# Panduan Developer Hurtrock Music Store

## Daftar Isi

* [Arsitektur Sistem](#arsitektur-sistem)
* [Tech Stack](#tech-stack)
* [Setup Development Environment](#setup-development-environment)
* [Struktur Project](#struktur-project)
* [Database Schema](#database-schema)
* [API Documentation](#api-documentation)
* [Frontend Architecture](#frontend-architecture)
* [Backend Architecture](#backend-architecture)
* [Authentication & Authorization](#authentication--authorization)
* [Payment Integration](#payment-integration)
* [Real-time Chat System](#real-time-chat-system)
* [File Upload & Management](#file-upload--management)
* [Testing](#testing)
* [Deployment](#deployment)
* [Performance Optimization](#performance-optimization)
* [Security Guidelines](#security-guidelines)
* [Troubleshooting](#troubleshooting)

## Arsitektur Sistem

### System Overview

Hurtrock Music Store menggunakan arsitektur **MVC (Model-View-Controller)** dengan **Flask** sebagai framework utama. Aplikasi dirancang dengan pendekatan **monolithic** yang modular dan dapat di-scale horizontal.

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ Frontend │ │ Backend │ │ Database │  
│ (Templates) │◄──►│ (Flask App) │◄──►│ (PostgreSQL) │  
│ Bootstrap 5 │ │ SQLAlchemy │ │ Relational │  
│ JavaScript │ │ Socket.IO │ │ ACID │  
└─────────────────┘ └─────────────────┘ └─────────────────┘  
 │ │ │  
 ▼ ▼ ▼  
┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ CDN/Static │ │ Payment │ │ File Storage │  
│ Images │ │ Gateway │ │ Local/Cloud │  
│ CSS/JS │ │ Stripe/MT │ │ Images │  
└─────────────────┘ └─────────────────┘ └─────────────────┘

### Component Architecture

* **Presentation Layer**: Jinja2 Templates + Bootstrap 5 + Vanilla JS
* **Business Logic Layer**: Flask Routes + Service Classes
* **Data Access Layer**: SQLAlchemy ORM + PostgreSQL
* **Integration Layer**: Payment Gateways + Email Services
* **Real-time Layer**: Socket.IO untuk chat dan notifications

## Tech Stack

### Backend

* **Python 3.11+**: Programming language utama
* **Flask 3.1.2**: Web framework
* **SQLAlchemy 2.0+**: ORM dan database abstraction
* **PostgreSQL**: Primary database
* **Flask-Login**: User session management
* **Flask-WTF**: Form handling dan CSRF protection
* **Flask-Migrate**: Database migration
* **Flask-SocketIO**: Real-time communication

### Frontend

* **Jinja2**: Template engine
* **Bootstrap 5.3**: CSS framework
* **Vanilla JavaScript**: Client-side logic
* **Font Awesome**: Icon library
* **Google Fonts**: Typography

### Payment & Integration

* **Stripe**: International payment gateway
* **Midtrans**: Indonesian payment gateway
* **Socket.IO**: Real-time messaging

### Development Tools

* **Git**: Version control
* **pip/uv**: Package management
* **Python-dotenv**: Environment variables
* **Werkzeug**: WSGI utilities

## Setup Development Environment

### Prerequisites

# Python 3.11+  
python --version  
  
# PostgreSQL  
psql --version  
  
# Git  
git --version

### Installation Steps

1. **Clone Repository**:

git clone [repository-url]  
cd hurtrock-music-store

1. **Auto Installation (Recommended)**:

# Universal installation  
chmod +x install.sh  
./install.sh  
  
# Atau untuk Linux server  
chmod +x install-linux.sh  
./install-linux.sh

1. **Manual Installation**:

# Install Dependencies  
pip install -r requirements.txt  
  
# Environment Variables  
cp .env.example .env  
# Edit .env dengan konfigurasi yang sesuai  
  
# Database Setup  
python migrate\_db.py  
  
# Load sample data  
python sample\_data.py

1. **Run Application**:

**Using Unified Server (Recommended)**:

python server.py

**Using Start Script**:

./start\_server.sh

**Development Mode (Flask only)**:

python main.py

### Unified Server Architecture

Server.py menjalankan Flask dan Django secara bersamaan:

* **Flask Main App**: Port 5000
* **Django Chat Service**: Port 8000
* **Automatic Setup**: Database migration dan service coordination
* **Package Ready**: Siap untuk PyInstaller packaging

### Environment Variables

# .env file  
SESSION\_SECRET=your-secret-key-here  
DATABASE\_URL=postgresql://user:password@localhost/hurtrock\_music\_store  
STRIPE\_SECRET\_KEY=sk\_test\_your\_stripe\_key  
STRIPE\_PUBLISHABLE\_KEY=pk\_test\_your\_stripe\_key  
MIDTRANS\_SERVER\_KEY=your\_midtrans\_server\_key  
MIDTRANS\_CLIENT\_KEY=your\_midtrans\_client\_key  
FLASK\_ENV=development  
FLASK\_DEBUG=1

## Struktur Project

hurtrock-music-store/  
├── main.py # Entry point aplikasi  
├── database.py # Database configuration  
├── models.py # SQLAlchemy models  
├── requirements.txt # Python dependencies  
├── pyproject.toml # Project metadata  
├── sample\_data.py # Sample data generator  
├── migrate\_db.py # Database migration script  
├── migrations/ # Database migrations  
│ ├── alembic.ini  
│ ├── env.py  
│ └── versions/  
├── static/ # Static assets  
│ ├── css/  
│ │ └── style.css  
│ ├── js/  
│ │ └── script.js  
│ └── images/  
├── templates/ # Jinja2 templates  
│ ├── base.html # Base template  
│ ├── index.html # Homepage  
│ ├── admin/ # Admin templates  
│ │ ├── base.html  
│ │ ├── dashboard.html  
│ │ ├── products.html  
│ │ ├── chat\_detail.html  
│ │ └── ...  
│ └── ...  
├── dokumentasi/ # Documentation  
│ ├── USER\_GUIDE.md  
│ ├── ADMIN\_GUIDE.md  
│ ├── DEVELOPER\_GUIDE.md  
│ └── MAINTAINER\_GUIDE.md  
└── README.md # Project overview

## Database Schema

### Entity Relationship Diagram (ERD)

erDiagram  
 User ||--o{ Order : creates  
 User ||--o{ CartItem : has  
 User ||--o{ ChatRoom : initiates  
   
 Order ||--o{ OrderItem : contains  
 Product ||--o{ OrderItem : ordered\_in  
 Product ||--o{ CartItem : added\_to  
 Product }o--|| Category : belongs\_to  
 Product }o--o| Supplier : supplied\_by  
   
 ChatRoom ||--o{ ChatMessage : contains  
 ChatMessage }o--o| Product : tags  
   
 Order }o--o| ShippingService : ships\_via  
   
 PaymentConfiguration ||--o{ Order : processes

### Core Models

# User Model  
class User(UserMixin, db.Model):  
 id = db.Column(Integer, primary\_key=True)  
 email = db.Column(String(120), unique=True, nullable=False)  
 password\_hash = db.Column(String(255), nullable=False)  
 name = db.Column(String(100), nullable=False)  
 phone = db.Column(String(20))  
 address = db.Column(Text)  
 role = db.Column(String(20), default='buyer') # admin, staff, buyer  
 active = db.Column(Boolean, default=True)  
 created\_at = db.Column(DateTime, default=datetime.utcnow)  
  
# Product Model  
class Product(db.Model):  
 id = db.Column(Integer, primary\_key=True)  
 name = db.Column(String(200), nullable=False)  
 description = db.Column(Text)  
 price = db.Column(Numeric(10, 2), nullable=False)  
 stock\_quantity = db.Column(Integer, default=0)  
 image\_url = db.Column(String(255))  
 brand = db.Column(String(100))  
 model = db.Column(String(100))  
 category\_id = db.Column(Integer, ForeignKey('categories.id'))  
 supplier\_id = db.Column(Integer, ForeignKey('suppliers.id'))  
 weight = db.Column(Numeric(8, 2), default=0) # dalam gram  
 length = db.Column(Numeric(8, 2), default=0) # dalam cm  
 width = db.Column(Numeric(8, 2), default=0)  
 height = db.Column(Numeric(8, 2), default=0)  
 is\_active = db.Column(Boolean, default=True)  
 is\_featured = db.Column(Boolean, default=False)  
 created\_at = db.Column(DateTime, default=datetime.utcnow)  
  
