**Cursor Prompts for MINGUS Security Hardening**

**🔒 PHASE 1: SECURITY HEADERS IMPLEMENTATION (Day 4)**

**Prompt 1: Implement Comprehensive Security Headers**

Create a comprehensive security headers system for the MINGUS Flask application that handles sensitive financial data for African American professionals.

Requirements:

- Create a security middleware file `security/headers.py` that implements all critical security headers

- Include these essential headers:

\* Content Security Policy (CSP) - strict policy for financial app

\* HTTP Strict Transport Security (HSTS) - force HTTPS

\* X-Content-Type-Options - prevent MIME sniffing

\* X-Frame-Options - prevent clickjacking

\* X-XSS-Protection - XSS filtering

\* Referrer-Policy - control referrer information

\* Permissions-Policy - control browser features

\* Expect-CT - Certificate Transparency

- Configure CSP for:

\* Self-hosted assets only

\* Stripe payment integration

\* Google Analytics/Microsoft Clarity

\* Block unsafe inline scripts

\* Restrict form submissions to same origin

- Create environment-specific configurations (development vs production)

- Include header validation and monitoring

- Add security header testing utilities

This is for a financial wellness app that processes payments, health data, and personal financial information, so security must be banking-grade.

**Prompt 2: Create HTTPS and SSL Security System**

Create a comprehensive HTTPS and SSL security system for MINGUS to ensure all data transmission is secure.

Requirements:

- Create `security/ssl\_config.py` for SSL/TLS configuration

- Implement automatic HTTPS redirect for all requests

- Configure secure session cookies with proper flags

- Set up HSTS preloading for major browsers

- Implement certificate pinning for production

- Add SSL/TLS version enforcement (TLS 1.2+)

- Configure secure cipher suites

- Add HTTPS health checks and monitoring

Key features:

- Force HTTPS in production environment

- Secure cookie configuration (HttpOnly, Secure, SameSite)

- SSL session management

- Certificate validation

- Mixed content prevention

- HTTPS-only forms and API endpoints

Include Digital Ocean deployment configuration for SSL termination and certificate management.

**Prompt 3: Implement Authentication Security**

Create a robust authentication security system for MINGUS that protects user accounts and sessions.

Requirements:

- Create `security/auth\_security.py` with enhanced authentication features

- Implement these security measures:

\* Password strength requirements (for financial app users)

\* Account lockout after failed attempts

\* Rate limiting on login endpoints

\* Session timeout and refresh

\* Secure password hashing (bcrypt with high cost)

\* Multi-factor authentication preparation

\* Suspicious activity detection

- Add these protections:

\* Brute force attack prevention

\* Session fixation protection

\* Concurrent session management

\* Password breach detection

\* Account compromise alerts

\* Secure password reset flow

- Include user activity logging for:

\* Login attempts (successful and failed)

\* Password changes

\* Profile modifications

\* Financial data access

\* Subscription changes

This is critical for protecting users' financial and personal data.

**🔒 PHASE 2: INPUT VALIDATION ENHANCEMENT (Day 4-5)**

**Prompt 4: Create Comprehensive Input Validation System**

Create a comprehensive input validation system for MINGUS that protects against injection attacks and data corruption.

Requirements:

- Create `security/validation.py` with validation classes for all data types

- Implement validation for these MINGUS-specific data categories:

\* Financial data (income, expenses, amounts) - prevent manipulation

\* Health metrics (stress levels, activity minutes) - ensure valid ranges

\* Personal information (names, addresses, phone) - sanitize and validate

\* Employment data (job titles, industries) - prevent XSS in career features

\* Subscription data (billing info, plans) - critical for revenue protection

- Include these validation types:

\* Input sanitization (remove/escape dangerous characters)

\* Data type validation (ensure correct types)

\* Range validation (financial amounts, health scores)

\* Length validation (prevent buffer overflow)

\* Format validation (email, phone, ZIP codes)

\* Regex patterns for specific fields

\* SQL injection prevention

\* XSS prevention

\* File upload validation (for resume/document uploads)

- Create validation decorators for Flask routes

- Include detailed error messages for users

- Add validation logging for security monitoring

This must protect against all common injection attacks while maintaining good user experience.

