**Cursor Security Hardening Prompts for Mingus**

**🔍 PROMPT 1: Configuration Assessment & Security Audit**

**Context:** Assess current configuration setup and identify all hard-coded secrets before implementing security fixes.

**Cursor Prompt:**

Perform a comprehensive security audit of the Mingus application configuration system. I need to understand the current setup before implementing security hardening.

\*\*Tasks:\*\*

1. \*\*Configuration Analysis:\*\*

- Scan all files in config/ directory for hard-coded secrets

- Check for .env files vs config files usage

- Identify all sensitive data (API keys, passwords, tokens, secrets)

- List all configuration files and their purposes

2. \*\*Security Vulnerability Assessment:\*\*

- Find hard-coded values in: config/development.py, config/testing.py, config/stripe.py

- Search for patterns like: SECRET\_KEY, API\_KEY, PASSWORD, TOKEN, DATABASE\_URL

- Check for exposed credentials in any Python files

3. \*\*Environment Setup Detection:\*\*

- Determine if application uses python-dotenv or os.environ

- Check how Flask app loads configuration

- Identify current environment variable usage patterns

4. \*\*Create Assessment Report:\*\*

- List all found hard-coded secrets with file locations

- Recommend configuration strategy (config classes vs .env)

- Provide migration plan from current setup to secure setup

\*\*File Analysis Focus:\*\*

- config/development.py

- config/testing.py

- config/production.py (if exists)

- config/stripe.py

- app.py or \_\_init\_\_.py (Flask app initialization)

- Any docker-compose.yml or deployment files

\*\*Output Format:\*\*

Create a security\_audit\_report.md with:

- Current configuration method analysis

- All hard-coded secrets found (with file:line references)

- Recommended security implementation approach

- Step-by-step remediation plan

**🔒 PROMPT 2: Implement Secure Configuration Management**

**Context:** Based on the audit results, implement secure configuration management using the optimal method for the existing setup.

**Cursor Prompt:**

Implement secure configuration management for Mingus application. Based on the current Flask config class structure, create a secure configuration system that eliminates all hard-coded secrets.

