   ├── botController.ts
│   │   ├── tradeController.ts
│   │   └── credentialController.ts
│   ├── routes/              # API 라우트
│   │   ├── auth.ts
│   │   ├── bots.ts
│   │   ├── trades.ts
│   │   ├── exchange.ts
│   │   └── credentials.ts
│   ├── middleware/          # 미들웨어
│   │   ├── auth.ts
│   │   ├── errorHandler.ts
│   │   └── validator.ts
│   ├── services/            # 외부 서비스 연동
│   │   ├── upbitService.ts
│   │   ├── binanceService.ts
│   │   ├── gridEngine.ts
│   │   └── tradeExecutor.ts
│   ├── jobs/                # 백그라운드 작업
│   │   ├── botMonitor.ts
│   │   └── priceUpdater.ts
│   ├── utils/               # 유틸리티 함수
│   │   ├── encryption.ts
│   │   ├── gridCalculator.ts
│   │   └── logger.ts
│   ├── types/               # TypeScript 타입 정의
│   │   └── index.ts
│   └── app.ts               # Express 앱 설정
├── .env                     # 환경변수
├── .env.example
├── package.json
├── tsconfig.json
└── README.md
```

🗄 데이터 모델

1. User (사용자)

```
interface User {
  _id: string
  email: string
  password: string          // bcrypt 해싱
  name?: string
  createdAt: Date
  updatedAt: Date
}
```

2. Bot (트레이딩 봇)

```
interface Bot {
  _id: string
  userId: string            // User 참조
  exchange: "upbit" | "binance"
  ticker: string            // 예: "KRW-BTC", "BTCUSDT"

  // 그리드 설정
  lowerPrice: number
  upperPrice: number
  priceChangePercent: number
  gridCount: number
  orderAmount: number       // 그리드당 주문 금액
  stopAtMax: boolean        // 상단 도달시 중지 여부

  // 상태
  status: "running" | "stopped" | "error"

  // 통계
  investmentAmount: number
  currentProfit: number
  totalTrades: number

  // 메타데이터
  createdAt: Date
  updatedAt: Date
  lastExecutedAt?: Date
  errorMessage?: string
}
```

3. GridLevel (그리드 레벨)

```
interface GridLevel {
  _id: string
  botId: string             // Bot 참조
  price: number
```

```
  type: "buy" | "sell"
  status: "available" | "pending" | "filled"
  orderId?: string          // 거래소 주문 ID
  filledAt?: Date
  createdAt: Date
  updatedAt: Date
}
```

4. Trade (거래 내역)

```
interface Trade {
  _id: string
  botId: string             // Bot 참조
  gridLevelId?: string      // GridLevel 참조

  type: "buy" | "sell"
  price: number
  amount: number
  total: number             // price * amount
  profit?: number           // 매도시 수익

  orderId: string           // 거래소 주문 ID
  executedAt: Date
  createdAt: Date
}
```

5. Credential (거래소 인증 정보)

```
interface Credential {
  _id: string
  userId: string            // User 참조
  exchange: "upbit" | "binance"

  // 암호화된 API 키
  apiKey: string            // AES-256 암호화
  secretKey: string         // AES-256 암호화

  // 선택 사항
  ipWhitelist?: string
  ipRestricted?: boolean

  isValid: boolean          // 인증 유효성 검증 결과
  lastValidatedAt?: Date

  createdAt: Date
  updatedAt: Date
}
```

🖋 API 엔드포인트 명세

Base URL

```
http://localhost:3000/api
```

---

1 인증 (Authentication)

POST /auth/register

사용자 회원가입

Request Body:

```
{
  "email": "user@example.com",
  "password": "securePassword123",
  "name": "홍길동"
}
```

Response (201):

```
{
  "success": true,
  "message": "회원가입이 완료되었습니다",
  "data": {
    "userId": "6501234567890abcdef12345",
    "email": "user@example.com",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}
```

POST /auth/login

사용자 로그인

Request Body:

```
{
  "email": "user@example.com",
  "password": "securePassword123"
}
```

Response (200):