# Order Model  
class Order(db.Model):  
 id = db.Column(Integer, primary\_key=True)  
 user\_id = db.Column(Integer, ForeignKey('users.id'))  
 total\_amount = db.Column(Numeric(10, 2), nullable=False)  
 status = db.Column(String(50), default='pending')  
 shipping\_service\_id = db.Column(Integer, ForeignKey('shipping\_services.id'))  
 shipping\_cost = db.Column(Numeric(10, 2), default=0)  
 shipping\_address = db.Column(Text)  
 tracking\_number = db.Column(String(100))  
 created\_at = db.Column(DateTime, default=datetime.utcnow)

### Migration Commands

# Inisialisasi migration (hanya sekali)  
flask db init  
  
# Buat migration baru setelah perubahan model  
flask db migrate -m "Description of changes"  
  
# Apply migration ke database  
flask db upgrade  
  
# Rollback migration  
flask db downgrade  
  
# Lihat history migration  
flask db history

## API Documentation

### Authentication Required Endpoints

Semua endpoint yang memerlukan authentication menggunakan @login\_required decorator.

### Chat API

**GET /api/chat/messages**

* **Description**: Mengambil semua pesan chat user
* **Authentication**: Required
* **Response**:

{  
 "success": true,  
 "messages": [  
 {  
 "id": 1,  
 "message": "Hello",  
 "sender\_type": "user",  
 "timestamp": "14:30",  
 "product\_id": null  
 }  
 ]  
}

**POST /api/chat/send**

* **Description**: Mengirim pesan chat
* **Authentication**: Required
* **Body**:

{  
 "message": "Product inquiry",  
 "product\_id": 123  
}

* **Response**:

{  
 "success": true,  
 "message\_id": 456  
}

**POST /api/chat/clear**

* **Description**: Menghapus semua pesan chat user
* **Authentication**: Required
* **Response**:

{  
 "success": true  
}

### Search API

**GET /search**

* **Description**: Mencari produk
* **Parameters**:
  + q: Query string (min 2 characters)
* **Response**:

[  
 {  
 "id": 1,  
 "name": "Yamaha F310",  
 "price": "1500000",  
 "image\_url": "/static/images/guitar.jpg",  
 "brand": "Yamaha",  
 "description": "Gitar akustik untuk pemula..."  
 }  
]

### Admin API

**POST /admin/send\_reply**

* **Description**: Admin mengirim balasan chat
* **Authentication**: Required (Staff/Admin)
* **Body**:

room\_id=123  
message=Terima kasih pertanyaannya  
product\_id=456  
csrf\_token=xxx

**GET /api/chat/pending\_count**

* **Description**: Jumlah chat pending untuk admin
* **Authentication**: Required (Staff/Admin)
* **Response**:

{  
 "count": 5  
}

## Frontend Architecture

### Template Structure

**Base Template** (templates/base.html):

* Header dengan navigation
* Search functionality
* Footer
* JavaScript dan CSS includes
* Flash message handling

**Page Templates**:

* Extend dari base template
* Block untuk content spesifik
* Page-specific JavaScript dan CSS

### JavaScript Architecture

**Script Organization**:

// Global variables dan konfigurasi  
const CONFIG = {  
 SEARCH\_DELAY: 300,  
 CHAT\_POLL\_INTERVAL: 5000  
};  
  
// Search functionality  
function initializeSearch() {  
 // Search implementation  
}  
  
// Chat functionality  
function initializeChat() {  
 // Chat implementation  
}  
  
// Product functionality  
function initializeProducts() {  
 // Product implementation  
}  
  
// Document ready initialization  
document.addEventListener('DOMContentLoaded', function() {  
 initializeSearch();  
 initializeChat();  
 initializeProducts();  
});

### Responsive Design

**Breakpoints**:

/\* Mobile First Approach \*/  
/\* Base styles untuk mobile \*/  
  
@media (min-width: 576px) {  
 /\* Small tablets \*/  
}  
  
@media (min-width: 768px) {  
 /\* Tablets \*/  
}  
  
@media (min-width: 992px) {  
 /\* Desktop \*/  
}  
  
@media (min-width: 1200px) {  
 /\* Large desktop \*/  
}

### CSS Architecture

**Style Organization**:

/\* 1. CSS Variables \*/  
:root {  
 --orange-primary: #ff7b00;  
 --black-primary: #1a1a1a;  
 --transition-smooth: all 0.3s ease;  
}  
  