\*\*Implementation Requirements:\*\*

1. \*\*Create Environment Template:\*\*

```bash

# Create env.template file with all required variables

# Include comments explaining each variable's purpose

1. **Update Configuration Classes:**
   * Modify config/development.py to use os.environ.get()
   * Update config/testing.py with environment variables
   * Secure config/production.py for production deployment
   * Fix config/stripe.py to use environment variables
   * **Environment Variable Implementation:**
   * import os
   * from dotenv import load\_dotenv
   * class Config:
   * SECRET\_KEY = os.environ.get('SECRET\_KEY') or 'dev-fallback-key'
   * SQLALCHEMY\_DATABASE\_URI = os.environ.get('DATABASE\_URL')
   * # Convert all hard-coded values to environment variables
2. **Security Enhancements:**
   * Add environment variable validation
   * Implement required vs optional variable checks
   * Add startup warnings for missing critical variables
   * Create secure random key generation for development
3. **Files to Update:**
   * config/development.py
   * config/testing.py
   * config/production.py
   * config/stripe.py
   * Create: env.template
   * Create: .env.example
   * Update: .gitignore (ensure .env\* files are ignored)
4. **Validation Function:**
5. def validate\_config():
6. required\_vars = ['SECRET\_KEY', 'DATABASE\_URL', 'SUPABASE\_KEY']
7. missing = [var for var in required\_vars if not os.environ.get(var)]
8. if missing:
9. raise ValueError(f"Missing required environment variables: {missing}")

**Security Requirements:**

* No hard-coded secrets in any file
* All sensitive data from environment variables
* Fallback values only for non-sensitive development settings
* Production configuration validates all required variables
* Clear error messages for missing critical variables

---

## 🔧 \*\*PROMPT 3: Fix Requirements.txt Merge Conflict\*\*

\*\*Context:\*\* Resolve the Git merge conflict in requirements.txt that's preventing deployment.

\*\*Cursor Prompt:\*\*

Fix the Git merge conflict in requirements.txt file for the Mingus application. Ensure all necessary dependencies are included and properly versioned.

**Tasks:**

1. **Analyze Current Conflict:**
   * Examine the merge conflict markers in requirements.txt
   * Identify conflicting dependency versions
   * Determine which dependencies are needed for current features
2. **Dependency Resolution:**
   * Keep all unique dependencies from both branches
   * Use latest compatible versions for conflicting packages
   * Ensure Flask ecosystem compatibility (Flask, SQLAlchemy, etc.)
   * Maintain Python 3.11+ compatibility
3. **Critical Dependencies Check:** Ensure these are included with proper versions:
4. Flask>=2.3.0
5. SQLAlchemy>=2.0.0
6. Flask-SQLAlchemy
7. psycopg2-binary
8. redis>=4.0.0
9. celery>=5.3.0
10. python-dotenv
11. stripe
12. twilio
13. resend
14. **Clean Up Process:**
    * Remove duplicate entries
    * Sort dependencies alphabetically
    * Add version pinning for stability
    * Remove any conflicted/broken packages
15. **Validation Steps:**
    * Test pip install -r requirements.txt in clean environment
    * Verify no dependency conflicts
    * Ensure all imports work in application
    * Create requirements-dev.txt if needed for development dependencies
16. **Git Resolution:**
17. # Remove merge conflict markers
18. # Test installation
19. # Commit resolved file

**Output:**

* Clean requirements.txt file
* Optional: requirements-dev.txt for development
* Installation validation report

---

## 🏥 \*\*PROMPT 4: Implement Health Check Endpoints\*\*

\*\*Context:\*\* Add comprehensive health check endpoints for production monitoring and deployment validation.

\*\*Cursor Prompt:\*\*

Implement comprehensive health check endpoints for the Mingus Flask application to enable proper production monitoring and deployment validation.

**Health Check Implementation:**

1. **Basic Health Endpoint:**
2. @app.route('/health')
3. def health\_check():
4. # Basic application health
5. return {'status': 'healthy', 'timestamp': datetime.utcnow().isoformat()}
6. **Detailed Health Endpoint:**
7. @app.route('/health/detailed')
8. def detailed\_health():
9. # Check database connectivity
10. # Check Redis connectivity
11. # Check external service status
12. # Return comprehensive status
13. **Database Health Check:**
    * Test SQLAlchemy connection
    * Verify database responsiveness
    * Check for any connection pool issues
14. **Redis Health Check:**
    * Test Redis connectivity
    * Verify cache operations
    * Check Redis memory usage
15. **External Services Health:**
    * Supabase API connectivity
    * Stripe API status
    * Email service (Resend) status
    * SMS service (Twilio) status
16. **File Structure:**
17. backend/
18. ├── routes/
19. │ └── health.py
20. ├── services/
21. │ └── health\_service.py
22. └── utils/
23. └── health\_checks.py
24. **Health Response Format:**
25. {
26. "status": "healthy|degraded|unhealthy",
27. "timestamp": "2025-08-03T12:00:00Z",
28. "version": "1.0.0",
29. "checks": {
30. "database": {"status": "healthy", "response\_time": 15},