**Prompt 5: Implement API Security and Rate Limiting**

Create a comprehensive API security system for MINGUS that protects all endpoints from abuse and attacks.

Requirements:

- Create `security/api\_security.py` with complete API protection

- Implement rate limiting for different endpoint types:

\* Authentication endpoints: 5 attempts per minute

\* Financial data endpoints: 100 requests per hour per user

\* Health checkin endpoints: 10 submissions per day

\* Income comparison: 3 requests per hour (lead magnet protection)

\* PDF generation: 5 requests per hour

\* General API: 1000 requests per hour per user

- Add these API security features:

\* Request signature validation

\* API key management (for premium features)

\* Request size limiting

\* Response data filtering

\* CORS configuration for web app

\* API versioning security

\* Endpoint monitoring and alerting

- Implement these protections:

\* SQL injection prevention in API parameters

\* NoSQL injection prevention for JSON inputs

\* Command injection prevention

\* Path traversal prevention

\* Request smuggling prevention

- Include security middleware for all routes

- Add detailed API access logging

- Create API abuse detection and blocking

This protects the revenue-generating features and user data from API-based attacks.

**Prompt 6: Create Data Encryption and Protection System**

Create a comprehensive data encryption and protection system for MINGUS financial data.

Requirements:

- Create `security/encryption.py` for data protection

- Implement field-level encryption for sensitive data:

\* Financial account information

\* Social Security numbers (if collected)

\* Bank account details

\* Credit scores

\* Detailed income information

\* Health data correlations

- Include these encryption features:

\* AES-256 encryption for data at rest

\* Key management and rotation

\* Encryption key derivation from user passwords

\* Secure key storage (environment variables/vault)

\* Database column encryption

\* File encryption for uploaded documents

- Add data masking for:

\* Credit card numbers (show last 4 digits)

\* Bank account numbers (show last 4 digits)

\* Social Security numbers (show last 4 digits)

\* Income details in logs

\* Personal information in error messages

- Implement these protection measures:

\* Data tokenization for payment processing

\* Secure data deletion (crypto-shredding)

\* Data integrity verification

\* Audit trail for data access

\* Compliance logging for financial regulations

This ensures MINGUS meets banking-level security standards for financial data protection.

**🔒 PHASE 3: COMPREHENSIVE SECURITY AUDIT (Day 5)**

**Prompt 7: Create Security Audit and Vulnerability Scanner**

Create a comprehensive security audit system for MINGUS that automatically scans for vulnerabilities and security issues.

Requirements:

- Create `security/audit.py` with automated security scanning

- Implement these vulnerability checks:

\* SQL injection vulnerability scanning

\* XSS vulnerability testing

\* Authentication bypass attempts

\* Authorization flaws detection

\* Session management vulnerabilities

\* File upload security testing

\* API security assessment

\* Configuration security review

- Include these security tests:

\* Password policy enforcement verification

\* SSL/TLS configuration testing

\* Security header verification

\* Cookie security validation

\* CSRF protection testing

\* Rate limiting effectiveness

\* Input validation coverage

\* Error handling security review

- Add automated testing for:

\* Payment processing security (Stripe integration)

\* User data protection

\* Financial calculation integrity

\* Health data privacy

\* Subscription security

\* Admin access controls

- Generate comprehensive security reports with:

\* Vulnerability severity ratings

\* Remediation recommendations

\* Compliance status (PCI DSS, GDPR considerations)

\* Security score and trending

This provides ongoing security monitoring and validation for the financial application.

**Prompt 8: Implement Security Logging and Monitoring**

Create a comprehensive security logging and monitoring system for MINGUS that tracks all security-relevant events.