/\* 2. Base Styles \*/  
body { ... }  
  
/\* 3. Layout Components \*/  
.navbar { ... }  
.footer { ... }  
  
/\* 4. UI Components \*/  
.btn-orange { ... }  
.card-product { ... }  
  
/\* 5. Page Specific \*/  
.admin-sidebar { ... }  
.chat-widget { ... }  
  
/\* 6. Utilities \*/  
.text-orange { ... }  
.bg-glass { ... }

## Backend Architecture

### Application Factory Pattern

# main.py  
def create\_app():  
 app = Flask(\_\_name\_\_)  
   
 # Configuration  
 app.config.from\_object(Config)  
   
 # Initialize extensions  
 db.init\_app(app)  
 migrate.init\_app(app, db)  
 login\_manager.init\_app(app)  
 csrf.init\_app(app)  
 socketio.init\_app(app)  
   
 # Register blueprints  
 from routes import main\_bp, admin\_bp, api\_bp  
 app.register\_blueprint(main\_bp)  
 app.register\_blueprint(admin\_bp, url\_prefix='/admin')  
 app.register\_blueprint(api\_bp, url\_prefix='/api')  
   
 return app

### Route Organization

**Main Routes** (customer-facing):

@app.route('/')  
def index():  
 # Homepage dengan featured products  
   
@app.route('/products')  
def products():  
 # Product catalog dengan filtering  
   
@app.route('/product/<int:product\_id>')  
def product\_detail(product\_id):  
 # Detail produk

**Admin Routes** (admin panel):

@app.route('/admin/dashboard')  
@login\_required  
@admin\_required  
def admin\_dashboard():  
 # Admin dashboard  
   
@app.route('/admin/products')  
@login\_required  
@staff\_required  
def admin\_products():  
 # Product management

**API Routes** (AJAX endpoints):

@app.route('/api/chat/send', methods=['POST'])  
@login\_required  
@csrf.exempt  
def api\_chat\_send():  
 # Chat API endpoint

### Service Layer Pattern

# services/product\_service.py  
class ProductService:  
 @staticmethod  
 def get\_featured\_products(limit=8):  
 return Product.query.filter\_by(  
 is\_active=True,   
 is\_featured=True  
 ).limit(limit).all()  
   
 @staticmethod  
 def search\_products(query, category\_id=None):  
 # Product search logic  
   
 @staticmethod  
 def update\_stock(product\_id, quantity\_change):  
 # Stock management logic

### Decorators

**Custom Decorators**:

def admin\_required(f):  
 @wraps(f)  
 def decorated\_function(\*args, \*\*kwargs):  
 if not current\_user.is\_authenticated or not current\_user.is\_admin:  
 flash('Access denied. Admin required.', 'error')  
 return redirect(url\_for('index'))  
 return f(\*args, \*\*kwargs)  
 return decorated\_function  
  
def staff\_required(f):  
 @wraps(f)  
 def decorated\_function(\*args, \*\*kwargs):  
 if not current\_user.is\_authenticated or (not current\_user.is\_admin and not current\_user.is\_staff):  
 flash('Access denied. Staff or admin required.', 'error')  
 return redirect(url\_for('index'))  
 return f(\*args, \*\*kwargs)  
 return decorated\_function

## Authentication & Authorization

### User Roles

1. **Admin**: Full access ke semua fitur
2. **Staff**: Access ke product management dan customer service
3. **Buyer**: Regular customer access

### Session Management

# Flask-Login configuration  
login\_manager = LoginManager()  
login\_manager.login\_view = 'login'  
login\_manager.login\_message = 'Silakan login untuk mengakses halaman ini.'  
  
