ðŸ¦ RoseCoin - Complete Mining Platform

Comprehensive Installation & Configuration Guide

Author: Olasunkanmi Julius Agbajelola
Version: 1.0
Last Updated: July 2025

Table of Contents

Overview

RoseCoin is a comprehensive gamified cryptocurrency mining platform built with Flask/Python. It features:

- **Tap-to-Mine System: Interactive mining with XP progression**
- **Social Media Tasks: Multi-platform task completion system**
- **Referral System: Multi-level referral earnings**
- **Social Media Promotion: Campaign management and promotion platform**
- **Airdrop System: Automated coin distribution campaigns**
- **Admin Panel: Complete administrative control**
- **PWA Support: Progressive Web App capabilities**
- **Multi-language: Internationalization ready**

Key Technologies

- **Backend: Flask 3.1.1, SQLAlchemy, Python 3.9+**
- **Frontend: Bootstrap 5, JavaScript, PWA**
- **Database: SQLite (development), PostgreSQL (production)**

- **Security: CSRF protection, password hashing, session management**

System Requirements

Minimum Requirements

- **Python: 3.9 or higher**
- **RAM: 512MB minimum, 2GB recommended**
- **Storage: 5GB minimum, 20GB recommended**
- **CPU: 1 vCPU minimum, 2 vCPU recommended**

Recommended Hosting Specifications

- **RAM: 4GB or higher**
- **Storage: SSD with 50GB+ space**
- **Bandwidth: Unlimited or 1TB+**
- **SSL: SSL certificate support**
- **Database: PostgreSQL 12+ support**

Installation Guide

Step 1: Download and Extract Files

1. **Download the complete source code package**
2. **Extract all files to your desired directory**
3. **Ensure all file permissions are correctly set**

Step 2: Install Python Dependencies

```
# Navigate to the project directory
cd rosecoin-platform

# Install required packages
pip install -r requirements.txt
```

Step 3: Database Setup

```
# Initialize the database
python app.py
```

The system will automatically: - Create all required database tables - Set up the admin user (demo/demo123) - Initialize default data

Step 4: Basic Configuration

1. **Edit environment variables in your hosting platform**
2. **Configure database settings**
3. **Set up email service (optional but recommended)**
4. **Configure security keys**

Step 5: Launch Application

```
# For development
python main.py

# For production (recommended)
gunicorn --bind 0.0.0.0:5000 --reuse-port --reload main:app
```

Platform-Specific Installation

Replit (Recommended)

Why Replit? - Zero setup configuration - Built-in database support - Automatic dependency management - Free hosting tier available - Easy deployment options

Installation Steps:

1. **Fork the Template**
   - **Login to Replit**
   - **Import the project files**
   - **Replit will automatically detect Flask and install dependencies**
2. **Configure Environment**
   - **Set environment variables in Replit Secrets**
   - **Configure database URL if using external database**
3. **Run the Application**
   - **Click the “Run” button**
   - **Application will be available at your Replit URL**
4. **Deploy to Production**
   - **Use Replit’s deployment feature**
   - **Configure custom domain if needed**

VPS/Dedicated Server

Ubuntu/Debian Installation:

```
# Update system packages
sudo apt update && sudo apt upgrade -y

# Install Python and pip
sudo apt install python3 python3-pip python3-venv -y

# Install PostgreSQL (for production)
sudo apt install postgresql postgresql-contrib -y

# Create application directory
sudo mkdir /var/www/rosecoin
sudo chown $USER:$USER /var/www/rosecoin

# Extract application files
cd /var/www/rosecoin
# Upload and extract your files here

# Install dependencies
pip3 install -r requirements.txt

# Set up systemd service (optional)
sudo nano /etc/systemd/system/rosecoin.service
```

Systemd Service File:

```
[Unit]
Description=rosecoin Flask App
After=network.target

[Service]
User=www-data
Group=www-data
WorkingDirectory=/var/www/rosecoin
Environment="PATH=/var/www/rosecoin/venv/bin"
ExecStart=/var/www/rosecoin/venv/bin/gunicorn --bind 0.0.0.0:5000 main:app
Restart=always

[Install]
WantedBy=multi-user.target
```

Shared Hosting

Requirements: - Python 3.9+ support - pip installation capability - Database access (SQLite or PostgreSQL) - SSL certificate support

Installation Steps: 1. Upload all files via FTP/cPanel File Manager 2. Install dependencies using hosting control panel 3. Configure database connection 4. Set up domain pointing to main.py 5. Configure SSL certificate

---

Hosting Platform Selection

Recommended Hosting Platforms

1. Replit (Best for Beginners)

Pros: - Zero configuration - Built-in database - Free tier available - Easy deployment - Automatic backups

Cons: - Limited resources on free tier - Replit-specific URL structure

Best For: Development, testing, small-scale deployment

2. DigitalOcean Droplets

Pros: - Full control over server - Scalable resources - SSD storage - Good documentation

Cons: - Requires server management knowledge - No managed services

Best For: Medium to large-scale deployments

3. Heroku

Pros: - Easy deployment - Managed database options - Built-in SSL - Automatic scaling

Cons: - More expensive - Less control

Best For: Quick deployment, managed hosting preference

4. AWS/Google Cloud

Pros: - Enterprise-grade infrastructure - Global CDN - Managed services - High availability

Cons: - Complex pricing - Requires cloud knowledge

Best For: Large-scale, enterprise deployments

Hosting Selection Criteria

Traffic Volume: - Low (< 1,000 users): Replit, shared hosting - Medium (1,000-10,000 users): VPS, cloud instances - High (10,000+ users): Dedicated servers, cloud scaling

Budget Considerations: - $0-10/month: Replit, shared hosting - $10-50/month: VPS, small cloud instances - $50+/month: Dedicated servers, managed cloud

Technical Expertise: - Beginner: Replit, managed hosting - Intermediate: VPS, cloud platforms - Advanced: Dedicated servers, custom setups

---

Database Configuration & Migration

Development Database (SQLite)

The application uses SQLite by default for development:

```
# Default configuration in app.py
database_url = os.environ.get("DATABASE_URL") or "sqlite:///rosecoin.db"
```

Location: `instance/rosecoin.db`

Production Database (PostgreSQL)

Setting up PostgreSQL

On Ubuntu/Debian:

```
# Install PostgreSQL
sudo apt install postgresql postgresql-contrib

# Create database and user
sudo -u postgres psql
CREATE DATABASE rosecoin;
CREATE USER dscuser WITH PASSWORD 'your_secure_password';
GRANT ALL PRIVILEGES ON DATABASE rosecoin TO dscuser;
\q
```

Environment Configuration:

```
# Set database URL
export DATABASE_URL="postgresql://dscuser:your_secure_password@localhost/rosecoin"
```