Requirements:

- Create `security/logging.py` with detailed security event logging

- Log these critical security events:

\* Authentication attempts (success/failure)

\* Authorization failures

\* Input validation violations

\* Rate limiting triggers

\* Suspicious user behavior

\* Financial data access

\* Payment processing events

\* Admin actions

\* Configuration changes

\* Security policy violations

- Implement these monitoring features:

\* Real-time security alerts

\* Anomaly detection for user behavior

\* Failed login attempt clustering

\* Unusual financial transaction patterns

\* Suspicious API usage patterns

\* Geographic access anomalies

\* Time-based access pattern analysis

- Create security dashboards showing:

\* Current security status

\* Recent security events

\* Threat indicators

\* Compliance metrics

\* User risk scores

\* System security health

- Include integration with:

\* Digital Ocean monitoring

\* Email/SMS alerting for critical events

\* Log aggregation and analysis

\* Security incident response workflows

This provides real-time security visibility and incident response capabilities.

**Prompt 9: Create Compliance and Privacy Protection System**

Create a comprehensive compliance and privacy protection system for MINGUS that ensures regulatory compliance for financial and health data.

Requirements:

- Create `security/compliance.py` with privacy and compliance controls

- Implement GDPR compliance features:

\* Data consent management

\* Right to access (data export)

\* Right to deletion (data erasure)

\* Data portability

\* Privacy policy enforcement

\* Cookie consent management

\* Data processing audit trails

- Add financial compliance features:

\* PCI DSS considerations for payment data

\* Financial data retention policies

\* Audit trail requirements

\* Data breach notification procedures

\* Customer data protection

- Implement health data privacy (HIPAA considerations):

\* Health data encryption

\* Access controls for health information

\* Health data anonymization

\* Consent for health tracking

\* Health data retention policies

- Include these privacy controls:

\* Data minimization (collect only necessary data)

\* Purpose limitation (use data only for stated purposes)

\* Data accuracy requirements

\* Storage limitation (automatic data deletion)

\* Transparency (clear privacy notices)

- Create compliance reporting and audit features

- Add user privacy dashboard

- Implement data subject request handling

This ensures MINGUS meets all relevant privacy and compliance requirements for a financial wellness application.

**🔒 PHASE 4: PRODUCTION SECURITY DEPLOYMENT (Day 5-6)**

**Prompt 10: Create Production Security Configuration**

Create production-ready security configuration for deploying MINGUS to Digital Ocean with maximum security.

Requirements:

- Create `security/production\_config.py` for production security settings

- Configure Digital Ocean security features:

\* VPC networking configuration

\* Firewall rules (allow only necessary ports)

\* Load balancer SSL termination

\* Database security groups

\* App platform security settings

\* CDN security configuration

- Implement environment-specific security:

\* Production vs staging vs development

\* Environment variable security

\* Secret management (database passwords, API keys)

\* Configuration validation

\* Security policy enforcement

- Add deployment security checks:

\* Pre-deployment security validation

\* SSL certificate verification

\* Security header testing in production

\* Database connection security verification

\* API endpoint security testing

- Configure production monitoring:

\* Security event alerting

\* Performance monitoring with security focus

\* Uptime monitoring

\* SSL certificate expiration alerts

\* Security scan scheduling

- Include disaster recovery and backup security:

\* Encrypted backups

\* Secure backup storage

\* Backup access controls

\* Recovery procedure security

This ensures maximum security for the production deployment on Digital Ocean.

**Prompt 11: Create Security Testing and Penetration Testing Suite**

Create a comprehensive security testing suite for MINGUS that validates all security measures are working correctly.

