Production Deployment Guide

This document outlines the steps to deploy the `platform_core` backend application to a production server with Thawani payment gateway integration.

Prerequisites

Before deploying, ensure the following are installed on your production server:

- Python 3.x
- pip (Python package installer)
- Git
- Nginx (for reverse proxy)
- A database system (e.g., SQLite, PostgreSQL, MySQL) compatible with SQLAlchemy.

Deployment Steps

1. Clone the Repository

First, clone your project repository to your production server:

```
 git clone https://github.com/alphazee09/platform_core.git
cd platform_core
```

2. Set up Environment Variables

Create a `.env` file in the root directory of your project and populate it with your production environment variables. Replace the placeholder values with your actual secrets and configurations:

```
SESSION_SECRET=your_production_session_secret_key_here_make_it_very_long_and_ranc
DATABASE_URL=sqlite:///platform.db  # Or your production database URL (e.g., 
postgresql://user:password@host:port/database_name)
STRIPE_PUBLISHABLE_KEY=pk_live_your_stripe_publishable_key
STRIPE_SECRET_KEY=sk_live_your_stripe_secret_key
THAWANI_API_KEY=your_thawani_production_api_key
THAWANI_MERCHANT_CODE=your_thawani_merchant_code
THAWANI_SUCCESS_URL=https://yourdomain.com/payment/success
THAWANI_CANCEL_URL=https://yourdomain.com/payment/cancel
THAWANI_WEBHOOK_URL=https://yourdomain.com/payment/webhook
```

Important Notes: - For production, use Thawani production API keys instead of test keys - Update the URLs to match your actual domain - Keep the `.env` file secure and never commit it to version control

3. Install Dependencies

Install the required Python packages using pip:

```
pip install -r requirements.txt
```

4. Database Migrations

Apply database migrations to set up your database schema. Ensure your `FLASK_APP` environment variable is set to `manage.py`.

```
export FLASK_APP=manage.py
flask db upgrade
```

5. Seed Database (Optional)

If you need to populate your database with initial data (e.g., admin users), run the seeding script:

```
python seed.py
```

6. Configure Gunicorn

The project includes a `gunicorn.conf.py` configuration file. You can start the application using:

```
gunicorn -c gunicorn.conf.py main:app
```

Or manually with custom settings:

```
gunicorn -w 4 -b 0.0.0.0:5000 main:app
```

7. Configure Nginx (Recommended)

Copy the provided Nginx configuration:

```
sudo cp nginx.conf /etc/nginx/sites-available/platform_core
sudo ln -s /etc/nginx/sites-available/platform_core /etc/nginx/sites-enabled/
```

Edit the configuration file to match your domain:

```
sudo nano /etc/nginx/sites-available/platform_core
```

Update the `server_name` directive with your actual domain name.

Test and restart Nginx:

```
sudo nginx -t
sudo systemctl restart nginx
```

8. Set up Systemd Service

Copy the provided systemd service file:

```
sudo cp platform_core.service /etc/systemd/system/
```

Edit the service file if needed:

```
sudo nano /etc/systemd/system/platform_core.service
```

Enable and start the service:

```
sudo systemctl daemon-reload
sudo systemctl enable platform_core
sudo systemctl start platform_core
```

Check the service status:

```
sudo systemctl status platform_core
```

9. Thawani Payment Gateway Configuration

The application is pre-configured to work with Thawani payment gateway. Key features include:

- **Test Environment**: Uses `https://uatcheckout.thawani.om/api/v1` for testing
- **Production Environment**: Update the API base URL in `payment.py` to production endpoint
- **Test Cards Available**:
- `4242 4242 4242 4242` - Always Accept (OTP: 1234)
- `4000 0000 0000 0002` - Always Reject (OTP: 1234)
- `4456 5300 0000 1096` - 3D Secure Accept (OTP: 1234)
- `4456 5300 0000 1104` - 3D Secure Reject (OTP: 1234)

Payment Flow: 1. User visits `/payment/checkout?amount=X&description=Y` 2. User fills in email and phone number 3. System creates Thawani checkout session 4. User is redirected to Thawani payment page 5. After payment, user is redirected to success/cancel page 6. Webhook updates payment status in database

10. SSL/TLS Configuration (Recommended)

For production, set up SSL certificates using Let's Encrypt:

```
 sudo apt install certbot python3-certbot-nginx
sudo certbot --nginx -d yourdomain.com -d www.yourdomain.com
```

11. Firewall Configuration

Configure UFW firewall:

```
sudo ufw allow 22/tcp
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable
```

12. Monitoring and Logs

View application logs:

```
sudo journalctl -u platform_core -f
```

View Nginx logs:

```
sudo tail -f /var/log/nginx/access.log
sudo tail -f /var/log/nginx/error.log
```

Security Considerations

- **Environment Variables:** Never hardcode sensitive information. Use environment variables or a dedicated secrets management solution.
- **Database Security:** Use strong passwords and consider encrypting sensitive data.
- **API Keys:** Keep Thawani API keys secure and rotate them regularly.
- **HTTPS:** Always use HTTPS in production for secure data transmission.
- **Firewall:** Configure firewall to allow only necessary ports.
- **Updates:** Keep the system and dependencies updated regularly.
- **Backup:** Implement regular database backups.

Troubleshooting

Common Issues:

1. **Port Already in Use:** `bash sudo lsof -i :5000 sudo kill -9 <PID>`

2. **Permission Denied:** `bash sudo chown -R ubuntu:ubuntu /home/ubuntu/platform_core chmod +x /home/ubuntu/platform_core/main.py`
3. **Database Connection Issues:**
4. Check DATABASE_URL in .env file
5. Ensure database server is running
6. Verify database permissions
7. **Thawani API Issues:**
8. Verify API keys are correct
9. Check if using correct environment (test vs production)
10. Ensure webhook URL is accessible from Thawani servers

Testing the Deployment:

1. **Health Check:** `bash curl -I http://yourdomain.com`
2. **Payment Test:**
3. Visit `/payment/checkout?amount=1.00&description=Test`
4. Use test card numbers provided
5. Verify success/cancel redirects work
6. **Database Test:** `bash python -c "from app import app, db; from models import User; print(User.query.count())"`

This guide provides a comprehensive overview for production deployment. Specific configurations may vary based on your server environment and requirements.