Database Migration

From SQLite to PostgreSQL

1. ## Export SQLite Data:

```
# Using the built-in backup system
python -c "
from app import app, db
from models.user import User
import json

with app.app_context():
    users = User.query.all()
    # Export your data to JSON
"
```

2. ## Import to PostgreSQL:

```
# Set PostgreSQL as database URL
export DATABASE_URL="postgresql://user:pass@localhost/dbname"

# Run the application to create tables
python app.py

# Import your data using admin panel or custom script
```

Automated Migration Script

```
# migration_script.py
import os
import json
from datetime import datetime
from app import app, db
from models.user import User
from models.task import Task
from models.mining import MiningSession

def export_data():
    """Export data from current database"""
    with app.app_context():
        data = {
            'users': [user.to_dict() for user in User.query.all()],
            'tasks': [task.to_dict() for task in Task.query.all()],
            'mining_sessions': [session.to_dict() for session in MiningSession.query.all()],
            'exported_at': datetime.utcnow().isoformat()
        }

        with open('database_backup.json', 'w') as f:
            json.dump(data, f, indent=2)

        print("Data exported successfully!")

def import_data():
    """Import data to new database"""
    with app.app_context():
        with open('database_backup.json', 'r') as f:
            data = json.load(f)

        # Import users
        for user_data in data['users']:
            user = User(**user_data)
            db.session.add(user)

        # Import other data...

        db.session.commit()
        print("Data imported successfully!")

if __name__ == "__main__":
    import sys
    if len(sys.argv) > 1 and sys.argv[1] == "export":
        export_data()
    elif len(sys.argv) > 1 and sys.argv[1] == "import":
        import_data()
    else:
        print("Usage: python migration_script.py [export|import]")
```

Database Backup Strategy

Automated Backups

The application includes built-in backup functionality in the admin panel:

```
# Admin panel backup feature
@admin_bp.route('/database/backup')
@login_required
@admin_required
def backup_database():
    # Creates JSON backup of all data
    # Downloadable via admin panel
```

Manual Backup Commands

SQLite:

```
# Create backup
cp instance/rosecoin.db backup_$(date +%Y%m%d_%H%M%S).db

# Restore backup
cp backup_20250719_120000.db instance/rosecoin.db
```

PostgreSQL:

```
# Create backup
pg_dump rosecoin > backup_$(date +%Y%m%d_%H%M%S).sql

# Restore backup
psql rosecoin < backup_20250719_120000.sql
```

Website Configuration

Environment Variables

Required Environment Variables

```
# Session Security
SESSION_SECRET="your-super-secret-key-change-this-in-production"

# Database Configuration
DATABASE_URL="sqlite:///rosecoin.db"  # or PostgreSQL URL

# Email Configuration (Optional)
SMTP_SERVER="smtp.gmail.com"
SMTP_PORT="587"
SMTP_USERNAME="your-email@gmail.com"
SMTP_PASSWORD="your-app-password"
SMTP_USE_TLS="true"

# File Upload Configuration
MAX_CONTENT_LENGTH="16777216"  # 16MB in bytes
UPLOAD_FOLDER="uploads"
```

Setting Environment Variables

On Replit: 1. Go to your Repl 2. Click on “Secrets” tab in sidebar 3. Add each variable with its value

On VPS/Linux:

```
# Add to ~/.bashrc or /etc/environment
export SESSION_SECRET="your-secret-key"
export DATABASE_URL="your-database-url"

# Or create .env file (not recommended for production)
echo "SESSION_SECRET=your-secret-key" > .env
```

On Windows:

`# Set environment variables set SESSION_SECRET=your-secret-key set DATABASE_URL=your-database-url`

Application Settings

Admin Panel Settings

Access the admin panel at `/admin` and configure:

1. ## Platform Settings
   - ## App Name: “RoseCoin”
   - ## Welcome Message
   - ## Maintenance Mode
   - ## Registration Settings
2. ## Mining Configuration
   - ## Base mining rate
   - ## Level multipliers
   - ## Daily mission rewards
   - ## Event bonuses
3. ## Task Settings
   - ## Available platforms
   - ## Reward amounts
   - ## Verification settings

- **Auto-approval thresholds**

4. **Referral System**
   - **Commission rates**
   - **Maximum referral levels**
   - **Bonus thresholds**
5. **Withdrawal Settings**
   - **Minimum withdrawal amounts**
   - **Processing fees**
   - **Supported payment methods**

SSL Configuration

Let’s Encrypt (Free SSL)

```
# Install Certbot
sudo apt install certbot python3-certbot-nginx

# Obtain certificate
sudo certbot --nginx -d yourdomain.com

# Auto-renewal
sudo crontab -e
# Add: 0 12 * * * /usr/bin/certbot renew --quiet
```

Cloudflare SSL (Recommended)

1. **Add your domain to Cloudflare**
2. **Update nameservers**
3. **Enable “Full (Strict)” SSL mode**
4. **Enable “Always Use HTTPS”**

Nginx Configuration

 # /etc/nginx/sites-available/rosecoin server { listen 80; server_name yourdomain.com www.yourdomain.com; return 301 https://$server_name$request_uri; } server { listen 443 ssl http2; server_name yourdomain.com www.yourdomain.com; ssl_certificate /path/to/certificate.crt; ssl_certificate_key /path/to/private.key; location / { proxy_pass http://127.0.0.1:5000; proxy_set_header Host $host; proxy_set_header X-Real-IP $remote_addr; proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for; proxy_set_header X-Forwarded-Proto $scheme; } location /static/ { alias /var/www/rosecoin/static/; expires 1y; add_header Cache-Control "public, immutable"; } location /uploads/ { alias /var/www/rosecoin/uploads/; expires 1y; add_header Cache-Control "public"; } }

File Modification Guide

Project Structure

rosecoin/ â”œâ”€â”€ app.py # Main application file â”œâ”€â”€ main.py # Entry point â”œâ”€â”€ requirements.txt # Python dependencies â”œâ”€â”€ forms.py # WTForms definitions â”œâ”€â”€ core/ # Core modules â”‚ â”œâ”€â”€ auth.py # Authentication â”‚ â”œâ”€â”€ mining.py # Mining system â”‚ â”œâ”€â”€ tasks.py # Task system â”‚ â”œâ”€â”€ admin.py # Admin panel â”‚ â””â”€â”€ promotions.py # Promotion system â”œâ”€â”€ models/ # Database models â”‚ â”œâ”€â”€ user.py # User model â”‚ â”œâ”€â”€ task.py # Task model â”‚ â”œâ”€â”€ mining.py # Mining models â”‚ â””â”€â”€ ... â”œâ”€â”€ templates/ # HTML templates â”‚ â”œâ”€â”€ base.html # Base template â”‚ â”œâ”€â”€ dashboard.html # Main dashboard â”‚ â””â”€â”€ ... â”œâ”€â”€ static/ # Static files â”‚ â”œâ”€â”€ css/ # Stylesheets â”‚ â”œâ”€â”€ js/ # JavaScript â”‚ â””â”€â”€ images/ # Images â””â”€â”€ utils/ # Utility modules â”œâ”€â”€ helpers.py # Helper functions â””â”€â”€ ...

