Documentation Technique - CRYPTOBU v1.0

Table des Matières

1. *Aperçu du Projet*
2. *Architecture Technique*
3. *Technologies Utilisées*
4. *Structure du Projet*
5. *Installation et Configuration*
6. *Base de Données*
7. *API Backend*
8. *Frontend React*
9. *Bot Discord*
10. *Sécurité et Authentification*
11. *Déploiement*
12. *Troubleshooting*

Aperçu du Projet

***CRYPTOBU** est une plateforme web complète de simulation de trading de cryptomonnaies développée avec une stack JavaScript moderne. Le projet permet aux utilisateurs de s'initier au trading dans un environnement sécurisé sans risque financier.*

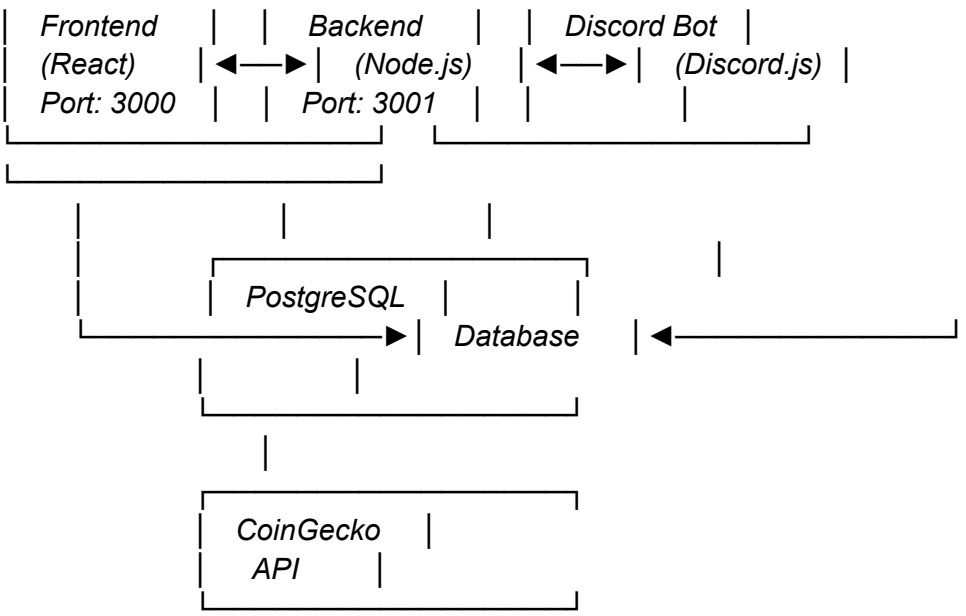
Objectifs Techniques

- *Interface utilisateur moderne et responsive*
- *API REST robuste avec authentification JWT*
- *Données en temps réel via WebSocket*
- *Intégration avec l'API CoinGecko*
- *Bot Discord avec commandes slash*
- *Gestion des devises (USD/EUR)*

Architecture Technique

Vue d'ensemble

┌─────────────────────┐ ┌─────────────────────┐
┌─────────────────────┐



Communication Inter-Services

- ***Frontend*** ↔ ***Backend****: HTTP/HTTPS REST API + WebSocket*
- ***Backend*** ↔ ***Database****: PostgreSQL avec pool de connexions*
- ***Backend*** ↔ ***CoinGecko****: HTTP API avec gestion du rate limiting*
- ***Discord Bot*** ↔ ***Backend****: Partage de certains modules (optionnel)*

Technologies Utilisées

Backend

- ***Node.js*** *v16+ - Runtime JavaScript*
- ***Express.js*** *- Framework web*
- ***PostgreSQL*** *- Base de données relationnelle*
- ***WebSocket (ws)*** *- Communication temps réel*
- ***JWT*** *- Authentification*
- ***bcryptjs*** *- Hachage des mots de passe*
- ***Joi*** *- Validation des données*
- ***Axios*** *- Client HTTP pour API externes*

Frontend

- ***React.js*** *v18+ - Interface utilisateur*
- ***Chart.js*** *- Graphiques interactifs*
- ***React Router*** *- Navigation*
- ***CSS3*** *avec variables personnalisées*
- ***WebSocket Client*** *- Temps réel*

Bot Discord

- ***Discord.js** v14 - SDK Discord*
- ***Canvas** - Génération d'images*
- ***Chart.js** - Graphiques pour Discord*

Outils de Développement

- ***npm** - Gestionnaire de paquets*
- ***nodemon** - Rechargement automatique*
- ***ESLint** - Linting JavaScript*