```
{
  "success": true,
  "data": {
    "userId": "6501234567890abcdef12345",
    "email": "user@example.com",
    "name": "홍길동",
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}
```

POST /auth/logout

사용자 로그아웃 (선택 사항)

Headers:

```
Authorization: Bearer {token}
```

Response (200):

```
{
  "success": true,
  "message": "로그아웃되었습니다"
}
```

---

2 봇 관리 (Bot Management)

POST /bots

새로운 그리드 봇 생성

Headers:

```
Authorization: Bearer {token}
```

Request Body:

```
{
  "exchange": "upbit",
  "ticker": "KRW-BTC",
```

```
  "lowerPrice": 50000000,
  "upperPrice": 70000000,
  "priceChangePercent": 2,
  "orderAmount": 10000,
  "stopAtMax": false,
  "autoStart": true
}
```

Response (201):

```
{
  "success": true,
  "message": "봇이 생성되었습니다",
  "data": {
    "botId": "6501234567890abcdef12346",
    "exchange": "upbit",
    "ticker": "KRW-BTC",
    "gridCount": 24,
    "investmentAmount": 240000,
    "status": "running",
    "createdAt": "2025-01-23T10:30:00.000Z"
  }
}
```

GET /bots

사용자의 모든 봇 조회

Headers:

```
Authorization: Bearer {token}
```

Query Parameters:

- `status` (optional): "running" | "stopped" | "error"
- `exchange` (optional): "upbit" | "binance"

Response (200):

```
{
  "success": true,
  "data": {
    "bots": [
      {
        "_id": "6501234567890abcdef12346",
        "exchange": "upbit",
```

```
        "ticker": "KRW-BTC",
        "status": "running",
        "currentProfit": 15000,
        "profitPercent": 6.25,
        "totalTrades": 8,
        "investmentAmount": 240000,
        "createdAt": "2025-01-23T10:30:00.000Z"
      }
    ],
    "summary": {
      "totalBots": 5,
      "activeBots": 3,
      "totalProfit": 45000,
      "totalInvestment": 1000000
    }
  }
}
```

GET /bots/:id

특정 봇 상세 정보 조회

Headers:

```
Authorization: Bearer {token}
```

Response (200):

```
{
  "success": true,
  "data": {
    "_id": "6501234567890abcdef12346",
    "exchange": "upbit",
    "ticker": "KRW-BTC",
    "lowerPrice": 50000000,
    "upperPrice": 70000000,
    "priceChangePercent": 2,
    "gridCount": 24,
    "orderAmount": 10000,
    "stopAtMax": false,
    "status": "running",
    "investmentAmount": 240000,
    "currentProfit": 15000,
    "profitPercent": 6.25,
    "totalTrades": 8,
    "currentPrice": 62000000,
    "createdAt": "2025-01-23T10:30:00.000Z",
    "lastExecutedAt": "2025-01-23T15:45:00.000Z"
```

```
    }
  }
```

PUT /bots/:id

봇 설정 수정

Headers:

```
Authorization: Bearer {token}
```

Request Body:

```
{
  "orderAmount": 15000,
  "stopAtMax": true
}
```

Response (200):

```
{
  "success": true,
  "message": "봇 설정이 업데이트되었습니다",
  "data": {
    "_id": "6501234567890abcdef12346",
    "orderAmount": 15000,
    "stopAtMax": true,
    "updatedAt": "2025-01-23T16:00:00.000Z"
  }
}
```

POST /bots/:id/start

봇 시작

Headers:

```
Authorization: Bearer {token}
```

Response (200):

```
{
  "success": true,
  "message": "봇이 시작되었습니다",
  "data": {
    "botId": "6501234567890abcdef12346",
    "status": "running"
  }
}
```

POST /bots/:id/stop

봇 중지

Headers:

```
Authorization: Bearer {token}
```

Response (200):

```
{
  "success": true,
  "message": "봇이 중지되었습니다",
  "data": {
    "botId": "6501234567890abcdef12346",
    "status": "stopped"
  }
}
```

DELETE /bots/:id

봇 삭제

Headers:

```
Authorization: Bearer {token}
```

Response (200):