Requirements:

- Create `security/security\_tests.py` with automated security testing

- Implement these security test categories:

\* Authentication security tests

\* Authorization and access control tests

\* Input validation and sanitization tests

\* Session management security tests

\* API security tests

\* Data encryption and protection tests

\* Payment processing security tests

\* File upload security tests

- Add penetration testing scenarios:

\* Simulated SQL injection attempts

\* XSS attack simulations

\* CSRF attack testing

\* Session hijacking attempts

\* Brute force attack simulations

\* Privilege escalation testing

\* Data exposure testing

\* API abuse testing

- Include performance security testing:

\* DDoS resilience testing

\* Rate limiting effectiveness

\* Resource exhaustion testing

\* Concurrent user security testing

- Create security regression testing:

\* Automated security test suite

\* Continuous security testing

\* Security test reporting

\* Integration with CI/CD pipeline

- Generate detailed security test reports with:

\* Test results and coverage

\* Vulnerability findings

\* Security posture assessment

\* Remediation recommendations

This provides comprehensive validation that all security measures are working effectively.

**Prompt 12: Create Security Incident Response System**

Create a comprehensive security incident response system for MINGUS that handles security breaches and threats.

Requirements:

- Create `security/incident\_response.py` with automated incident handling

- Implement incident detection for:

\* Data breach attempts

\* Unauthorized access attempts

\* Payment fraud attempts

\* Account compromise indicators

\* System intrusion attempts

\* Data exfiltration attempts

\* Service disruption attacks

- Add automated response capabilities:

\* Account lockout for suspicious activity

\* IP blocking for malicious requests

\* Automatic scaling for DDoS attacks

\* Data access restrictions during incidents

\* Payment processing suspension if needed

\* Admin notification and escalation

- Create incident response workflows:

\* Incident classification and severity

\* Response team notification

\* Evidence collection and preservation

\* Customer communication procedures

\* Regulatory notification requirements

\* Post-incident analysis and remediation

- Include recovery procedures:

\* System restoration processes

\* Data integrity verification

\* Service restoration validation

\* Security measure enhancement

\* Incident documentation and learning

- Add compliance incident handling:

\* GDPR breach notification (72-hour requirement)

\* Customer notification procedures

\* Regulatory reporting

\* Legal team coordination

This ensures MINGUS can effectively respond to and recover from security incidents.

**🔒 PHASE 5: SECURITY MAINTENANCE AND MONITORING (Day 6-7)**

**Prompt 13: Create Security Update and Patch Management System**

Create a comprehensive security update and patch management system for MINGUS that keeps all security measures current.

Requirements:

- Create `security/patch\_management.py` for automated security updates

- Implement dependency security monitoring:

\* Python package vulnerability scanning

\* JavaScript dependency security checks

\* Database security update monitoring

\* Operating system security updates

\* Third-party service security advisories

- Add automated security updates:

\* Critical security patch deployment

\* Dependency update automation

\* Security configuration updates

\* Certificate renewal automation

\* Security policy updates

- Create security change management:

\* Security update testing procedures

\* Rollback procedures for security changes

\* Security update documentation

\* Change approval workflows

\* Emergency security update procedures

- Include vulnerability management:

\* Vulnerability scanning schedule

\* Vulnerability assessment and prioritization

\* Patch deployment timeline

\* Vulnerability remediation tracking

\* Security metric reporting

- Add security awareness and training:

\* Security policy documentation

\* Security training materials

\* Security best practices

\* Incident response training

\* Security culture development

This ensures MINGUS maintains strong security posture over time with automated updates and monitoring.

**Prompt 14: Create Security Performance and Optimization System**

Create a security performance optimization system for MINGUS that ensures security measures don't impact user experience.

Requirements:

- Create `security/performance\_optimization.py` for security performance tuning

- Optimize these security features for performance:

\* Authentication and session management

\* Input validation and sanitization

\* Encryption and decryption operations

\* Rate limiting and throttling

\* Security logging and monitoring

\* Vulnerability scanning

- Implement performance monitoring for security:

\* Security operation timing

\* Authentication performance metrics

\* Encryption overhead measurement

\* Rate limiting impact analysis

\* Security middleware performance

\* Database security query optimization

- Add security caching strategies:

\* Authentication result caching

\* Rate limiting state caching

\* Validation result caching

\* Security configuration caching

\* Threat intelligence caching

- Create security load balancing:

\* Security service distribution

\* Encryption workload balancing

\* Authentication service scaling

\* Security monitoring distribution

- Include user experience optimization:

\* Progressive security validation

\* Asynchronous security operations

\* Security feedback optimization

\* Error message optimization

\* Security feature graceful degradation

This ensures MINGUS security measures enhance rather than hinder the user experience.