@login\_manager.user\_loader  
def load\_user(user\_id):  
 return db.session.get(User, int(user\_id))

### Password Security

from werkzeug.security import generate\_password\_hash, check\_password\_hash  
  
# Hash password saat register  
hashed\_password = generate\_password\_hash(password)  
  
# Verify password saat login  
if user and check\_password\_hash(user.password\_hash, password):  
 login\_user(user)

### CSRF Protection

# Inisialisasi CSRF  
csrf = CSRFProtect(app)  
  
# Exempt API endpoints  
@app.route('/api/chat/send', methods=['POST'])  
@csrf.exempt  
def api\_chat\_send():  
 # API logic

## Payment Integration

### Stripe Integration

**Configuration**:

stripe.api\_key = payment\_config.stripe\_secret\_key  
  
# Create checkout session  
checkout\_session = stripe.checkout.Session.create(  
 line\_items=line\_items,  
 mode='payment',  
 success\_url=f'https://{domain}/payment-success',  
 cancel\_url=f'https://{domain}/cart',  
 customer\_email=current\_user.email,  
)

### Midtrans Integration

**Configuration**:

snap = midtransclient.Snap(  
 is\_production=not payment\_config.is\_sandbox,  
 server\_key=payment\_config.midtrans\_server\_key,  
 client\_key=payment\_config.midtrans\_client\_key  
)  
  
# Create transaction  
transaction = snap.create\_transaction(param)

### Payment Flow

1. **User** selects products dan checkout
2. **System** calculates total dengan shipping
3. **Payment Gateway** processes payment
4. **Webhook** updates order status
5. **Email** notification dikirim ke user
6. **Admin** dapat process order untuk shipping

## Real-time Chat System

### Socket.IO Implementation

**Server-side**:

from flask\_socketio import SocketIO, emit, join\_room, leave\_room  
  
socketio = SocketIO(app, cors\_allowed\_origins="\*")  
  
@socketio.on('join')  
def on\_join(data):  
 if current\_user.is\_authenticated:  
 if current\_user.is\_admin or current\_user.is\_staff:  
 join\_room('admin\_room')  
 else:  
 join\_room(f'user\_{current\_user.id}')  
  
@socketio.on('disconnect')  
def on\_disconnect():  
 # Handle disconnect

**Client-side**:

const socket = io();  
  
socket.emit('join', {});  
  
socket.on('new\_message', function(data) {  
 displayMessage(data.message, data.sender\_type);  
});

### Chat Features

1. **Real-time messaging** antara user dan admin
2. **Product tagging** dalam chat
3. **Unread message badges**
4. **Message history** tersimpan di database
5. **Admin notifications** untuk chat baru

## File Upload & Management

### Image Upload

def allowed\_file(filename):  
 return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS  
  
def compress\_image(image\_path, max\_size\_mb=1):  
 img = Image.open(image\_path)  
   
 if img.mode in ('RGBA', 'LA', 'P'):  
 img = img.convert('RGB')  
   
 quality = 85  
 while True:  
 img.save(image\_path, 'JPEG', quality=quality, optimize=True)  
 file\_size\_mb = os.path.getsize(image\_path) / (1024 \* 1024)  
   
 if file\_size\_mb <= max\_size\_mb or quality <= 20:  
 break  
 quality -= 10  
   
 return image\_path

### File Organization

static/  
├── images/  
│ ├── products/ # Product images  
│ ├── categories/ # Category images  
│ ├── uploads/ # User uploads  
│ └── placeholders/ # Placeholder images  
├── css/  
│ └── style.css  
└── js/  
 └── script.js

## Testing

### Unit Testing

import unittest  
from main import create\_app  
from database import db  
from models import User, Product  
  
class ProductTestCase(unittest.TestCase):  
 def setUp(self):  
 self.app = create\_app()  
 self.app.config['TESTING'] = True  
 self.app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///:memory:'  
   
 with self.app.app\_context():  
 db.create\_all()  
   
 def tearDown(self):  
 with self.app.app\_context():  
 db.session.remove()  
 db.drop\_all()  
   
 def test\_product\_creation(self):  
 with self.app.app\_context():  
 product = Product(  
 name='Test Guitar',  
 price=1000000,  
 category\_id=1  
 )  
 db.session.add(product)  
 db.session.commit()  
   
 self.assertEqual(product.name, 'Test Guitar')  
 self.assertEqual(product.price, 1000000)

### Integration Testing

def test\_checkout\_flow(self):  
 with self.client as c:  
 # Login  
 c.post('/login', data={'email': 'test@test.com', 'password': 'password'})  
   