```
{
  "success": true,
  "message": "봇이 삭제되었습니다"
}
```

3 그리드 & 거래 (Grid & Trading)

GET /bots/:id/grid-levels

봇의 그리드 레벨 목록 조회

Headers:

```
Authorization: Bearer {token}
```

Query Parameters:

- `status` (optional): "available" | "pending" | "filled"

Response (200):

```
{
  "success": true,
  "data": {
    "gridLevels": [
      {
        "_id": "6501234567890abcdef12347",
        "price": 50000000,
        "type": "buy",
        "status": "filled",
        "orderId": "upbit-order-123",
        "filledAt": "2025-01-23T11:00:00.000Z"
      },
      {
        "_id": "6501234567890abcdef12348",
        "price": 51000000,
        "type": "sell",
        "status": "pending",
        "orderId": "upbit-order-124"
      },
      {
        "_id": "6501234567890abcdef12349",
        "price": 52000000,
        "type": "buy",
        "status": "available"
      }
    ],
    "currentPrice": 51500000
  }
}
```

GET /bots/:id/trades

봇의 거래 내역 조회

Headers:

```
Authorization: Bearer {token}
```

Query Parameters:

- `limit` (optional): 기본값 50
- `offset` (optional): 기본값 0
- `startDate` (optional): ISO 8601 날짜
- `endDate` (optional): ISO 8601 날짜

Response (200):


```
{
  "success": true,
  "data": {
    "trades": [
      {
        "_id": "6501234567890abcdef1234a",
        "type": "sell",
        "price": 51000000,
        "amount": 0.0002,
        "total": 10200,
        "profit": 200,
        "orderId": "upbit-order-124",
        "executedAt": "2025-01-23T14:30:00.000Z"
      },
      {
        "_id": "6501234567890abcdef1234b",
        "type": "buy",
        "price": 50000000,
        "amount": 0.0002,
        "total": 10000,
        "orderId": "upbit-order-123",
        "executedAt": "2025-01-23T11:00:00.000Z"
      }
    ],
    "pagination": {
      "total": 150,
      "limit": 50,
      "offset": 0,
      "hasMore": true
    }
  }
}
```

GET /bots/:id/performance

봇의 성과 통계 조회

Headers:

```
Authorization: Bearer {token}
```

Response (200):

```
{
  "success": true,
  "data": {
    "totalProfit": 15000,
    "profitPercent": 6.25,
    "totalTrades": 8,
    "buyTrades": 4,
    "sellTrades": 4,
    "avgProfitPerTrade": 1875,
    "investmentAmount": 240000,
    "currentValue": 255000,
    "runningDays": 7,
    "dailyAvgProfit": 2142.86
  }
}
```

---

4 거래소 연동 (Exchange Integration)

GET /exchange/tickers/:exchange

거래소의 사용 가능한 티커 목록 조회

Headers:

```
Authorization: Bearer {token}
```

Path Parameters:

- `exchange`: "upbit" | "binance"

Response (200):

```
{
  "success": true,
```

```
  "data": {
    "tickers": [
      {
        "symbol": "KRW-BTC",
        "koreanName": "비트코인",
        "englishName": "Bitcoin"
      },
      {
        "symbol": "KRW-ETH",
        "koreanName": "이더리움",
        "englishName": "Ethereum"
      }
    ]
  }
}
```

GET /exchange/price/:exchange/:ticker

특정 티커의 현재 가격 조회

Headers:

```
Authorization: Bearer {token}
```

Path Parameters:

- `exchange`: "upbit" | "binance"
- `ticker`: 티커 심볼 (예: "KRW-BTC", "BTCUSDT")

Response (200):

```
{
  "success": true,
  "data": {
    "ticker": "KRW-BTC",
    "currentPrice": 62000000,
    "change24h": 1.5,
    "volume24h": 1234567890,
    "timestamp": "2025-01-23T16:30:00.000Z"
  }
}
```

POST /exchange/validate-credentials

거래소 API 인증 정보 유효성 검증

Headers:

```
Authorization: Bearer {token}
```

Request Body:

```
{
  "exchange": "upbit",
  "apiKey": "user-api-key",
  "secretKey": "user-secret-key"
}
```

Response (200):