**Prompt 15: Create Final Security Validation and Certification System**

Create a final security validation and certification system that verifies MINGUS meets all security requirements for a production financial application.

Requirements:

- Create `security/final\_validation.py` for comprehensive security certification

- Implement complete security checklist validation:

\* Security headers verification

\* Input validation coverage testing

\* Authentication security validation

\* Authorization control verification

\* Data encryption validation

\* API security confirmation

\* Compliance requirement verification

- Add security certification testing:

\* Penetration testing execution

\* Vulnerability assessment completion

\* Security control effectiveness testing

\* Incident response testing

\* Disaster recovery testing

\* Performance security testing

- Create security documentation generation:

\* Security architecture documentation

\* Security policy documentation

\* Incident response procedures

\* Compliance certification reports

\* Security training materials

- Include production readiness assessment:

\* Security deployment checklist

\* Production security configuration validation

\* Monitoring and alerting verification

\* Backup and recovery validation

\* Business continuity verification

- Generate final security certification:

\* Security posture assessment

\* Risk assessment and mitigation

\* Compliance status certification

\* Security readiness confirmation

\* Production deployment approval

This provides final validation that MINGUS is secure and ready for production deployment with financial data.

**🎯 EXECUTION ORDER AND TIMELINE**

**Day 4 (Security Foundation):**

1. **Prompt 1** - Security Headers Implementation
2. **Prompt 2** - HTTPS and SSL Security
3. **Prompt 3** - Authentication Security
4. **Prompt 4** - Input Validation System

**Day 5 (Security Enhancement):**

1. **Prompt 5** - API Security and Rate Limiting
2. **Prompt 6** - Data Encryption and Protection
3. **Prompt 7** - Security Audit and Vulnerability Scanner
4. **Prompt 8** - Security Logging and Monitoring

**Day 6 (Production Security):**

1. **Prompt 9** - Compliance and Privacy Protection
2. **Prompt 10** - Production Security Configuration
3. **Prompt 11** - Security Testing and Penetration Testing
4. **Prompt 12** - Security Incident Response

**Day 7 (Security Optimization):**

1. **Prompt 13** - Security Update and Patch Management
2. **Prompt 14** - Security Performance Optimization
3. **Prompt 15** - Final Security Validation and Certification

**🔒 SECURITY VALIDATION CRITERIA**

After completing all prompts, MINGUS should have:

**Security Headers:**

* ✅ Content Security Policy (CSP) blocking unsafe scripts
* ✅ HSTS forcing HTTPS for all connections
* ✅ XSS protection preventing script injection
* ✅ Clickjacking protection via X-Frame-Options
* ✅ MIME sniffing prevention

**Input Validation:**

* ✅ SQL injection prevention on all database queries
* ✅ XSS prevention on all user inputs
* ✅ Financial data validation preventing manipulation
* ✅ Health data range validation
* ✅ File upload security for documents

**Security Audit:**

* ✅ Automated vulnerability scanning
* ✅ Penetration testing capabilities
* ✅ Security logging and monitoring
* ✅ Compliance validation (GDPR, financial regulations)
* ✅ Incident response procedures

**Production Security:**

* ✅ SSL/TLS encryption for all communications
* ✅ Database encryption for sensitive data
* ✅ Rate limiting preventing abuse
* ✅ Authentication security with lockout protection
* ✅ API security with proper authorization

**This security hardening transforms MINGUS into a bank-grade secure financial application ready for production deployment with sensitive user data.**