 # Add to cart  
 c.post('/add\_to\_cart/1', data={'quantity': 1})  
   
 # Checkout  
 response = c.post('/checkout', data={  
 'shipping\_service\_id': 1,  
 'payment\_config\_id': 1  
 })  
   
 self.assertEqual(response.status\_code, 302) # Redirect to payment

### Testing Commands

# Run all tests  
python -m pytest  
  
# Run specific test file  
python -m pytest tests/test\_products.py  
  
# Run with coverage  
python -m pytest --cov=main  
  
# Run with verbose output  
python -m pytest -v

## Deployment

### Environment Configuration

**Production Settings**:

# Production environment  
FLASK\_ENV=production  
FLASK\_DEBUG=0  
DATABASE\_URL=postgresql://user:pass@host:5432/db  
SESSION\_SECRET=very-secure-secret-key  
STRIPE\_SECRET\_KEY=sk\_live\_...  
MIDTRANS\_SERVER\_KEY=production\_key

### Database Migration

# Production deployment  
flask db upgrade  
  
# Rollback jika ada masalah  
flask db downgrade

### Static File Handling

**Development**:

app.run(debug=True) # Flask serves static files

**Production**:

# Nginx configuration  
location /static {  
 alias /path/to/app/static;  
 expires 1y;  
 add\_header Cache-Control "public, immutable";  
}

### Process Management

**Systemd Service**:

[Unit]  
Description=Hurtrock Music Store  
After=network.target  
  
[Service]  
User=www-data  
WorkingDirectory=/var/www/hurtrock  
Environment=PATH=/var/www/hurtrock/venv/bin  
ExecStart=/var/www/hurtrock/venv/bin/gunicorn --workers 3 --bind unix:hurtrock.sock -m 007 main:app  
Restart=always  
  
[Install]  
WantedBy=multi-user.target

## Performance Optimization

### Database Optimization

**Query Optimization**:

# Eager loading untuk menghindari N+1 queries  
products = Product.query.options(  
 joinedload(Product.category),  
 joinedload(Product.supplier)  
).filter\_by(is\_active=True).all()  
  
# Index pada kolom yang sering di-query  
class Product(db.Model):  
 name = db.Column(String(200), nullable=False, index=True)  
 category\_id = db.Column(Integer, ForeignKey('categories.id'), index=True)

**Pagination**:

products = Product.query.paginate(  
 page=page,   
 per\_page=20,   
 error\_out=False  
)

### Caching Strategy

from flask\_caching import Cache  
  
cache = Cache(app)  
  
@cache.cached(timeout=300) # Cache 5 menit  
def get\_featured\_products():  
 return Product.query.filter\_by(is\_featured=True).all()

### Frontend Optimization

**Image Optimization**:

* Kompres gambar sebelum upload
* Lazy loading untuk gambar
* WebP format support
* Responsive images

**JavaScript Optimization**:

* Minify JS dan CSS
* Bundle dan compress assets
* Defer non-critical JavaScript
* Use CDN untuk libraries

## Security Guidelines

### Input Validation

from wtforms import ValidationError  
  
def validate\_email(form, field):  
 user = User.query.filter\_by(email=field.data).first()  
 if user:  
 raise ValidationError('Email already registered.')

### SQL Injection Prevention

# Menggunakan SQLAlchemy ORM (aman)  
users = User.query.filter\_by(email=email).all()  
  
# Jika harus raw SQL, gunakan parameterized queries  
result = db.session.execute(  
 text("SELECT \* FROM users WHERE email = :email"),   
 {"email": email}  
)

### XSS Prevention

<!-- Jinja2 auto-escape HTML -->  
<p>{{ user.name }}</p> <!-- Aman -->  
  
<!-- Jika perlu HTML, gunakan |safe dengan hati-hati -->  
<p>{{ user.bio|safe }}</p> <!-- Pastikan data sudah di-sanitize -->

### CSRF Protection

<!-- Include CSRF token dalam form -->  
<form method="POST">  
 <input type="hidden" name="csrf\_token" value="{{ csrf\_token() }}"/>  
 <!-- form fields -->  
</form>

### Secure Headers

@app.after\_request  
def set\_secure\_headers(response):  
 response.headers['X-Content-Type-Options'] = 'nosniff'  
 response.headers['X-Frame-Options'] = 'DENY'  
 response.headers['X-XSS-Protection'] = '1; mode=block'  
 return response