31. "redis": {"status": "healthy", "response\_time": 5},
32. "external\_apis": {
33. "supabase": {"status": "healthy"},
34. "stripe": {"status": "healthy"}
35. }
36. }
37. }
38. **Monitoring Integration:**
    * Add Prometheus metrics for health checks
    * Include response time measurements
    * Add failure counters and alerting
39. **Security Considerations:**
    * No sensitive information in health responses
    * Rate limiting on health endpoints
    * Optional authentication for detailed checks

**Files to Create:**

* backend/routes/health.py
* backend/services/health\_service.py
* backend/utils/health\_checks.py
* tests/test\_health\_endpoints.py

---

## 🚀 \*\*PROMPT 5: Production Security Validation\*\*

\*\*Context:\*\* Final security validation and production readiness check after implementing all security fixes.

\*\*Cursor Prompt:\*\*

Perform final security validation and production readiness check for the Mingus application after implementing security hardening measures.

**Security Validation Tasks:**

1. **Configuration Security Audit:**
   * Verify no hard-coded secrets remain in any files
   * Confirm all sensitive data uses environment variables
   * Test configuration loading in all environments (dev, test, prod)
   * Validate environment variable requirements are documented
2. **Environment File Creation:**
   * Create .env.production template with all required variables
   * Generate secure random values for production secrets
   * Document all environment variables with descriptions
   * Create deployment-specific environment files
3. **Flask Security Headers:**
4. # Implement security headers if not already present
5. from flask\_talisman import Talisman
6. # Add HTTPS redirect, HSTS, CSP headers
7. # Configure secure session cookies
8. # Add XSS protection headers
9. **Production Security Checklist:**
   * [ ] No hard-coded secrets in codebase
   * [ ] All environment variables documented
   * [ ] Secure session configuration
   * [ ] Database connection security
   * [ ] API rate limiting enabled
   * [ ] HTTPS enforcement configured
   * [ ] Security headers implemented
   * [ ] Health endpoints secured
   * [ ] Logging configured (no sensitive data)
10. **Deployment Security:**
    * Update Docker configurations for production
    * Secure container environment variable handling
    * Validate production deployment process
    * Test backup and recovery procedures
11. **Security Testing:**
    * Test with missing environment variables
    * Verify error handling doesn't expose secrets
    * Validate all endpoints respond appropriately
    * Test health checks with various failure scenarios
12. **Documentation Updates:**
    * Update README with environment setup instructions
    * Document all required environment variables
    * Create production deployment guide
    * Add security best practices documentation

**Files to Update/Create:**

* .env.production (template)
* .env.example
* .gitignore (security additions)
* docs/SECURITY.md
* docs/DEPLOYMENT.md
* docker-compose.prod.yml (if applicable)

**Output:**

* Security validation report
* Production deployment checklist
* Environment variable documentation
* Security configuration guide

---

## 📋 \*\*EXECUTION ORDER\*\*

Run these prompts in sequence for systematic security hardening:

### \*\*Phase 1: Assessment\*\* ⏱️ 30 mins

1. \*\*PROMPT 1\*\* → Assess current configuration and identify issues

### \*\*Phase 2: Implementation\*\* ⏱️ 2-3 hours

2. \*\*PROMPT 2\*\* → Implement secure configuration management

3. \*\*PROMPT 3\*\* → Fix requirements.txt merge conflict

4. \*\*PROMPT 4\*\* → Add health check endpoints

### \*\*Phase 3: Validation\*\* ⏱️ 1 hour

5. \*\*PROMPT 5\*\* → Final security validation and production prep

---

## 🎯 \*\*Success Criteria\*\*

After completing all prompts, you should have:

✅ \*\*Zero hard-coded secrets\*\* in codebase

✅ \*\*Secure environment variable management\*\*

✅ \*\*Clean requirements.txt\*\* without conflicts

✅ \*\*Production-ready health endpoints\*\*

✅ \*\*Complete security validation\*\*

✅ \*\*Production deployment readiness\*\*

---

## ⚠️ \*\*Important Notes\*\*

- \*\*Run prompts sequentially\*\* - each builds on the previous

- \*\*Test after each prompt\*\* - validate changes work correctly

- \*\*Backup before starting\*\* - commit current state to git

- \*\*Review all changes\*\* - ensure no functionality is broken

- \*\*Update documentation\*\* - keep security docs current

---

## 🔒 \*\*Security Best Practices Applied\*\*

- \*\*Defense in Depth:\*\* Multiple security layers

- \*\*Least Privilege:\*\* Minimal required access

- \*\*Fail Secure:\*\* Safe defaults and error handling

- \*\*Security by Design:\*\* Built-in security from start

- \*\*Regular Auditing:\*\* Ongoing security validation

**REQUIRED BEFORE TESTING**

* [ ] **Remove hard-coded secrets** (CRITICAL)
* [ ] **Resolve requirements.txt conflict** (HIGH)
* [ ] **Implement health check endpoints** (MEDIUM)
* [ ] **Deploy to Digital Ocean** (HIGH)
* [ ] **Set up production database** (HIGH)
* [ ] **Configure environment variables** (HIGH)
* [ ] **Test core functionality** (MEDIUM)