Structure du Projet

```
CRYPTOBU/
├── backend/                # API Node.js
│   ├── middleware/
│   │   └── auth.js         # Middleware d'authentification
│   ├── routes/
│   │   ├── auth.js         # Routes d'authentification
│   │   └── admin.js        # Panel d'administration
│   ├── server.js           # Serveur principal
│   ├── package.json
│   └── .env.example
├── frontend/               # Application React
│   ├── src/
│   │   ├── components/     # Composants React
│   │   ├── services/       # Services (API, auth)
│   │   ├── utils/          # Utilitaires
│   │   ├── App.js
│   │   └── index.js
│   ├── public/
│   └── package.json
├── discord-bot/            # Bot Discord
│   ├── bot.js              # Bot principal (avancé)
│   ├── bot-simple.js       # Version simplifiée
│   ├── deploy-commands.js  # Déploiement des commandes
│   └── package.json
├── database/
│   ├── schema.sql          # Schéma de la BDD
│   └── migrations.sql      # Migrations
├── logs/                   # Fichiers de logs
├── start-all.sh            # Script de démarrage
└── README.md
```

Installation et Configuration

Prérequis

- *Node.js v16+*
- *PostgreSQL v12+*
- *npm v8+*
- *Git*

Installation Complète

1. ***Cloner le projet***

```
git clone <repository-url>
cd CRYPTOBU
```

2. ***Base de Données***

```
# Installer PostgreSQL
sudo apt-get install postgresql postgresql-contrib

# Créer la base de données
sudo -u postgres psql
CREATE DATABASE crypto_trading;
CREATE USER crypto_user WITH PASSWORD 'crypto_password';
GRANT ALL PRIVILEGES ON DATABASE crypto_trading TO crypto_user;
\q

# Importer le schéma
psql -U crypto_user -d crypto_trading -f database/schema.sql
```

3. ***Backend***

```
cd backend
npm install

# Configuration environnement
cp .env.example .env
# Éditer .env avec vos paramètres
```

4. ***Frontend***

```
cd ../frontend
npm install
```

5. **Bot Discord**

```
cd ../discord-bot
npm install

# Configuration bot Discord
# Créer une application sur https://discord.com/developers/applications
# Ajouter le token dans .env
```

Variables d'Environnement

Backend (.env)

```
# Base de données
DB_HOST=localhost
DB_PORT=5432
DB_NAME=crypto_trading
DB_USER=crypto_user
DB_PASSWORD=crypto_password

# JWT
JWT_SECRET=your-super-secret-key-change-in-production

# Admin
ADMIN_PASSWORD=admin123

# API
PORT=3001
```

Discord Bot (.env)

```
DISCORD_BOT_TOKEN=your_discord_bot_token
DISCORD_CLIENT_ID=your_discord_client_id
```

Base de Données

Schéma Principal

Table `users`

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
```

```
    password_hash VARCHAR(255) NOT NULL,
    balance DECIMAL(15, 2) DEFAULT 10000.00,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Table `portfolio`

```
CREATE TABLE portfolio (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    crypto_id VARCHAR(50) NOT NULL,
    quantity DECIMAL(20, 8) NOT NULL,
    avg_buy_price DECIMAL(15, 2) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, crypto_id)
);
```

Table `transactions`

```
CREATE TABLE transactions (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    crypto_id VARCHAR(50) NOT NULL,
    type VARCHAR(10) CHECK (type IN ('buy', 'sell')),
    quantity DECIMAL(20, 8) NOT NULL,
    price DECIMAL(15, 2) NOT NULL,
    total_amount DECIMAL(15, 2) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Index et Optimisations

```
-- Index pour les performances
CREATE INDEX idx_portfolio_user_id ON portfolio(user_id);
CREATE INDEX idx_transactions_user_id ON transactions(user_id);
CREATE INDEX idx_transactions_created_at ON transactions(created_at);

-- Triggers pour updated_at
CREATE TRIGGER update_users_updated_at BEFORE UPDATE ON users
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();
```

API Backend

Architecture API

Structure des Routes

```
/api/
├── auth/          # Authentification
│   ├── POST /login
│   ├── POST /register
│   ├── GET  /me
│   └── POST /logout
├── prices         # Prix des cryptos
├── history/:id    # Données historiques
├── portfolio      # Portfolio utilisateur
├── buy            # Achat crypto
├── sell           # Vente crypto
├── admin/         # Administration
│   ├── GET /stats
│   ├── GET /users
│   └── GET /transactions
└── health         # Status de l'API
```

Authentification JWT

Middleware d'authentification