## Packaging untuk Distribusi

### PyInstaller Setup

Aplikasi telah didesain untuk packaging:

# Install PyInstaller  
pip install pyinstaller  
  
# Basic executable  
pyinstaller --onefile server.py  
  
# Dengan assets  
pyinstaller --onefile \  
 --add-data "templates:templates" \  
 --add-data "static:static" \  
 --add-data "chat\_service:chat\_service" \  
 server.py  
  
# GUI application (tanpa console)  
pyinstaller --onefile --windowed \  
 --add-data "templates:templates" \  
 --add-data "static:static" \  
 --add-data "chat\_service:chat\_service" \  
 server.py

### Distribusi

**Windows Executable**:

* Hasil: dist/server.exe
* Standalone, tidak perlu Python install
* Include semua dependencies

**Linux Package (.deb)**:

# Install fpm  
gem install fpm  
  
# Create package  
fpm -s dir -t deb \  
 -n hurtrock-music-store \  
 -v 1.0.0 \  
 --description "Hurtrock Music Store" \  
 dist/server=/usr/local/bin/

**macOS Application**:

# Create .app bundle  
pyinstaller --onefile --windowed \  
 --name "Hurtrock Music Store" \  
 server.py

### Docker Alternative

FROM python:3.11-slim  
WORKDIR /app  
COPY . .  
RUN pip install -r requirements.txt  
EXPOSE 5000 8000  
CMD ["python", "server.py"]

## Troubleshooting

### Common Issues

**Database Connection Error**:

# Check database status  
pg\_isready -h localhost -p 5432  
  
# Check connection string  
echo $DATABASE\_URL  
  
# Test connection  
psql $DATABASE\_URL -c "SELECT 1;"

**Migration Issues**:

# Reset migration (development only)  
rm -rf migrations/  
flask db init  
flask db migrate -m "Initial migration"  
flask db upgrade  
  
# Fix migration conflicts  
flask db merge heads

**Import Errors**:

# Check Python path  
import sys  
print(sys.path)  
  
# Check installed packages  
pip list | grep flask

**Socket.IO Connection Issues**:

// Debug Socket.IO connection  
socket.on('connect', function() {  
 console.log('Connected to server');  
});  
  
socket.on('disconnect', function() {  
 console.log('Disconnected from server');  
});  
  
socket.on('connect\_error', function(error) {  
 console.log('Connection error:', error);  
});

### Debugging Tools

**Flask Debug Mode**:

app.run(debug=True) # Development only

**Logging**:

import logging  
  
logging.basicConfig(level=logging.DEBUG)  
app.logger.debug('Debug message')  
app.logger.info('Info message')  
app.logger.error('Error message')

**Database Debugging**:

# SQLAlchemy query debugging  
app.config['SQLALCHEMY\_ECHO'] = True

### Performance Monitoring

**Query Profiling**:

from flask\_debugtoolbar import DebugToolbarExtension  
  
app.config['DEBUG\_TB\_ENABLED'] = True  
toolbar = DebugToolbarExtension(app)

**Application Monitoring**:

import time  
  
@app.before\_request  
def before\_request():  
 g.start\_time = time.time()  
  
@app.after\_request  
def after\_request(response):  
 duration = time.time() - g.start\_time  
 app.logger.info(f'Request took {duration:.2f}s')  
 return response

## Development Workflow

### Git Workflow

# Feature development  
git checkout -b feature/new-feature  
git add .  
git commit -m "Add new feature"  
git push origin feature/new-feature  
  
# Create pull request  
# Code review  
# Merge to main

### Code Style

**Python (PEP 8)**:

# Use 4 spaces for indentation  
# Max line length: 88 characters  
# Use snake\_case for variables dan functions  
# Use PascalCase untuk classes

**JavaScript**:

// Use 2 spaces for indentation  
// Use camelCase untuk variables dan functions  
// Use semicolons  
// Use const/let instead of var

### Documentation

* **Code Comments**: Explain complex logic
* **Docstrings**: Untuk functions dan classes
* **API Documentation**: Keep updated
* **README**: Setup instructions

*Panduan ini akan terus diperbarui seiring dengan pengembangan aplikasi. Untuk pertanyaan teknis lebih lanjut, silakan hubungi tim development.*