Common Modifications

1. Changing Platform Name/Branding

File: `models/settings.py`

```
# Change app name
class AppSettings(db.Model):
    app_name = db.Column(db.String(100), default="Your Platform Name")
```

File: `templates/base.html`

```
<!-- Update title and branding -->
<title>{% block title %}Your Platform Name{% endblock %}</title>
```

File: `static/manifest.json`

```
{
  "name": "Your Platform Name",
  "short_name": "YPN"
}
```

2. Customizing Mining Rates

File: `models/user.py`

```
def calculate_mining_reward(self):
    base_reward = 10.0  # Change base mining reward
    level_multiplier = 1.0 + (self.level * 0.1)  # Adjust level bonus
    return base_reward * level_multiplier
```

3. Adding New Task Platforms

File: `models/task.py`

```
TASK_PLATFORMS = [
    ('twitter', 'Twitter/X'),
    ('telegram', 'Telegram'),
    ('youtube', 'YouTube'),
    ('your_platform', 'Your Platform'),  # Add new platform
]
```

4. Modifying Referral Rates

File: `models/referral.py`

```
def calculate_commission(self, amount):
    commission_rate = 0.15  # 15% commission (adjust as needed)
    return amount * commission_rate
```

5. Customizing Email Templates

File: `utils/smtp_client.py`

```
def send_welcome_email(user_email, username):
    subject = "Welcome to Your Platform!"
    body = f"""
    <h1>Welcome {username}!</h1>
    <p>Thank you for joining our platform...</p>
    """
    return send_email(user_email, subject, body)
```

CSS Customization

Color Scheme

File: `static/css/style.css`

```
:root {
    --primary-color: #your-color;
    --secondary-color: #your-secondary;
    --accent-color: #your-accent;
    --text-color: #your-text;
    --background-color: #your-background;
}
```

Typography

```
/* Custom fonts */
@import url('https://fonts.googleapis.com/css2?family=YourFont:wght@400;600;700&display=swap');

body {
    font-family: 'YourFont', sans-serif;
}
```

JavaScript Modifications

Mining Animation

File: `static/js/mining.js`

```
class MiningSystem {
    constructor() {
        this.animationDuration = 1000;  // Adjust animation speed
        this.clickCooldown = 1000;      // Adjust click cooldown
    }

    // Customize mining animation
    animateMining() {
        // Your custom animation code
    }
}
```

Custom Features

```
// Add new feature to app.js
class CustomFeature {
    constructor() {
        this.initCustomFeature();
    }

    initCustomFeature() {
        // Your custom feature implementation
    }
}

// Initialize on page load
document.addEventListener('DOMContentLoaded', () => {
    new CustomFeature();
});
```

Security Configuration

Essential Security Settings

1. Session Security

```
# app.py - Update session configuration
app.config.update(
    SESSION_COOKIE_SECURE=True,      # HTTPS only
    SESSION_COOKIE_HTTPONLY=True,    # No JavaScript access
    SESSION_COOKIE_SAMESITE='Lax',   # CSRF protection
    PERMANENT_SESSION_LIFETIME=timedelta(hours=24)  # Session timeout
)
```

2. CSRF Protection

```
# Already implemented via Flask-WTF
from flask_wtf.csrf import CSRFProtect

csrf = CSRFProtect()
csrf.init_app(app)
```

3. Password Security

```
# models/user.py - Password requirements
def set_password(self, password):
    # Add password strength validation
    if len(password) < 8:
        raise ValueError("Password must be at least 8 characters")
    if not re.search(r"[A-Z]", password):
        raise ValueError("Password must contain uppercase letter")
    if not re.search(r"[a-z]", password):
        raise ValueError("Password must contain lowercase letter")
    if not re.search(r"\d", password):
        raise ValueError("Password must contain number")

    self.password_hash = generate_password_hash(password)
```

4. File Upload Security

```
# utils/helpers.py
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif', 'webp'}
MAX_FILE_SIZE = 16 * 1024 * 1024  # 16MB

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def secure_upload(file):
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        # Additional security checks
        return filename
    return None
```

5. Rate Limiting

```
# Add to requirements.txt: Flask-Limiter
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address

limiter = Limiter(
    app,
    key_func=get_remote_address,
    default_limits=["1000 per hour"]
)

# Apply to sensitive endpoints
@app.route('/login', methods=['POST'])
@limiter.limit("5 per minute")
def login():
    # Login logic
```

Firewall Configuration

UFW (Ubuntu Firewall)

```
# Enable firewall
sudo ufw enable

# Allow SSH (if using)
sudo ufw allow ssh
```

```
# Allow HTTP and HTTPS
sudo ufw allow 80
sudo ufw allow 443

# Allow specific application port
sudo ufw allow 5000

# Check status
sudo ufw status
```

Fail2Ban (Intrusion Prevention)

```
# Install Fail2Ban
sudo apt install fail2ban

# Configure for Flask app
sudo nano /etc/fail2ban/jail.local
```

```
[flask-app]
enabled = true
port = 5000
filter = flask-app
logpath = /var/log/rosecoin/app.log
maxretry = 5
bantime = 3600
```

Regular Security Tasks

1. Security Updates

```
# Update system packages
sudo apt update && sudo apt upgrade -y

# Update Python packages
pip list --outdated
pip install --upgrade package_name
```

2. Log Monitoring

```
# Monitor application logs
tail -f logs/app.log

# Monitor system logs
sudo tail -f /var/log/auth.log
sudo tail -f /var/log/nginx/error.log
```

3. Database Security

```
# PostgreSQL security
sudo nano /etc/postgresql/12/main/postgresql.conf
# ssl = on
# log_statement = 'all'

# Backup encryption
pg_dump rosecoin | gpg --symmetric --cipher-algo AES256 > backup.sql.gpg
```

Email System Setup

SMTP Configuration

Gmail SMTP Setup

1. **Enable 2-Factor Authentication on your Gmail account**
2. **Generate App Password:**
   - **Go to Google Account settings**
   - **Security → 2-Step Verification → App passwords**
   - **Generate password for “Mail”**
3. **Configure Environment Variables:**

```
SMTP_SERVER=smtp.gmail.com
SMTP_PORT=587
SMTP_USERNAME=your-email@gmail.com
SMTP_PASSWORD=your-app-password
SMTP_USE_TLS=true
```

Other Email Providers

Outlook/Hotmail:

```
SMTP_SERVER=smtp-mail.outlook.com
SMTP_PORT=587
SMTP_USE_TLS=true
```

