👟 SneakerDropBot - Complete Sneaker Tracking & Alert System

A fully functional Telegram bot that tracks sneaker restocks, price drops, and resell opportunities across major retailers. Built with real-time monitoring, comprehensive APIs, payment integration, and affiliate tracking.

🚀 Features

Core Features

- 🔁 **Restock Alerts**: Instant notifications when sneakers come back in stock
- 💸 **Price Drop Tracking**: Monitor price changes across all retailers
- 📈 **Resell Deal Analysis**: Identify profitable flip opportunities
- ⚡ **Real-time Monitoring**: Continuous scraping of major sneaker retailers
- 👥 **User Management**: Free and premium tiers with different limits
- 💳 **Payment Integration**: Stripe-powered subscription system
- 💰 **Affiliate Revenue**: Built-in affiliate link management and tracking

Supported Retailers

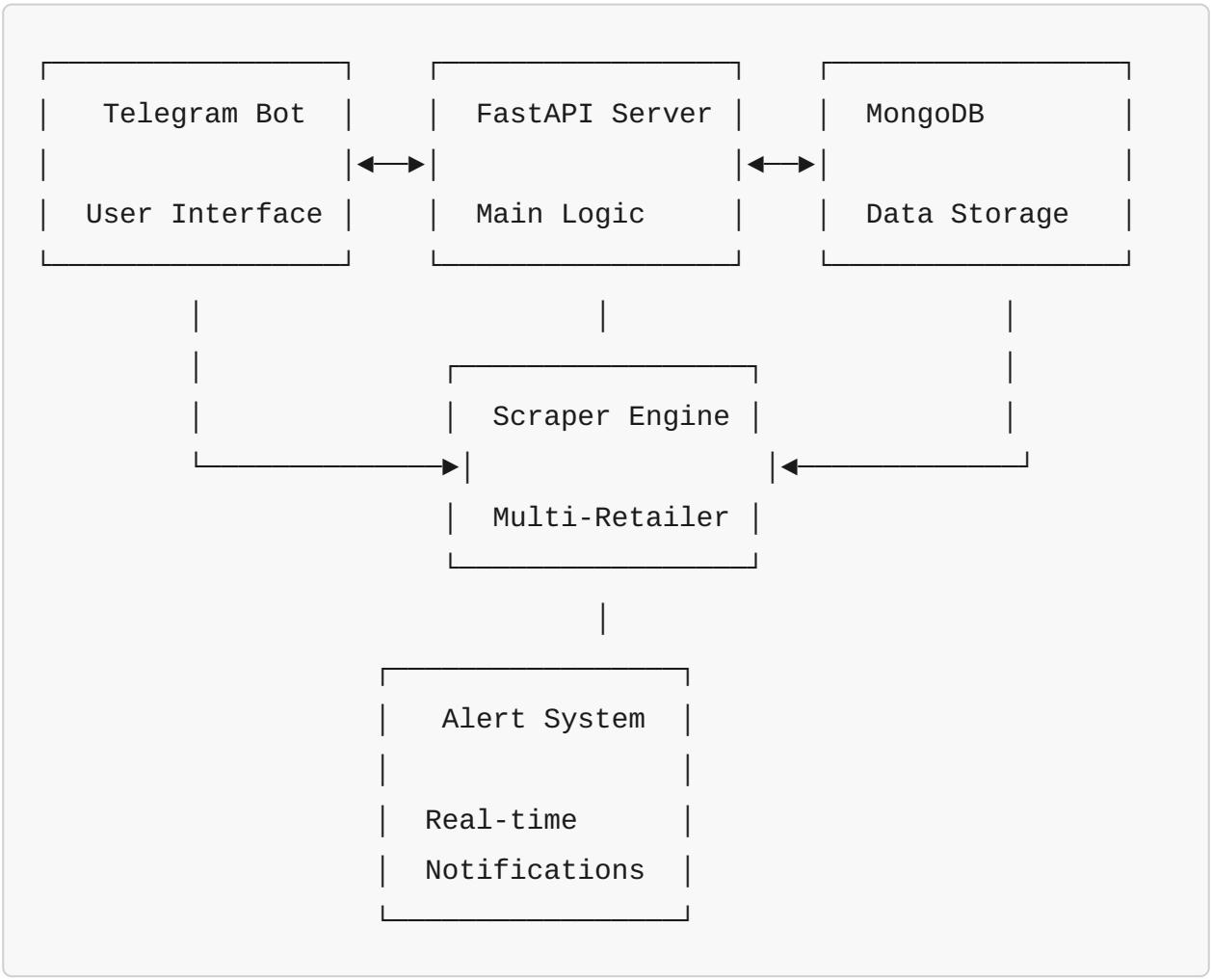
- **Nike** (including SNKRS)
- **Adidas**
- **FootLocker**
- **Finish Line**
- **Champs Sports**
- **JD Sports**

