MongoDB Models Production Implementation Guide

This guide focuses on preparing the individual MongoDB models of Travian Whispers for production use, with special attention to removing mock data and ensuring data integrity.

1. User Model (`database/models/user.py`)

Production Readiness

The User model is generally well-structured with proper validation, but needs some refinements:

Updates Required:

1. **Password Security Enhancement**

python Copy

```python
def hash_password(self, password):
    """
    Hash a password using passlib with improved security parameters.
    """
    # Increase rounds for better security in production
    return pbkdf2_sha256.using(rounds=350000, salt_size=32).hash(password)
```

2. **Add Method for Admin Reset**

python Copy

```python
def admin_reset_password(self, user_id, new_password):
    """
    Reset a user's password (admin function).

    Args:
        user_id (str): User ID
        new_password (str): New password

    Returns:
        bool: True if reset successful, False otherwise
    """
    if self.collection is None:
        return False

    try:
        hashed_password = self.hash_password(new_password)
        result = self.collection.update_one(
            {"_id": ObjectId(user_id)},
            {"$set": {
                "password": hashed_password,
                "updatedAt": datetime.utcnow()
            }}
        )
        return result.modified_count > 0
    except Exception as e:
        logger.error(f"Error resetting password: {e}")
        return False
```

3. **Add Method to Delete User**

python Copy

```python
def delete_user(self, user_id):
    """
    Delete a user.

    Args:
        user_id (str): User ID

    Returns:
        bool: True if successful, False otherwise
    """
    if self.collection is None:
        return False

    try:
        result = self.collection.delete_one({"_id": ObjectId(user_id)})
        return result.deleted_count > 0
    except Exception as e:
        logger.error(f"Error deleting user: {e}")
        return False
```

4. **Add Verification by ID Method**

python Copy

```python
def verify_user_by_id(self, user_id):
    """
    Verify a user by ID (for admin-created accounts).

    Args:
        user_id (str): User ID

    Returns:
        bool: True if successful, False otherwise
    """
    if self.collection is None:
        return False

    try:
        result = self.collection.update_one(
            {"_id": ObjectId(user_id)},
            {"$set": {
                "isVerified": True,
                "updatedAt": datetime.utcnow()
            }}
        )
        return result.modified_count > 0
    except Exception as e:
        logger.error(f"Error verifying user: {e}")
        return False
```

2. Subscription Model (`database/models/subscription.py`)

Production Readiness

The Subscription model is well-structured but requires some refinements for production:

Updates Required:

1. **Method to Check Plan Availability**

python Copy

```python
def plan_exists(self, plan_id):
    """
    Check if a plan exists.

    Args:
        plan_id (str): Plan ID

    Returns:
        bool: True if plan exists, False otherwise
    """
    if self.collection is None:
        return False

    try:
        count = self.collection.count_documents({"_id": ObjectId(plan_id)})
        return count > 0
    except Exception as e:
        logger.error(f"Error checking plan existence: {e}")
        return False
```

2. **Method to Get Plan Feature**

python Copy

```python
def get_plan_feature(self, plan_id, feature_name):
    """
    Get a specific feature value for a plan.

    Args:
        plan_id (str): Plan ID
        feature_name (str): Feature name

    Returns:
        Feature value or None if not found
    """
    if self.collection is None:
        return None

    try:
        plan = self.get_plan_by_id(plan_id)
        if not plan or "features" not in plan:
            return None
        return plan["features"].get(feature_name)
    except Exception as e:
        logger.error(f"Error retrieving plan feature: {e}")
        return None
```

3. Transaction Model (`database/models/transaction.py`)

Production Readiness

The Transaction model appears ready for production but would benefit from additional methods:

Updates Required:

1. **Method to Aggregate Revenue Data**

python Copy

```python
def get_revenue_stats(self, start_date=None, end_date=None):
    """
    Get revenue statistics for a date range.

    Args:
        start_date (datetime, optional): Start date
        end_date (datetime, optional): End date

    Returns:
        dict: Revenue statistics
    """
    if self.collection is None:
        return {"total": 0, "count": 0, "average": 0}

    try:
        query = {"status": "completed"}

        if start_date:
            if not end_date:
                end_date = datetime.utcnow()

            query["createdAt"] = {
                "$gte": start_date,
                "$lte": end_date
            }

        pipeline = [
            {"$match": query},
            {"$group": {
                "_id": None,
                "total": {"$sum": "$amount"},
                "count": {"$sum": 1},
                "average": {"$avg": "$amount"}
            }}
        ]

        result = list(self.collection.aggregate(pipeline))

        if result:
            stats = result[0]
            stats.pop("_id", None)  # Remove _id field
            return stats
```

```
        else:
            return {"total": 0, "count": 0, "average": 0}
    except Exception as e:
        logger.error(f"Error getting revenue stats: {e}")
        return {"total": 0, "count": 0, "average": 0}
```