Yahoo:

```
SMTP_SERVER=smtp.mail.yahoo.com
SMTP_PORT=587
SMTP_USE_TLS=true
```

Custom SMTP:

```
SMTP_SERVER=mail.yourdomain.com
SMTP_PORT=587  # or 465 for SSL
SMTP_USE_TLS=true  # or false for SSL
```

Email Templates

Customizing Email Templates

File: `utils/smtp_client.py`

```
def get_email_template(template_name, **kwargs):
    templates = {
        'welcome': {
            'subject': 'Welcome to {app_name}!',
            'body': '''
            <html>
            <body style="font-family: Arial, sans-serif;">
                <h2>Welcome to {app_name}, {username}!</h2>
                <p>Thank you for joining our cryptocurrency mining platform.</p>
                <p>Your account has been created successfully. You can now:</p>
                <ul>
                    <li>Start mining coins</li>
                    <li>Complete social media tasks</li>
                    <li>Refer friends and earn commissions</li>
                    <li>Participate in airdrops</li>
                </ul>
                <p><a href="{login_url}" style="background: #007bff; color: white; padding: 10px 20px; text-decoration: none; border-radius: 5px;">Login Now</a></p>
                <p>Best regards,<br>The {app_name} Team</p>
            </body>
            </html>
            '''
        },
        'withdrawal_approved': {
            'subject': 'Withdrawal Approved - {amount} DSC',
            'body': '''
            <html>
            <body style="font-family: Arial, sans-serif;">
                <h2>Withdrawal Approved!</h2>
                <p>Dear {username},</p>
                <p>Your withdrawal request has been approved:</p>
                <ul>
                    <li>Amount: {amount} DSC</li>
                    <li>Method: {method}</li>
                    <li>Processing Date: {date}</li>
                </ul>
                <p>Your payment will be processed within 24-48 hours.</p>
                <p>Best regards,<br>The {app_name} Team</p>
            </body>
            </html>
            '''
        }
    }

    template = templates.get(template_name, {})
    return {
        'subject': template.get('subject', '').format(**kwargs),
        'body': template.get('body', '').format(**kwargs)
    }
```

Email Testing

Test Email Configuration

```
# test_email.py
from utils.smtp_client import send_email, test_smtp_connection

def test_email_system():
    # Test SMTP connection
    if test_smtp_connection():
        print("âœ… SMTP connection successful")

        # Send test email
        result = send_email(
            to_email="test@example.com",
            subject="Test Email from DSC Platform",
            body="<h1>Test Email</h1><p>If you receive this, email system is working!</p>"
        )

        if result:
            print("âœ… Test email sent successfully")
        else:
            print("âŒ Failed to send test email")
    else:
        print("âŒ SMTP connection failed")

if __name__ == "__main__":
    test_email_system()
```

Email Delivery Best Practices

1. SPF Record

Add to your DNS:

TXT record: v=spf1 include:_spf.google.com ~all

2. DKIM Setup

Configure DKIM signing for your domain

3. DMARC Policy

TXT record: v=DMARC1; p=quarantine; rua=mailto:dmarc@yourdomain.com

---

Payment Integration

Supported Payment Methods

The platform supports multiple withdrawal methods:

1. ## PayPal
2. ## Bank Transfer
3. ## Cryptocurrency Wallets
4. ## Mobile Money
5. ## Gift Cards

PayPal Integration

Setup PayPal API

1. ## Create PayPal Developer Account
2. ## Get API Credentials
3. ## Configure Environment Variables:

```
PAYPAL_CLIENT_ID=your_client_id
PAYPAL_CLIENT_SECRET=your_client_secret
PAYPAL_MODE=sandbox  # or live for production
```

PayPal Implementation

```
# utils/paypal_client.py
import paypalrestsdk
import os

paypalrestsdk.configure({
    "mode": os.environ.get("PAYPAL_MODE", "sandbox"),
    "client_id": os.environ.get("PAYPAL_CLIENT_ID"),
    "client_secret": os.environ.get("PAYPAL_CLIENT_SECRET")
})

def process_paypal_payment(email, amount):
    payout = paypalrestsdk.Payout({
        "sender_batch_header": {
            "sender_batch_id": f"batch-{int(time.time())}",
            "email_subject": "Payment from rosecoin"
        },
        "items": [{
            "recipient_type": "EMAIL",
            "amount": {
                "value": str(amount),
                "currency": "USD"
            },
            "receiver": email,
            "note": "Payment from DSC Platform"
        }]
    })

    if payout.create():
        return True, payout.batch_header.payout_batch_id
    else:
        return False, payout.error
```

Cryptocurrency Integration

Bitcoin/Ethereum Payments

```
# utils/crypto_payments.py
import requests

def send_bitcoin_payment(address, amount):
    # Integration with cryptocurrency exchange API
    # Example: Coinbase Commerce, BitPay, etc.
    pass

def validate_crypto_address(address, currency):
    # Validate cryptocurrency address format
    if currency.lower() == 'bitcoin':
```

```
        return validate_bitcoin_address(address)
    elif currency.lower() == 'ethereum':
        return validate_ethereum_address(address)
    return False
```

Manual Payment Processing

For manual payment methods (bank transfer, mobile money):

```
# core/admin.py
@admin_bp.route('/withdrawals/process/<int:withdrawal_id>')
@login_required
@admin_required
def process_withdrawal(withdrawal_id):
    withdrawal = Withdrawal.query.get_or_404(withdrawal_id)

    # Manual processing workflow
    if request.method == 'POST':
        action = request.form.get('action')

        if action == 'approve':
            withdrawal.status = 'approved'
            withdrawal.processed_at = datetime.utcnow()

            # Send notification email
            send_withdrawal_approved_email(withdrawal.user.email, withdrawal)

        elif action == 'reject':
            withdrawal.status = 'rejected'
            withdrawal.user.balance += withdrawal.amount  # Refund balance

        db.session.commit()
        flash(f'Withdrawal {action}d successfully!', 'success')

    return redirect(url_for('admin.withdrawals'))
```

Customization Guide

Theme Customization

Color Scheme

File: `static/css/style.css`

```
/* Primary Color Scheme */
:root {
    /* Main Colors */
    --primary-color: #667eea;      /* Main brand color */
    --secondary-color: #764ba2;    /* Secondary brand color */
    --accent-color: #f093fb;       /* Accent highlights */

    /* Neutral Colors */
    --dark-color: #2c3e50;         /* Dark text/backgrounds */
    --light-color: #ecf0f1;        /* Light backgrounds */
    --white-color: #ffffff;        /* Pure white */

    /* Status Colors */
    --success-color: #28a745;      /* Success messages */
    --warning-color: #ffc107;      /* Warning messages */
    --danger-color: #dc3545;       /* Error messages */
    --info-color: #17a2b8;         /* Info messages */

    /* Gradients */
    --gradient-primary: linear-gradient(135deg, var(--primary-color) 0%, var(--secondary-color) 100%);
    --gradient-accent: linear-gradient(135deg, var(--accent-color) 0%, var(--primary-color) 100%);
}

/* Dark Mode Support */
@media (prefers-color-scheme: dark) {
    :root {
        --dark-color: #ecf0f1;
        --light-color: #2c3e50;
        --white-color: #34495e;
    }
}
```