```
{
  "success": true,
  "message": "API 인증 정보가 유효합니다",
  "data": {
    "isValid": true,
    "accountInfo": {
      "currency": "KRW",
      "balance": 1000000,
      "locked": 50000,
      "avgBuyPrice": 0
    }
  }
}
```

---

5 인증 정보 관리 (Credentials)

POST /credentials

거래소 API 인증 정보 저장

Headers:

```
Authorization: Bearer {token}
```

Request Body:

```
{
  "exchange": "upbit",
  "apiKey": "user-api-key",
```

```
  "secretKey": "user-secret-key",
  "ipWhitelist": "123.456.789.0"
}
```

Response (201):

```
{
  "success": true,
  "message": "인증 정보가 저장되었습니다",
  "data": {
    "credentialId": "6501234567890abcdef1234c",
    "exchange": "upbit",
    "isValid": true
  }
}
```

GET /credentials

사용자의 저장된 인증 정보 목록 조회

Headers:

```
Authorization: Bearer {token}
```

Response (200):

```
{
  "success": true,
  "data": {
    "credentials": [
      {
        "_id": "6501234567890abcdef1234c",
        "exchange": "upbit",
        "apiKey": "****-****-****-ab12",
        "isValid": true,
        "lastValidatedAt": "2025-01-23T10:00:00.000Z",
        "createdAt": "2025-01-20T09:00:00.000Z"
      },
      {
        "_id": "6501234567890abcdef1234d",
        "exchange": "binance",
        "apiKey": "****-****-****-cd34",
        "isValid": true,
        "lastValidatedAt": "2025-01-23T10:00:00.000Z",
        "createdAt": "2025-01-21T14:00:00.000Z"
      }
```

```
    ]
  }
}
```

GET /credentials/:exchange

특정 거래소의 인증 정보 조회

Headers:

```
Authorization: Bearer {token}
```

Path Parameters:

- `exchange`: "upbit" | "binance"

Response (200):

```
{
  "success": true,
  "data": {
    "_id": "6501234567890abcdef1234c",
    "exchange": "upbit",
    "apiKey": "****-****-****-ab12",
    "ipWhitelist": "123.456.789.0",
    "isValid": true,
    "lastValidatedAt": "2025-01-23T10:00:00.000Z",
    "createdAt": "2025-01-20T09:00:00.000Z"
  }
}
```

PUT /credentials/:exchange

인증 정보 수정

Headers:

```
Authorization: Bearer {token}
```

Request Body:

```
{
  "apiKey": "new-api-key",
```

```
  "secretKey": "new-secret-key",
  "ipWhitelist": "123.456.789.100"
}
```

Response (200):

```
{
  "success": true,
  "message": "인증 정보가 업데이트되었습니다",
  "data": {
    "exchange": "upbit",
    "isValid": true,
    "updatedAt": "2025-01-23T16:00:00.000Z"
  }
}
```

DELETE /credentials/:exchange

인증 정보 삭제

Headers:

```
Authorization: Bearer {token}
```

Response (200):

```
{
  "success": true,
  "message": "인증 정보가 삭제되었습니다"
}
```

🔒 보안 요구사항

1. API 키 암호화

- **알고리즘:** AES-256-GCM
- **키 관리:** 환경변수에 암호화 키 저장 (`ENCRYPTION_KEY`)
- **저장:** DB에는 암호화된 값만 저장
- **복호화:** 거래 실행 시에만 메모리에서 일시적으로 복호화

2. 비밀번호 해싱

- **알고리즘:** bcrypt

- **Salt Rounds:** 10 이상

3. JWT 토큰

- **만료시간:** Access Token 1시간, Refresh Token 7일 (선택)
- **시크릿 키:** 환경변수 `JWT_SECRET`
- **Payload:** userId, email

4. Rate Limiting

- **인증 API:** 5회/분
- **봇 생성:** 10회/시간
- **일반 API:** 100회/분

5. CORS 설정