```
const authenticateToken = async (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ error: 'Token d\'accès requis' });
  }

  try {
    const decoded = jwt.verify(token, JWT_SECRET);
    const user = await getUserById(decoded.userId);
    req.user = user;
    next();
  } catch (error) {
    return res.status(403).json({ error: 'Token invalide' });
  }
};
```

Gestion WebSocket

Connexion et diffusion

```
const wss = new WebSocket.Server({ port: 8080 });

// Diffuser les prix à tous les clients connectés
function broadcastPrices(prices) {
  const message = JSON.stringify({
    type: 'price_update',
    data: prices,
    timestamp: new Date().toISOString()
  });

  clients.forEach(client => {
    if (client.readyState === WebSocket.OPEN) {
      client.send(message);
    }
  });
}
```

Rate Limiting CoinGecko

Gestion du cache et des appels API

```
let priceCache = new Map();
let lastApiCall = 0;
const MIN_DELAY_BETWEEN_CALLS = 60000; // 1 minute

async function fetchCryptoPrices() {
  const now = Date.now();

  if (now - lastApiCall < MIN_DELAY_BETWEEN_CALLS) {
    return Object.fromEntries(priceCache);
  }

  try {
    const response = await axios.get('https://api.coingecko.com/api/v3/simple/price', {
      params: {
        ids: 'bitcoin,ethereum,cardano,polkadot,chainlink',
        vs_currencies: 'usd,eur',
        include_24hr_change: 'true'
      },
      timeout: 15000
    });

    lastApiCall = now;
    // Mettre à jour le cache...
  } catch (error) {
    // Gestion d'erreur et fallback...
  }
```

```
}
```

Frontend React

Architecture des Composants

```
src/
├── components/
│   ├── AnimatedBackground.js    # Arrière-plan animé
│   ├── CryptoChart.js           # Graphiques interactifs
│   ├── CryptoSelector.js        # Sélecteur de crypto
│   ├── CurrencySwitch.js        # Basculeur USD/EUR
│   ├── DiscordInvite.js         # Section Discord
│   ├── Login.js                 # Connexion
│   ├── Portfolio.js             # Portfolio utilisateur
│   ├── PriceList.js             # Liste des prix
│   ├── Register.js              # Inscription
│   └── TradingPanel.js          # Panel de trading
├── services/
│   └── authService.js           # Service d'authentification
├── utils/
│   └── currencyUtils.js         # Utilitaires de devise
├── App.js                       # Composant principal
└── App.css                      # Styles personnalisés
```

Gestion des États

État principal de l'application

```
const [prices, setPrices] = useState({});
const [selectedCrypto, setSelectedCrypto] = useState('bitcoin');
const [currency, setCurrency] = useState('usd');
const [user, setUser] = useState(null);
const [portfolio, setPortfolio] = useState({
  holdings: [],
  total_crypto_value: 0,
  cash_balance: 0
});
```

Communication WebSocket

```
useEffect(() => {
  const websocket = new WebSocket('ws://localhost:8080');
```

```
  websocket.onmessage = (event) => {
    const data = JSON.parse(event.data);
    if (data.type === 'price_update') {
      setPrices(data.data);
    }
  };

  return () => websocket.close();
}, []);
```

Graphiques Interactifs

Configuration Chart.js

```
const options = {
  responsive: true,
  plugins: {
    zoom: {
      pan: { enabled: true, mode: 'xy' },
      zoom: { wheel: { enabled: true }, mode: 'xy' }
    }
  },
  scales: {
    x: { type: 'time' },
    y: {
      ticks: {
        callback: function(value) {
          return formatPrice(value, currency);
        }
      }
    }
  }
};
```

Bot Discord

Architecture Bot

Structure principale

```
const { Client, GatewayIntentBits, SlashCommandBuilder } = require('discord.js');

const client = new Client({
  intents: [GatewayIntentBits.Guilds]
```

```
});

// Commandes disponibles
const commands = [
  new SlashCommandBuilder()
    .setName('crypto')
    .setDescription('Afficher le prix d\'une cryptomonnaie'),
  // ... autres commandes
];
```

Commandes Principales

1. `/crypto <coin>` - Prix détaillé d'une crypto
2. `/portfolio` - Vue d'ensemble des cryptos
3. `/chart <coin> [days]` - Graphique de prix
4. `/compare <crypto1> <crypto2>` - Comparaison
5. `/help` - Aide complète

Génération de Graphiques

```
async function createPriceChart(cryptoId, days = 7) {
  const canvas = createCanvas(800, 400);
  const ctx = canvas.getContext('2d');