Component Styling

```
/* Custom Button Styles */
.btn-custom {
    background: var(--gradient-primary);
    border: none;
    color: white;
    padding: 12px 24px;
    border-radius: 25px;
    font-weight: 600;
    transition: all 0.3s ease;
}

.btn-custom:hover {
    transform: translateY(-2px);
    box-shadow: 0 10px 20px rgba(102, 126, 234, 0.3);
}

/* Custom Card Styles */
.card-custom {
    background: rgba(255, 255, 255, 0.1);
    backdrop-filter: blur(10px);
    border: 1px solid rgba(255, 255, 255, 0.2);
    border-radius: 15px;
```

```
}
```

Logo and Branding

Adding Custom Logo

1. ## Add logo files to `static/images/`:

   - ### `logo.png` - Main logo
   - ### `logo-small.png` - Small version
   - ### `favicon.ico` - Favicon

2. ## Update templates:

File: `templates/base.html`

```
<!-- Update header logo -->
<img src="{{ url_for('static', filename='images/logo.png') }}"
     alt="rosecoin" class="logo">

<!-- Update favicon -->
<link rel="icon" type="image/x-icon" href="{{ url_for('static', filename='images/favicon.ico') }}">
```

Custom Fonts

```
/* Import custom fonts */
@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600;700&display=swap');

body {
    font-family: 'Poppins', sans-serif;
}

h1, h2, h3, h4, h5, h6 {
    font-family: 'Poppins', sans-serif;
    font-weight: 600;
}
```

Layout Customization

Custom Dashboard Layout

File: `templates/dashboard.html`

```
<!-- Custom dashboard sections -->
<div class="dashboard-grid">
    <div class="stats-section">
        <div class="stat-card">
            <div class="stat-icon">
                <i class="fas fa-coins"></i>
            </div>
            <div class="stat-content">
                <h3>{{ current_user.balance }}</h3>
                <p>Total Balance</p>
            </div>
        </div>
        <!-- More stat cards -->
    </div>

    <div class="activity-section">
        <h4>Recent Activity</h4>
        <!-- Activity feed -->
    </div>

    <div class="tasks-section">
        <h4>Available Tasks</h4>
        <!-- Task list -->
    </div>
</div>
```

Responsive Design

```
/* Mobile-first responsive design */
.dashboard-grid {
    display: grid;
    grid-template-columns: 1fr;
    gap: 20px;
}

/* Tablet and up */
@media (min-width: 768px) {
    .dashboard-grid {
        grid-template-columns: 2fr 1fr;
    }
}

/* Desktop and up */
@media (min-width: 1024px) {
    .dashboard-grid {
        grid-template-columns: 2fr 1fr 1fr;
    }
}
```

Feature Customization

Adding New Task Types

File: `models/task.py`

```
# Add new task types
TASK_TYPES = [
    ('follow', 'Follow Account'),
    ('like', 'Like Post'),
    ('share', 'Share Content'),
    ('comment', 'Comment on Post'),
    ('subscribe', 'Subscribe to Channel'),
    ('watch', 'Watch Video'),
    ('custom_task', 'Custom Task'),  # New task type
]

# Add custom validation for new task type
def validate_custom_task(self, proof_text, proof_image):
    # Custom validation logic
    if self.task_type == 'custom_task':
        # Implement custom validation
        return True
    return False
```

Custom Mining Mechanics

File: `static/js/mining.js`

```
class CustomMiningSystem extends MiningSystem {
    constructor() {
        super();
        this.bonusMultiplier = 1.0;
        this.streakBonus = 0;
    }

    calculateMiningReward(baseReward) {
        // Custom reward calculation
        let reward = baseReward * this.bonusMultiplier;

        // Streak bonus
        if (this.streakBonus > 0) {
            reward *= (1 + this.streakBonus * 0.1);
        }

        // Time-based bonus
        const hour = new Date().getHours();
        if (hour >= 18 && hour <= 22) {  // Evening bonus
            reward *= 1.2;
        }

        return Math.round(reward);
    }

    addCustomAnimation() {
        // Custom mining animation
        const miningButton = document.getElementById('mining-button');

        // Add particle effect
        this.createParticles(miningButton);

        // Add screen shake
        document.body.classList.add('mining-shake');
        setTimeout(() => {
            document.body.classList.remove('mining-shake');
        }, 500);
    }
}
```

Admin Panel Configuration

Admin Account Setup

Default Admin Account

- **Email: admin@rosecoin.com**
- **Password: admin123**
- **Access: Full administrative privileges**

Creating Additional Admins

```
# Create admin user programmatically
from models.user import User
from werkzeug.security import generate_password_hash

def create_admin_user(username, email, password):
    admin_user = User(
        username=username,
        email=email,
        password_hash=generate_password_hash(password),
        is_admin=True,
        email_verified=True,
        balance=0,
        xp=0,
        level=1
    )
```

```
    db.session.add(admin_user)
    db.session.commit()
    print(f"Admin user {username} created successfully!")
```

Admin Panel Features

Dashboard Overview

- **User Statistics: Total users, active users, new registrations**
- **Financial Overview: Total balance, pending withdrawals, revenue**
- **System Status: Server health, database status, background tasks**
- **Recent Activity: Latest user actions, system events**

User Management

- **User List: View all users with filtering and search**
- **User Details: Complete user profiles with activity history**
- **User Actions:**
  - **Edit user information**
  - **Adjust balances and XP**
  - **Ban/unban users**
  - **Delete accounts (with data preservation options)**

Task Management

- **Create Tasks: Add new social media tasks**
- **Task Categories: Organize tasks by platform and type**
- **Task Review: Approve/reject task completions**
- **Bulk Operations: Process multiple tasks simultaneously**

Withdrawal Management

- **Pending Withdrawals: Review and process withdrawal requests**
- **Payment Methods: Configure supported payment options**
- **Bulk Processing: Process multiple withdrawals**
- **Payment History: Track all processed payments**

System Settings

- **Platform Configuration: App name, descriptions, maintenance mode**
- **Mining Settings: Rates, bonuses, daily limits**
- **Referral Settings: Commission rates, maximum levels**
- **Email Settings: SMTP configuration, template management**