```
const corsOptions = {
  origin: 'http://localhost:3009',
  credentials: true
}
```

6. IP 화이트리스트 검증

- 사용자가 설정한 IP와 거래소 설정 일치 여부 확인
- 불일치 시 경고 메시지 반환

⚙️ 그리드 트레이딩 로직

1. 그리드 초기화 (봇 생성 시)

```
// 1. 그리드 개수 계산
gridCount = Math.floor(
  Math.log(upperPrice / lowerPrice) /
  Math.log(1 + priceChangePercent / 100)
)

// 2. 그리드 레벨 생성
const priceStep = (upperPrice - lowerPrice) / gridCount

for (let i = 0; i <= gridCount; i++) {
  const price = lowerPrice + (priceStep * i)
  const type = i % 2 === 0 ? 'buy' : 'sell'

  // GridLevel 저장
  await GridLevel.create({
    botId,
    price,
    type,
```

```
    status: 'available'
  })
}
```

2. 초기 매수 주문 실행

```
// 현재가 조회
const currentPrice = await getExchangePrice(exchange, ticker)

// 현재가보다 낮은 가격의 buy 그리드에 주문
const buyGrids = await GridLevel.find({
  botId,
  type: 'buy',
  price: { $lt: currentPrice },
  status: 'available'
})

for (const grid of buyGrids) {
  const order = await placeOrder({
    exchange,
    ticker,
    type: 'buy',
    price: grid.price,
    amount: orderAmount / grid.price
  })

  await GridLevel.updateOne(
    { _id: grid._id },
    {
      status: 'pending',
      orderId: order.id
    }
  )
}
```

3. 주문 체결 모니터링 (백그라운드 작업)

```
// 매 5초마다 실행
setInterval(async () => {
  const runningBots = await Bot.find({ status: 'running' })

  for (const bot of runningBots) {
    // pending 상태의 그리드 조회
    const pendingGrids = await GridLevel.find({
      botId: bot._id,
      status: 'pending'
    })

    for (const grid of pendingGrids) {
```

```
        const orderStatus = await checkOrderStatus(
          bot.exchange,
          grid.orderId
        )

        if (orderStatus === 'filled') {
          // 그리드 상태 업데이트
          await GridLevel.updateOne(
            { _id: grid._id },
            {
              status: 'filled',
              filledAt: new Date()
            }
          )

          // 거래 내역 저장
          await Trade.create({
            botId: bot._id,
            gridLevelId: grid._id,
            type: grid.type,
            price: grid.price,
            amount: orderAmount / grid.price,
            total: orderAmount,
            orderId: grid.orderId,
            executedAt: new Date()
          })

          // 매수 체결 시 -> 위쪽 매도 주문 생성
          if (grid.type === 'buy') {
            const nextSellGrid = await GridLevel.findOne({
              botId: bot._id,
              type: 'sell',
              price: { $gt: grid.price },
              status: 'available'
            }).sort({ price: 1 })

            if (nextSellGrid) {
              const sellOrder = await placeOrder({
                exchange: bot.exchange,
                ticker: bot.ticker,
                type: 'sell',
                price: nextSellGrid.price,
                amount: orderAmount / grid.price
              })

              await GridLevel.updateOne(
                { _id: nextSellGrid._id },
                {
                  status: 'pending',
                  orderId: sellOrder.id
                }
              )
            }
          }
```

```
        // 매도 체결 시 -> 아래쪽 매수 주문 생성 & 수익 계산
        if (grid.type === 'sell') {
          // 수익 계산 (매도가 - 매수가)
          const buyTrade = await Trade.findOne({
            botId: bot._id,
            type: 'buy',
            price: { $lt: grid.price }
          }).sort({ executedAt: -1 })

          const profit = buyTrade
            ? (grid.price - buyTrade.price) * (orderAmount / grid.price)
            : 0

          await Trade.updateOne(
            { gridLevelId: grid._id },
            { profit }
          )

          // 봇 수익 업데이트
          await Bot.updateOne(
            { _id: bot._id },
            {
              $inc: {
                currentProfit: profit,
                totalTrades: 1
              }
            }
          )