  // Configuration Chart.js pour Discord
  const chart = new Chart(ctx, {
    type: 'line',
    data: chartData,
    options: discordOptimizedOptions
  });

  return canvas.toBuffer('image/png');
}
```

Sécurité et Authentification

Mesures de Sécurité Implémentées

1. **Authentification JWT**
    - Tokens avec expiration (7 jours)
    - Validation côté serveur
    - Refresh automatique

Hachage des Mots de Passe

```
 const saltRounds = 12;
const hashedPassword = await bcrypt.hash(password, saltRounds);
```

2.

Validation des Données

```
 const registerSchema = Joi.object({
  username: Joi.string().alphanum().min(3).max(30).required(),
  email: Joi.string().email().required(),
  password: Joi.string().min(6).required(),
});
```

3.

Protection CORS

```
 app.use(cors({
  origin: process.env.FRONTEND_URL || 'http://localhost:3000',
  credentials: true
}));
```

4.
5. ***Rate Limiting***

    - *Limitation des appels API externes*
    - *Protection contre le spam*

Recommandations Production

- *Utiliser HTTPS en production*
- *Configurer un reverse proxy (Nginx)*
- *Mettre en place des logs de sécurité*
- *Sauvegardes régulières de la BDD*
- *Monitoring des performances*

Déploiement

Déploiement Local

Script de démarrage automatique

./start-all.sh

Ce script :

- *Nettoie les anciens processus*
- *Démarre le backend (port 3001)*
- *Démarre le frontend (port 3000)*
- *Démarre le bot Discord*
- *Fournit un monitoring temps réel*

Déploiement Production

1. Serveur (Ubuntu/Debian)

```
# Installer Node.js
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs

# Installer PostgreSQL
sudo apt-get install postgresql postgresql-contrib

# Installer PM2 pour la gestion des processus
npm install -g pm2
```

2. Configuration PM2

```
// ecosystem.config.js
module.exports = {
  apps: [
    {
      name: 'cryptobu-backend',
      script: './backend/server.js',
      env: {
        NODE_ENV: 'production',
        PORT: 3001
      }
    },
    {
      name: 'cryptobu-discord-bot',
      script: './discord-bot/bot.js',
      env: {
        NODE_ENV: 'production'
      }
    }
  ]
};
```

3. Build Frontend

cd frontend
npm run build
 # Servir avec Nginx ou hébergement statique

Variables d'Environnement Production

NODE_ENV=production
JWT_SECRET=<strong-random-secret>
DB_HOST=<production-db-host>
DB_PASSWORD=<strong-password>
ADMIN_PASSWORD=<strong-admin-password>

Troubleshooting

Problèmes Courants

1. Port déjà utilisé

 # Identifier le processus
lsof -i :3001

 # Tuer le processus
kill -9 <PID>

2. Erreur de connexion PostgreSQL

 # Vérifier le service
sudo systemctl status postgresql

 # Redémarrer si nécessaire
sudo systemctl restart postgresql

3. Bot Discord non connecté

- *Vérifier le token dans .env*
- *Vérifier les permissions du bot*
- *Consulter les logs :* `tail -f logs/discord-bot.log`

4. Problème WebSocket

- *Vérifier que le port 8080 est libre*
- *Vérifier la configuration firewall*

Logs et Monitoring

Consulter les logs

```
# Tous les logs
tail -f logs/*.log

# Backend seulement
tail -f logs/backend.log

# Frontend seulement
tail -f logs/frontend.log
```

Vérifier l'état des services

```
# Ports utilisés
lsof -i :3000 -i :3001 -i :8080

# Processus Node.js
ps aux | grep node

# Espace disque
df -h
```

Performance

Optimisations recommandées

- *Activer la compression gzip*
- *Mettre en cache les réponses API*
- *Optimiser les requêtes SQL*
- *Utiliser un CDN pour les assets statiques*

Maintenance et Évolution

Sauvegarde Base de Données

```
# Sauvegarde
pg_dump -U crypto_user crypto_trading > backup_$(date +%Y%m%d).sql

# Restauration
psql -U crypto_user crypto_trading < backup_20241220.sql
```

Mises à Jour

Dépendances

npm audit
npm update

1.

Base de Données

-- Utiliser des migrations pour les changements de schéma
ALTER TABLE users ADD COLUMN last_login TIMESTAMP;

2.
3. ***Features Discord***

    - *Nouvelles commandes slash*
    - *Amélioration des graphiques*
    - *Notifications push*

Améliorations Futures

- *Application mobile React Native*
- *Trading algorithmique simulé*
- *Chatbot IA intégré*
- *Analyse technique avancée*
- *Support multi-langues*