Admin Security

Role-Based Access

```
# models/admin.py
class AdminRole(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), unique=True, nullable=False)
    permissions = db.Column(db.JSON, default=list)

class AdminUser(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    role_id = db.Column(db.Integer, db.ForeignKey('admin_role.id'), nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

# Permission decorator
def require_permission(permission):
    def decorator(f):
```

```
        @wraps(f)
        def decorated_function(*args, **kwargs):
            if not current_user.has_permission(permission):
                abort(403)
            return f(*args, **kwargs)
        return decorated_function
    return decorator

# Usage
@admin_bp.route('/users/delete/<int:user_id>')
@login_required
@admin_required
@require_permission('delete_users')
def delete_user(user_id):
    # Delete user logic
    pass
```

Admin Activity Logging

```
# models/admin_log.py
class AdminLog(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    admin_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    action = db.Column(db.String(100), nullable=False)
    target_type = db.Column(db.String(50))  # user, task, withdrawal, etc.
    target_id = db.Column(db.Integer)
    details = db.Column(db.JSON)
    ip_address = db.Column(db.String(45))
    user_agent = db.Column(db.Text)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

# Log admin actions
def log_admin_action(action, target_type=None, target_id=None, details=None):
    log_entry = AdminLog(
        admin_id=current_user.id,
        action=action,
        target_type=target_type,
        target_id=target_id,
        details=details,
        ip_address=request.remote_addr,
        user_agent=request.user_agent.string
    )
    db.session.add(log_entry)
    db.session.commit()
```

Admin Panel Customization

Custom Admin Theme

File: `static/css/admin.css`

```
/* Admin panel custom styling */
.admin-panel {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    min-height: 100vh;
}

.admin-sidebar {
    background: rgba(255, 255, 255, 0.1);
    backdrop-filter: blur(10px);
    border-right: 1px solid rgba(255, 255, 255, 0.2);
}

.admin-card {
    background: rgba(255, 255, 255, 0.95);
    border-radius: 15px;
    box-shadow: 0 10px 30px rgba(0, 0, 0, 0.1);
    border: none;
}

.admin-stat-card {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
    border-radius: 15px;
    padding: 20px;
    margin-bottom: 20px;
}
```

Dashboard Widgets

```
# core/admin.py - Custom dashboard widgets
def get_dashboard_stats():
    stats = {
        'total_users': User.query.count(),
        'active_users': User.query.filter_by(is_active=True).count(),
        'total_balance': db.session.query(func.sum(User.balance)).scalar() or 0,
        'pending_withdrawals': Withdrawal.query.filter_by(status='pending').count(),
        'completed_tasks_today': TaskCompletion.query.filter(
            TaskCompletion.submitted_at >= datetime.utcnow().date()
        ).count(),
        'revenue_today': calculate_daily_revenue(),
    }
    return stats

def calculate_daily_revenue():
    # Calculate revenue from promotions, fees, etc.
    today = datetime.utcnow().date()
    # Implementation based on your revenue model
    return 0.0
```

Troubleshooting

Common Issues and Solutions

1. Database Connection Issues

Issue: `sqlalchemy.exc.OperationalError: (sqlite3.OperationalError) database is locked`

Solution:

```
# Check for processes using the database
sudo lsof /path/to/rosecoin.db

# Kill any hanging processes
sudo kill -9 <process_id>

# Restart the application
python main.py
```

Prevention:

```
# Add connection pooling
app.config["SQLALCHEMY_ENGINE_OPTIONS"] = {
    "pool_recycle": 300,
    "pool_pre_ping": True,
    "pool_size": 10,
    "max_overflow": 20
}
```

2. Email Sending Failures

Issue: SMTP authentication errors

Diagnosis:

```
# Test email configuration
from utils.smtp_client import test_smtp_connection

if test_smtp_connection():
    print("âœ… SMTP connection successful")
else:
    print("â SMTP connection failed - check credentials")
```

Solutions: - Verify SMTP credentials - Enable “Less secure app access” for Gmail (or use App Passwords) - Check firewall settings for SMTP ports - Verify TLS/SSL settings

3. File Upload Issues

Issue: File uploads failing or corrupting

Check permissions:

```
# Check upload directory permissions
ls -la uploads/
chmod 755 uploads/
chown www-data:www-data uploads/
```

Check file size limits:

```
# app.py
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024  # 16MB
```

4. Performance Issues

Issue: Slow page loading, high server load

Database optimization:

```
# Add database indexes
class User(db.Model):
    email = db.Column(db.String(120), unique=True, nullable=False, index=True)
    username = db.Column(db.String(80), unique=True, nullable=False, index=True)
    created_at = db.Column(db.DateTime, default=datetime.utcnow, index=True)
```

Query optimization:

```
# Use eager loading for relationships
users = User.query.options(joinedload(User.mining_sessions)).all()

# Paginate large result sets
users = User.query.paginate(page=1, per_page=50)
```

Static file caching:

 # Nginx configuration location /static/ { alias /var/www/rosecoin/static/; expires 1y; add_header Cache-Control "public, immutable"; }

5. Mining System Issues

Issue: Mining button not responding

Check JavaScript errors:

```
// Enable debugging in mining.js
class MiningSystem {
    constructor() {
        this.debug = true;  // Enable debug mode
        this.initMining();
    }

    mine() {
        if (this.debug) console.log('Mining attempt started');
        // Mining logic
    }
}
```

Check network requests:

```
# Add logging to mining endpoint
@mining_bp.route('/mine', methods=['POST'])
@login_required
def mine():
    app.logger.info(f'Mining request from user {current_user.id}')
    # Mining logic
```

Error Log Analysis

Log Configuration

```
# app.py - Enhanced logging
import logging
from logging.handlers import RotatingFileHandler

if not app.debug:
    # File logging
    file_handler = RotatingFileHandler('logs/rosecoin.log',
                                       maxBytes=10240000, backupCount=10)
    file_handler.setFormatter(logging.Formatter(
        '%(asctime)s %(levelname)s: %(message)s [in %(pathname)s:%(lineno)d]'
    ))
    file_handler.setLevel(logging.INFO)
    app.logger.addHandler(file_handler)

    app.logger.setLevel(logging.INFO)
    app.logger.info('rosecoin startup')
```

Common Error Patterns

Database Errors:

`ERROR: (sqlite3.IntegrityError) UNIQUE constraint failed: user.email`

- **Solution: Check for duplicate email registrations**

Template Errors:

`jinja2.exceptions.TemplateNotFound: template.html`

- **Solution: Verify template file exists and path is correct**

Import Errors:

`ImportError: No module named 'package_name'`

- **Solution: Install missing dependencies with `pip install package_name`**

Performance Monitoring

System Resource Monitoring

```
# Monitor CPU and memory usage
htop

# Monitor disk usage
df -h

# Monitor database performance
sudo tail -f /var/log/postgresql/postgresql-12-main.log
```