          // 아래쪽 매수 주문 생성
          const nextBuyGrid = await GridLevel.findOne({
            botId: bot._id,
            type: 'buy',
            price: { $lt: grid.price },
            status: 'available'
          }).sort({ price: -1 })

          if (nextBuyGrid) {
            const buyOrder = await placeOrder({
              exchange: bot.exchange,
              ticker: bot.ticker,
              type: 'buy',
              price: nextBuyGrid.price,
              amount: orderAmount / nextBuyGrid.price
            })

            await GridLevel.updateOne(
              { _id: nextBuyGrid._id },
              {
                status: 'pending',
                orderId: buyOrder.id
              }
            )
```

```
            }

            // 그리드를 다시 사용 가능 상태로
            await GridLevel.updateOne(
              { _id: grid._id },
              {
                status: 'available',
                orderId: null,
                filledAt: null
              }
            )
          }
        }
      }
    }
  }, 5000)
```

4. 상단 도달 시 중지 로직

```
if (bot.stopAtMax && currentPrice >= bot.upperPrice) {
  // 모든 미체결 주문 취소
  const pendingGrids = await GridLevel.find({
    botId: bot._id,
    status: 'pending'
  })

  for (const grid of pendingGrids) {
    await cancelOrder(bot.exchange, grid.orderId)
    await GridLevel.updateOne(
      { _id: grid._id },
      { status: 'available', orderId: null }
    )
  }

  // 봇 중지
  await Bot.updateOne(
    { _id: bot._id },
    { status: 'stopped' }
  )
}
```

백그라운드 작업 (Background Jobs)

1. 봇 모니터링 (`botMonitor.ts`)

- **실행 주기:** 5초
- **작업 내용:**
  - 모든 running 상태 봇 조회

    - pending 주문 상태 확인
    - 체결된 주문 처리
    - 다음 그리드 주문 생성
    - 에러 발생 시 봇 상태를 'error'로 변경

2. 가격 업데이터 (`priceUpdater.ts`)

- **실행 주기:** 10초
- **작업 내용:**
    - 모든 활성 봇의 티커 현재가 조회
    - 캐시에 저장 (Redis 권장)
    - WebSocket으로 프론트엔드에 실시간 전송 (선택)

3. 통계 업데이터

- **실행 주기:** 1분
- **작업 내용:**
    - 각 봇의 수익률 재계산
    - 일일/주간/월간 통계 업데이트

📊 에러 처리

에러 응답 포맷