- **StockX** (resell market)
- **GOAT** (resell market)
- **Stadium Goods**

Premium Features

- Unlimited sneaker tracking
- Instant priority alerts
- Flip margin analysis
- Early access notifications
- Price history & trends
- Advanced filtering options

🏗️ Architecture



🛠️ Quick Setup

Prerequisites

- Docker & Docker Compose
- Telegram Bot Token (from @BotFather)
- Stripe Account (for payments)
- MongoDB Atlas (or local MongoDB)

1. Clone & Configure

```
# Clone the repository
git clone https://github.com/yourusername/sneakerdropbot.git
cd sneakerdropbot

# Copy environment template
cp .env.example .env

# Edit .env with your configuration
nano .env
```

2. Essential Configuration

Edit `.env` with your credentials:

```
# Telegram Bot
TELEGRAM_BOT_TOKEN=your_bot_token_from_botfather

# Database
MONGODB_URL=mongodb://localhost:27017

# Payments (Stripe)
STRIPE_SECRET_KEY=sk_test_your_stripe_key
STRIPE_PUBLISHABLE_KEY=pk_test_your_stripe_key

# Admin Access
ADMIN_IDS=your_telegram_user_id
```

3. Deploy with Docker

```
# Start all services
docker-compose up -d

# Check logs
docker-compose logs -f sneakerdropbot

# Check status
docker-compose ps
```

4. Verify Installation

1. **Health Check**: Visit http://localhost:8000/health
2. **API Documentation**: Visit http://localhost:8000/docs
3. **Monitoring**: Visit http://localhost:3000 (Grafana)
4. **Bot Test**: Message your bot on Telegram with `/start`

📱 Bot Commands

User Commands

- `/start` - Start the bot and see main menu
- `/track` - Add a new sneaker to track
- `/list` - View your tracked sneakers
- `/remove` - Remove a tracked sneaker
- `/status` - Check your account status
- `/trending` - See trending sneakers
- `/market <sneaker>` - Get market analysis

- `/premium` - Upgrade to premium

Admin Commands

- `/admin` - Admin panel
- `/broadcast <message>` - Send to all users
- `/stats` - Bot statistics
- `/setdrop <sneaker>` - Manual drop alert

🔧 Configuration

Environment Variables

Core Settings

```
# Application
ENVIRONMENT=production
DEBUG=false
LOG_LEVEL=INFO

# Monitoring
MONITORING_INTERVAL=300    # 5 minutes
SCRAPING_INTERVAL=600      # 10 minutes
ALERT_COOLDOWN=300         # 5 minutes

# Limits
FREE_ALERTS_PER_DAY=5
PREMIUM_ALERTS_PER_DAY=1000
```

Retailer APIs

```
# Add your API keys for better access
NIKE_API_KEY=your_nike_key
ADIDAS_API_KEY=your_adidas_key
FOOTLOCKER_API_KEY=your_footlocker_key
```

Affiliate Programs

```
# Your affiliate codes
NIKE_AFFILIATE_CODE=your_code
ADIDAS_AFFILIATE_CODE=your_code
FOOTLOCKER_AFFILIATE_CODE=your_code
```

Feature Flags

```
# Enable/disable features
ENABLE_RESELL_TRACKING=true
ENABLE_PRICE_HISTORY=true
ENABLE_FLIP_ANALYSIS=true
ENABLE_EARLY_ACCESS=true
```

🏪 Retailer Integration

Nike/SNKRS

- Uses Nike's internal APIs
- Supports size availability tracking
- Monitors SNKRS exclusive releases

StockX Integration

```
# Example usage
from scrapers.stockx_scraper import StockXScraper

async with StockXScraper() as scraper:
    products = await scraper.search_products("Jordan 4 Bred")
    market_data = await scraper.get_market_data(products[0].url)
```

Custom Retailer

```
# Add new retailer
from scrapers.base_scraper import BaseScraper

class CustomRetailerScraper(BaseScraper):
    async def search_products(self, keyword: str):
        # Implementation
        pass
```

💳 Payment Integration

Stripe Setup

1. Create Stripe account
2. Get API keys from dashboard
3. Configure webhook endpoint: `https://yourdomain.com/webhook/stripe`
4. Add webhook events:
   - `checkout.session.completed`
   - `invoice.payment_succeeded`
   - `customer.subscription.deleted`

Subscription Flow

```
# Create subscription
payment_url = await payment_processor.create_subscription_payment(
    user_id=user_id,
    plan_type="monthly"
)

# Handle webhook
success = await payment_processor.handle_webhook(payload,
signature)
```

📊 Monitoring & Analytics

Built-in Monitoring

- **Health Checks**: `/health` endpoint
- **Metrics**: Prometheus integration
- **Dashboards**: Grafana dashboards
- **Logging**: Structured logging with loguru
- **Error Tracking**: Sentry integration