Application Performance

```
# Add performance monitoring
import time
from flask import g

@app.before_request
def before_request():
    g.start_time = time.time()

@app.after_request
def after_request(response):
    if hasattr(g, 'start_time'):
        total_time = time.time() - g.start_time
        if total_time > 1.0:  # Log slow requests
```

```
            app.logger.warning(f'Slow request: {request.endpoint} took {total_time:.2f}s')
    return response
```

Backup and Recovery

Automated Backup System

```
# utils/backup.py
import os
import json
import shutil
from datetime import datetime

def create_backup():
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    backup_dir = f'backups/backup_{timestamp}'
    os.makedirs(backup_dir, exist_ok=True)

    # Backup database
    if app.config['SQLALCHEMY_DATABASE_URI'].startswith('sqlite'):
        shutil.copy2('instance/rosecoin.db',
                     f'{backup_dir}/database.db')

    # Backup uploads
    shutil.copytree('uploads', f'{backup_dir}/uploads')

    # Backup configuration
    backup_config = {
        'timestamp': timestamp,
        'app_version': '1.0',
        'database_type': 'sqlite',
        'user_count': User.query.count(),
    }

    with open(f'{backup_dir}/backup_info.json', 'w') as f:
        json.dump(backup_config, f, indent=2)

    return backup_dir

# Schedule backup (add to cron)
# 0 2 * * * cd /var/www/rosecoin && python -c "from utils.backup import create_backup; create_backup()"
```

Maintenance & Updates

Regular Maintenance Tasks

Daily Tasks

- **Monitor application logs**
- **Check system resources (CPU, memory, disk)**
- **Review user activity and detect anomalies**
- **Process pending withdrawals**
- **Backup database**

Weekly Tasks

- **Update system packages**
- **Review security logs**
- **Clean up temporary files**
- **Analyze performance metrics**
- **Update documentation**

Monthly Tasks

- **Security audit**
- **Full system backup**
- **Performance optimization review**
- **User feedback analysis**
- **Feature planning**

Update Procedures

Application Updates

1. ## Backup Current System

```
# Create backup before updates
python -c "from utils.backup import create_backup; create_backup()"
```

2. ## Update Dependencies

```
# Update Python packages
pip list --outdated
pip install --upgrade package_name

# Update requirements.txt
pip freeze > requirements.txt
```

3. ## Database Migrations

```
# Create migration script
from app import app, db
from flask_migrate import Migrate, upgrade

migrate = Migrate(app, db)

# Run migrations
upgrade()
```

4. ## Test Updates

```
# Run test suite (if available)
python -m pytest tests/

# Manual testing checklist:
# - User registration/login
# - Mining functionality
# - Task completion
# - Admin panel access
# - Email notifications
```

Security Updates

System Security Updates:

```
# Ubuntu/Debian
sudo apt update && sudo apt upgrade -y

# CentOS/RHEL
sudo yum update -y
```

Python Security Updates:

```
# Check for security vulnerabilities
pip install safety
safety check

# Update vulnerable packages
pip install --upgrade vulnerable-package
```

SSL Certificate Renewal:

```
# Let's Encrypt renewal
sudo certbot renew

# Check certificate expiry
openssl x509 -in /path/to/certificate.crt -text -noout | grep "Not After"
```

Version Control

Git Workflow

```
# Initialize repository
git init
git add .
git commit -m "Initial commit"

# Create development branch
git checkout -b development

# Make changes and commit
git add modified_files
git commit -m "Feature: Add new functionality"

# Merge to main branch
git checkout main
git merge development

# Tag releases
git tag -a v1.0.1 -m "Version 1.0.1 release"
git push origin v1.0.1
```

Configuration Management

```
# config.py - Environment-based configuration
import os

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'dev-secret-key'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or 'sqlite:///rosecoin.db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

```
class DevelopmentConfig(Config):
    DEBUG = True
    TESTING = False

class ProductionConfig(Config):
    DEBUG = False
    TESTING = False

class TestingConfig(Config):
    DEBUG = True
    TESTING = True
    SQLALCHEMY_DATABASE_URI = 'sqlite:///test.db'

config = {
    'development': DevelopmentConfig,
    'production': ProductionConfig,
    'testing': TestingConfig,
    'default': DevelopmentConfig
}
```

Monitoring and Alerts

Health Check Endpoints

```
# app.py - Health check endpoints
@app.route('/health')
def health_check():
    health_status = {
        'status': 'healthy',
        'timestamp': datetime.utcnow().isoformat(),
        'version': '1.0',
        'checks': {
            'database': check_database_health(),
            'email': check_email_health(),
            'storage': check_storage_health()
        }
    }

    # Determine overall health
    if all(health_status['checks'].values()):
        return jsonify(health_status), 200
    else:
        health_status['status'] = 'unhealthy'
        return jsonify(health_status), 503

def check_database_health():
    try:
        db.session.execute('SELECT 1')
        return True
    except Exception:
        return False

def check_email_health():
    try:
        from utils.smtp_client import test_smtp_connection
        return test_smtp_connection()
    except Exception:
        return False

def check_storage_health():
    import shutil
    try:
        disk_usage = shutil.disk_usage('.')
        free_space_gb = disk_usage.free / (1024**3)
        return free_space_gb > 1.0  # At least 1GB free
    except Exception:
        return False
```

Alerting System

```
# utils/alerts.py
import smtplib
from email.mime.text import MIMEText

def send_alert(subject, message, severity='info'):
    """Send alert email to administrators"""
    admin_emails = ['admin@yourdomain.com']

    msg = MIMEText(f"""
    Alert: {subject}
    Severity: {severity}
    Time: {datetime.utcnow()}

    Details:
    {message}

    Server: {os.uname().nodename}
    Application: rosecoin
    """)

    msg['Subject'] = f'[DSC Alert] {subject}'
    msg['From'] = 'alerts@yourdomain.com'
    msg['To'] = ', '.join(admin_emails)

    try:
        smtp = smtplib.SMTP('localhost')
        smtp.send_message(msg)
        smtp.quit()
        return True
    except Exception as e:
        app.logger.error(f'Failed to send alert: {e}')
        return False

# Usage in application
```

```
def monitor_system():
    # Check critical metrics
    if not check_database_health():
        send_alert('Database Connection Failed', 'Cannot connect to database', 'critical')

    # Check disk space
    free_space = shutil.disk_usage('.').free / (1024**3)
    if free_space < 1.0:
        send_alert('Low Disk Space', f'Only {free_space:.2f}GB remaining', 'warning')
```

API Documentation

Authentication API

Login Endpoint

POST /auth/login Content-Type: application/json { "email": "user@example.com", "password": "password123" }

Response:

```
{
    "success": true,
    "user": {
        "id": 1,
        "username": "testuser",
        "email": "user@example.com",
        "balance": 1500.50,
        "level": 5,
        "xp": 2500
    },
    "session_token": "abc123..."
}
```

Registration Endpoint

POST /auth/register Content-Type: application/json { "username": "newuser", "email": "newuser@example.com", "password": "password123", "referral_code": "ABC123" // optional }

Mining API

Mine Coins

POST /mining/mine Authorization: Bearer <session_token> Content-Type: application/json { "mining_power": 1.0 }

Response:

```
{
    "success": true,
    "coins_earned": 25.5,
    "new_balance": 1526.0,
    "xp_gained": 10,
    "new_xp": 2510,
    "level_up": false,
    "cooldown": 60
}
```

Get Mining Stats

GET /mining/stats Authorization: Bearer <session_token>

Response:

```
{
    "total_mined": 5000.0,
    "mining_sessions": 250,
    "last_mining": "2025-07-19T10:30:00Z",
    "next_available": "2025-07-19T10:31:00Z",
    "daily_limit": 1000.0,
    "daily_mined": 150.0
}
```

Task API

Get Available Tasks

GET /tasks/available Authorization: Bearer <session_token>

Response:

```
{
    "tasks": [
        {
            "id": 1,
            "title": "Follow us on Twitter",
            "description": "Follow our official Twitter account",
            "platform": "twitter",
            "reward": 50.0,
            "task_type": "follow",
            "external_url": "https://twitter.com/ourhandle",
```

```
            "requirements": "Must follow the account"
        }
    ],
    "total": 10,
    "page": 1,
    "per_page": 20
}
```

Submit Task Completion

POST /tasks/complete/<task_id> Authorization: Bearer <session_token> Content-Type: multipart/form-data { "proof_text": "Completed the task as requested", "proof_image": <file_upload> }

User API

Get User Profile

GET /api/user/profile Authorization: Bearer <session_token>

Response:

```
{
    "id": 1,
    "username": "testuser",
    "email": "user@example.com",
    "balance": 1500.50,
    "xp": 2500,
    "level": 5,
    "mining_power": 1.2,
    "referral_code": "ABC123",
    "created_at": "2025-01-01T00:00:00Z",
    "last_login": "2025-07-19T10:00:00Z"
}
```

Update Profile

PUT /api/user/profile Authorization: Bearer <session_token> Content-Type: application/json { "username": "newusername", "wallet_address": "0x123..." }

Admin API

Get System Stats

GET /api/admin/stats Authorization: Bearer <admin_token>

Response:

```
{
    "users": {
        "total": 1000,
        "active": 850,
        "new_today": 25
    },
    "financial": {
        "total_balance": 50000.0,
        "pending_withdrawals": 5000.0,
        "revenue_today": 100.0
    },
    "tasks": {
        "pending_review": 15,
        "completed_today": 150
    }
}
```

Webhook API

Withdrawal Status Update

POST /webhooks/withdrawal-status Content-Type: application/json Authorization: Bearer <webhook_secret> { "withdrawal_id": 123, "status": "completed", "transaction_id": "tx123...", "processed_at": "2025-07-19T10:30:00Z" }

Rate Limiting

All API endpoints are rate-limited: - General endpoints: 1000 requests per hour per IP - Mining endpoint: 1 request per minute per user - Authentication endpoints: 5 requests per minute per IP - Admin endpoints: 100 requests per hour per admin

Error Responses

```
{
    "success": false,
    "error": {
        "code": "INSUFFICIENT_BALANCE",
        "message": "Not enough balance for this operation",
        "details": {
            "required": 100.0,
            "available": 50.0
        }
    }
```

```
}
```

Common Error Codes: - `AUTHENTICATION_FAILED` - Invalid credentials - `AUTHORIZATION_REQUIRED` - Missing or invalid token - `RATE_LIMIT_EXCEEDED` - Too many requests - `VALIDATION_ERROR` - Invalid input data - `RESOURCE_NOT_FOUND` - Requested resource doesn't exist - `INSUFFICIENT_BALANCE` - Not enough funds - `MINING_COOLDOWN` - Mining too frequently - `TASK_ALREADY_COMPLETED` - Task already submitted

---

Support & Contact

Getting Help

Documentation Resources

- **Installation Guide: This document**
- **API Documentation: Section 16 of this guide**
- **Video Tutorials: Available on our support portal**
- **FAQ: Common questions and solutions**

Support Channels

Email Support: - General Support: support@rosecoin.com - Technical Issues: tech@rosecoin.com - Business Inquiries: business@rosecoin.com

Response Times: - Critical Issues: Within 2 hours - General Support: Within 24 hours - Feature Requests: Within 48 hours

Community Support

Discord Community: - Join our Discord server for real-time help - Connect with other platform operators - Get unofficial community support

GitHub Repository: - Report bugs and issues - Request new features - Contribute to the project - Access source code updates

Professional Services

Custom Development

We offer custom development services for: - Custom feature development - Third-party integrations - UI/UX customization - Performance optimization - Security audits

Managed Hosting

- **Fully managed hosting solutions**
- **99.9% uptime guarantee**
- **Automatic backups and updates**
- **24/7 monitoring and support**
- **Scalability management**

Consulting Services

- **Platform strategy consultation**
- **Marketing and growth advice**
- **Technical architecture review**
- **Security assessment**
- **Compliance guidance**

Author Information

Olasunkanmi Julius Agbajelola - Role: Lead Developer & Creator - Email:

ceo@digitalskeleton.com.ng -

License Information

This software is licensed under a commercial license.

License Terms: - âœ… Commercial use permitted - âœ… Modification allowed - âœ… Distribution allowed (with restrictions) - âœ… Private use permitted - âŒ Sublicensing not permitted - âŒ Warranty not provided

Usage Rights: - Single domain license (upgrade available) - Unlimited users per domain - Free updates for 12 months - Extended support options available

Changelog

Version 1.0.0 (Current)

- **Initial release**
- **Complete mining platform**
- **Social media task system**
- **Referral program**
- **Admin panel**
- **PWA support**
- **Multi-language ready**

Planned Updates

Version 1.1.0 (Q3 2025): - AI-powered task verification - Mobile app (iOS/Android) - Advanced analytics dashboard - Multi-currency support - Enhanced security features

Version 1.2.0 (Q4 2025): - Blockchain integration - NFT rewards system - Advanced gamification - Social trading features - Enterprise features

Acknowledgments

Special Thanks: - Flask community for the excellent framework - Bootstrap team for the UI framework - All beta testers and early adopters - Open source contributors

Third-Party Libraries: - Flask and extensions (MIT License) - Bootstrap (MIT License) - Font Awesome (Font Awesome License) - jQuery (MIT License)

Document Version: 1.0
Last Updated: July 19, 2025
Document Status: Final

This documentation is comprehensive and covers all aspects of installing, configuring, and maintaining the rosecoin platform. For the most up-to-date information and additional resources, please visit our support portal or contact our team directly.