```
{
  "success": false,
  "error": {
    "code": "BOT_NOT_FOUND",
    "message": "봇을 찾을 수 없습니다",
    "details": {}
  }
}
```

에러 코드 목록

코드	HTTP Status	메시지
`UNAUTHORIZED`	401	인증이 필요합니다
`INVALID_TOKEN`	401	유효하지 않은 토큰입니다
`FORBIDDEN`	403	접근 권한이 없습니다
`USER_NOT_FOUND`	404	사용자를 찾을 수 없습니다
`BOT_NOT_FOUND`	404	봇을 찾을 수 없습니다
`CREDENTIAL_NOT_FOUND`	404	인증 정보를 찾을 수 없습니다
`INVALID_CREDENTIALS`	400	잘못된 API 인증 정보입니다

코드	HTTP Status	메시지
`INVALID_PRICE_RANGE`	400	가격 범위가 올바르지 않습니다
`INSUFFICIENT_BALANCE`	400	잔액이 부족합니다
`EXCHANGE_API_ERROR`	502	거래소 API 오류가 발생했습니다
`ORDER_FAILED`	500	주문 실행에 실패했습니다
`INTERNAL_SERVER_ERROR`	500	서버 내부 오류가 발생했습니다

🚀 구현 우선순위

Phase 1: 기본 인프라 (1-2일)

- [ ] 프로젝트 초기 설정 (Express, TypeScript)
- [ ] 데이터베이스 연결 (MongoDB/PostgreSQL)
- [ ] 환경변수 설정
- [ ] 기본 미들웨어 (CORS, helmet, logger)
- [ ] 에러 핸들러

Phase 2: 인증 시스템 (1일)

- [ ] User 모델 생성
- [ ] 회원가입 API
- [ ] 로그인 API
- [ ] JWT 인증 미들웨어

Phase 3: 인증 정보 관리 (1일)

- [ ] Credential 모델 생성
- [ ] API 키 암호화/복호화 유틸리티
- [ ] 인증 정보 CRUD API
- [ ] 거래소 API 검증 로직

Phase 4: 봇 관리 기본 (2-3일)

- [ ] Bot, GridLevel, Trade 모델 생성
- [ ] 봇 생성 API
- [ ] 봇 조회 API (목록, 상세)
- [ ] 봇 시작/중지 API
- [ ] 봇 삭제 API

Phase 5: 거래소 연동 (2-3일)

- [ ] Upbit API 서비스 (ccxt 또는 직접)
- [ ] Binance API 서비스
- [ ] 티커 목록 조회
- [ ] 현재가 조회
- [ ] 주문 생성/조회/취소

Phase 6: 그리드 트레이딩 엔진 (3-4일)

- ☐ 그리드 계산 로직
- ☐ 초기 그리드 레벨 생성
- ☐ 초기 매수 주문 실행
- ☐ 주문 체결 모니터링
- ☐ 자동 재주문 로직
- ☐ 수익 계산 로직

Phase 7: 백그라운드 작업 (2일)

- ☐ 봇 모니터링 스케줄러
- ☐ 가격 업데이터
- ☐ 통계 업데이터
- ☐ PM2 설정

Phase 8: 거래 내역 & 통계 (1-2일)

- ☐ 거래 내역 조회 API
- ☐ 그리드 레벨 조회 API
- ☐ 성과 통계 API
- ☐ 페이지네이션

Phase 9: 테스트 & 최적화 (2-3일)

- ☐ 단위 테스트
- ☐ 통합 테스트
- ☐ 에러 처리 강화
- ☐ 로깅 개선
- ☐ 성능 최적화

Phase 10: 배포 준비 (1일)

- ☐ 환경변수 문서화
- ☐ README 작성
- ☐ API 문서 자동화 (Swagger)
- ☐ Docker 설정 (선택)

예상 총 개발 기간: 약 2-3주

🌐 환경변수 (.env)

```
# Server
NODE_ENV=development
PORT=3000
FRONTEND_URL=http://localhost:3009

# Database
```

```
MONGO_URI=mongodb://localhost:27017/grid-trading-bot
# 또는
POSTGRES_URI=postgresql://user:password@localhost:5432/grid_trading_bot

# JWT
JWT_SECRET=your-super-secret-jwt-key-change-this-in-production
JWT_EXPIRES_IN=1h

# Encryption
ENCRYPTION_KEY=your-32-byte-encryption-key-for-aes-256

# Redis (선택)
REDIS_HOST=localhost
REDIS_PORT=6379

# Logging
LOG_LEVEL=debug

# Rate Limiting
RATE_LIMIT_WINDOW_MS=60000
RATE_LIMIT_MAX_REQUESTS=100
```

📝 추가 권장 사항

1. 모의 거래 (Paper Trading)

- 실제 주문 전에 시뮬레이션 모드 제공
- `bot.mode: "live" | "paper"` 추가
- 리스크 없이 전략 테스트 가능

2. 알림 시스템

- 봇 상태 변경 시 이메일/SMS 알림
- 수익 목표 달성 시 알림
- 에러 발생 시 즉시 알림

3. 백테스팅

- 과거 데이터로 전략 검증
- 예상 수익률 계산
- 최적 그리드 설정 추천

4. WebSocket 실시간 통신

- 가격 변동 실시간 전송
- 주문 체결 실시간 알림
- 봇 상태 변화 실시간 반영

5. 로깅 & 모니터링

- Winston으로 구조화된 로그
- 모든 거래 기록
- 에러 스택 추적
- Sentry 또는 Datadog 연동 (선택)

6. API 문서 자동화

- Swagger/OpenAPI 사용
- `/api-docs` 엔드포인트에서 확인 가능

🔗 참고 자료

- Upbit API 문서
- Binance API 문서
- CCXT 라이브러리
- Express.js 공식 문서
- Mongoose 공식 문서

작성일: 2025-01-23 **버전:** 1.0.0 **담당자:** Backend Development Team