Access Monitoring

- **Grafana**: http://localhost:3000 (admin/admin)
- **Prometheus**: http://localhost:9090
- **Flower** (Celery): http://localhost:5555
- **Kibana**: http://localhost:5601

Key Metrics

- User engagement rates
- Alert delivery success
- Scraper performance
- Revenue tracking
- Affiliate conversions

🚀 Deployment

Production Deployment

Docker Swarm

```
# Initialize swarm
docker swarm init

# Deploy stack
docker stack deploy -c docker-compose.prod.yml sneakerbot
```

Kubernetes

```
# Apply configurations
kubectl apply -f k8s/

# Check status
kubectl get pods -n sneakerbot
```

Cloud Deployment

- **AWS**: Use ECS/EKS with RDS and ElastiCache

- **Google Cloud**: Use GKE with Cloud SQL and Memorystore
- **Azure**: Use AKS with Azure Database and Redis Cache

Environment-Specific Configs

Development

```
docker-compose -f docker-compose.dev.yml up
```

Production

```
docker-compose -f docker-compose.prod.yml up -d
```

Staging

```
docker-compose -f docker-compose.staging.yml up -d
```

🔒 Security

Best Practices

- Store secrets in environment variables
- Use HTTPS for webhooks
- Implement rate limiting
- Monitor for suspicious activity
- Regular security updates

API Security

```
# Rate limiting
from slowapi import Limiter

limiter = Limiter(key_func=lambda: "global")

@app.get("/api/search")
@limiter.limit("10/minute")
async def search_endpoint():
    pass
```

🧪 Testing

Run Tests

```
# Unit tests
pytest tests/

# Integration tests
pytest tests/integration/

# API tests
pytest tests/api/

# Load tests
locust -f tests/load/locustfile.py
```

Test Configuration

```
# Test environment
TEST_MODE=true
MOCK_SCRAPERS=true
MOCK_PAYMENTS=true
```

📈 Scaling

Horizontal Scaling

- Multiple bot instances
- Load balancer (nginx)
- Database clustering
- Redis cluster

Performance Optimization

- Database indexing
- Caching strategies
- Connection pooling
- Async processing

Resource Requirements

Minimum (Development)

- 2 CPU cores
- 4GB RAM
- 20GB storage

Recommended (Production)

- 4+ CPU cores
- 8+ GB RAM
- 100+ GB SSD storage

🤝 Contributing

Development Setup

```
# Clone repository
git clone https://github.com/yourusername/sneakerdropbot.git

# Install dependencies
pip install -r requirements-dev.txt

# Setup pre-commit hooks
pre-commit install

# Run in development mode
python main.py
```

Code Standards

- Black formatting
- Flake8 linting
- Type hints with mypy
- Comprehensive testing

📚 API Documentation

Interactive Docs

- **Swagger UI**: http://localhost:8000/docs
- **ReDoc**: http://localhost:8000/redoc

Key Endpoints

```
# Search sneakers
GET /api/search/{keyword}

# Market analysis
GET /api/market/{sneaker_name}

# Trending sneakers
GET /api/trending

# Health check
GET /health

# Statistics
GET /stats
```

🐛 Troubleshooting

Common Issues

Bot Not Responding

```
# Check bot status
docker-compose logs sneakerdropbot

# Verify token
echo $TELEGRAM_BOT_TOKEN

# Test webhook
curl -X POST https://api.telegram.org/bot$TOKEN/getMe
```

Scrapers Failing

```
# Check scraper health
curl http://localhost:8000/health

# View scraper logs
docker-compose logs -f sneakerdropbot | grep scraper

# Test individual scraper
python -c "from scrapers.nike_scraper import NikeScraper;
print('OK')"
```

Database Issues

```
# Check MongoDB
docker-compose exec mongodb mongosh

# Check connections
docker-compose logs mongodb
```

Performance Issues

- Monitor resource usage
- Check database indexes
- Optimize scraping intervals
- Review cache hit rates

📄 License

This project is licensed under the MIT License - see the LICENSE file for details.

🙏 Acknowledgments

- Telegram Bot API
- FastAPI framework
- MongoDB for data storage
- Stripe for payments
- All the amazing sneaker community

📞 Support

- **Documentation**: Wiki

- **Issues**: GitHub Issues
- **Telegram**: @SneakerDropSupport
- **Email**: support@sneakerdropbot.com

Built with ❤️ for sneakerheads by sneakerheads

🚀 Quick Start Checklist

- [ ] Clone repository
- [ ] Copy `.env.example` to `.env`
- [ ] Get Telegram bot token from @BotFather
- [ ] Configure Stripe account
- [ ] Set up MongoDB
- [ ] Run `docker-compose up -d`
- [ ] Test bot with `/start` command
- [ ] Check health endpoint
- [ ] Configure affiliate codes
- [ ] Set up monitoring
- [ ] Deploy to production

Happy sneaker hunting! 👟✨