**Cursor Prompts for Mingus Production Environment Setup**

**1. Production Environment Configuration (.env)**

**Prompt:**

Create a production-ready .env configuration file for a Python Flask/Django personal finance app called Mingus. Include:

- Database connection settings for PostgreSQL

- Security keys and tokens (with placeholder values)

- Email service configuration (SendGrid or similar)

- API keys for financial data services

- Redis configuration for caching

- SSL and security settings

- Logging levels

- Domain and URL configurations

- Payment processing variables (Stripe)

Also create a .env.example file that shows the structure without sensitive values. Include comments explaining each variable's purpose.

**2. Database Setup (PostgreSQL Migration)**

**Prompt:**

Help me migrate my Mingus personal finance app from SQLite to PostgreSQL for production. Create:

1. Database migration scripts to transfer existing user data, budgets, expenses, and health check-in data

2. Updated database configuration in my Python app to use PostgreSQL

3. Connection pooling setup for better performance

4. Database backup and restore scripts

5. User table, budget table, expense table, and health\_checkin table schemas optimized for PostgreSQL

6. Indexes for common queries (user lookups, date ranges, budget calculations)

Assume I'm using either Flask-SQLAlchemy or Django ORM. Ask me which one I'm using first.

**3. SSL Certificates and Security Headers**

**Prompt:**

Set up production-grade security for my Mingus personal finance app including:

1. SSL certificate configuration (Let's Encrypt or custom certificate)

2. Security headers middleware (HSTS, CSP, X-Frame-Options, etc.)

3. HTTPS redirect configuration

4. CORS settings for my frontend

5. API rate limiting implementation

6. Input validation and sanitization for financial data

7. Password hashing and authentication security

8. Session management security

9. Environment-specific security configurations

Create both the configuration files and the implementation code. The app handles sensitive financial data so security is critical.

**4. Logging and Monitoring Setup**

**Prompt:**

Create a comprehensive logging and monitoring system for my Mingus personal finance app production environment:

1. Structured logging configuration (JSON format)

2. Different log levels for different environments (DEBUG, INFO, WARNING, ERROR)

3. Log rotation and management

4. Application performance monitoring setup

5. Database query monitoring

6. User activity logging (without sensitive data)

7. Error tracking and alerting system

8. Health check endpoints for monitoring services

9. Metrics collection for key business indicators (user signups, budget creation, etc.)

10. Integration with monitoring services like DataDog, New Relic, or open-source alternatives

Include sample code for logging important events like user registration, budget creation, and financial calculations.

**5. Complete Production Deployment Configuration**

**Prompt:**

Create a complete production deployment setup for my Mingus personal finance app. Include:

1. Docker containerization with multi-stage builds

2. Docker Compose for production with PostgreSQL, Redis, and app containers

3. Nginx reverse proxy configuration with SSL termination

4. Gunicorn or uWSGI configuration for Python app serving

5. Environment variable management across containers

6. Health checks and restart policies

7. Volume management for persistent data

8. Network security between containers

9. Resource limits and scaling considerations

10. Deployment scripts and CI/CD pipeline basic setup

The app serves 1,000 users across three pricing tiers ($10, $20, $50) and handles personal financial data, so reliability and security are crucial.

**6. Database Performance and Optimization**

**Prompt:**

Optimize my PostgreSQL database for the Mingus personal finance app production environment:

1. Create optimized table schemas for users, budgets, expenses, income, and health check-ins

2. Design proper indexes for common query patterns (user dashboards, date-range reports, budget calculations)

3. Set up database connection pooling (PgBouncer or similar)

4. Configure query optimization and explain plans

5. Set up automated backup strategies with retention policies

6. Database monitoring and performance tracking

7. Partitioning strategies for large tables (expenses over time)

8. Cache strategies using Redis for frequently accessed data

9. Database security configurations (user permissions, SSL connections)

Focus on supporting 1,000 active users with financial forecasting and health data correlations.

**7. Security Hardening and Compliance**

**Prompt:**

Implement security hardening for my Mingus personal finance app to protect user financial data:

1. Data encryption at rest and in transit

2. PCI DSS compliance considerations for payment processing

3. User data privacy controls (GDPR/CCPA compliance)

4. API security (authentication, authorization, input validation)

5. Database security (encrypted connections, user permissions)

6. Secure file upload handling

7. Audit logging for sensitive operations

8. Data anonymization for analytics

9. Secure session management

10. Vulnerability scanning and security testing setup

11. Incident response procedures

12. Data retention and deletion policies

Create both the implementation code and security policies documentation.

**Usage Instructions:**

1. **Copy each prompt** into Cursor one at a time
2. **Start with prompt #1** (Environment Configuration) as it sets the foundation
3. **Customize the responses** based on your specific tech stack (Flask vs Django, etc.)
4. **Test each component** in a staging environment before production
5. **Ask follow-up questions** in Cursor if you need clarification on any generated code

**Pro Tips for Using These Prompts:**

* **Be specific about your current setup** when Cursor asks clarifying questions
* **Request step-by-step implementation guides** if the initial response is too complex
* **Ask for testing procedures** to verify each component works correctly
* **Request documentation** for future maintenance and team onboarding