2. **Method to Get Monthly Revenue Data**

python Copy

```python
def get_monthly_revenue(self, year=None, month=None):
    """
    Get monthly revenue data.

    Args:
        year (int, optional): Year (defaults to current year)
        month (int, optional): Month (defaults to current month)

    Returns:
        float: Total revenue for the month
    """
    if self.collection is None:
        return 0.0

    try:
        if not year:
            year = datetime.utcnow().year

        if not month:
            month = datetime.utcnow().month

        start_date = datetime(year, month, 1)

        # Calculate end date (first day of next month)
        if month == 12:
            end_date = datetime(year + 1, 1, 1)
        else:
            end_date = datetime(year, month + 1, 1)

        pipeline = [
            {"$match": {
                "createdAt": {"$gte": start_date, "$lt": end_date},
                "status": "completed"
            }},
            {"$group": {
                "_id": None,
                "total": {"$sum": "$amount"}
            }}
        ]

        result = list(self.collection.aggregate(pipeline))
```

```
        if result:
            return result[0]["total"]
        else:
            return 0.0
    except Exception as e:
        logger.error(f"Error getting monthly revenue: {e}")
        return 0.0
```

4. Activity Models (`database/models/activity.py` and `database/models/log.py`)

Production Readiness

These models need additional methods to replace mock data in the admin routes:

Updates Required:

1. **Add Method to Get Recent Activity**

python

Copy

```python
# In ActivityLog class
def get_system_activity(self, limit=5):
    """
    Get recent system activity.

    Args:
        limit (int, optional): Maximum number of activities to return

    Returns:
        list: List of formatted activity logs
    """
    if self.collection is None:
        return []

    try:
        logs = self.collection.find().sort("timestamp", -1).limit(limit)

        formatted_logs = []
        for log in logs:
            formatted_logs.append({
                "timestamp": log["timestamp"].strftime('%Y-%m-%d %H:%M'),
                "username": log.get("username", "system"),
                "action": log["action"],
                "status": log["level"].capitalize(),
                "status_class": self._get_status_class(log["level"])
            })

        return formatted_logs
    except Exception as e:
        logger.error(f"Error getting system activity: {e}")
        return []

def _get_status_class(self, level):
    """Get Bootstrap status class based on log level."""
    if level.lower() == "info":
        return "bg-success"
    elif level.lower() == "warning":
        return "bg-warning"
    elif level.lower() == "error":
        return "bg-danger"
```

```
    else:
        return "bg-info"
```

5. IP Pool Model (`database/models/ip_pool.py`)

Production Readiness

The IP Pool model is well-structured but would benefit from additional error checking and a method to validate IP addresses:

Updates Required:

1. **Add IP Validation Method**

python Copy

```python
def validate_ip_address(self, ip_address):
    """
    Validate an IP address format.

    Args:
        ip_address (str): IP address to validate

    Returns:
        bool: True if valid, False otherwise
    """
    import re

    # Basic IPv4 validation pattern
    ipv4_pattern = r"^(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})$"
    match = re.match(ipv4_pattern, ip_address)

    if not match:
        return False

    # Check each octet is between 0 and 255
    for octet in match.groups():
        if int(octet) > 255:
            return False

    return True
```

6. Settings Model (`database/settings.py`)

Production Readiness

The Settings model needs to be updated to initialize with default settings:

Updates Required:

1. **Add Method to Initialize Default Settings**

python Copy

```python
def initialize_default_settings(self):
    """
    Initialize default settings if they don't exist.

    Returns:
        bool: True if successful, False otherwise
    """
    if self.collection is None:
        logger.error("Database not connected")
        return False

    default_settings = {
        'general.site_name': 'Travian Whispers',
        'general.site_description': 'Advanced Travian Automation Suite',
        'general.timezone': 'UTC',
        'general.maintenance_mode': False,
        'general.maintenance_message': 'We are currently performing scheduled maintena

        'email.smtp_server': 'smtp.gmail.com',
        'email.smtp_port': 587,
        'email.smtp_security': 'starttls',
        'email.smtp_username': '',
        'email.smtp_password': '',
        'email.sender_email': 'noreply@travianwhispers.com',
        'email.sender_name': 'Travian Whispers',

        'payment.currency': 'USD',
        'payment.currency_position': 'before',

        'security.email_verification': True,
        'security.session_timeout': 60,
        'security.max_login_attempts': 5,
        'security.account_lock_duration': 30,
        'security.password_policy': 'standard',
        'security.force_https': True,
        'security.enable_hsts': True,

        'backup.auto_backup': True,
        'backup.backup_frequency': 'daily',
        'backup.backup_time': '02:00',
        'backup.backup_type': 'full',
        'backup.compress_backups': True,
```

```
        'backup.retention_period': 30,
        'backup.max_backups': 10,
        'backup.backup_location': 'backups'
    }

    try:
        success = True
        for key, value in default_settings.items():
            # Only set if the setting doesn't already exist
            if self.get_setting(key) is None:
                if not self.update_setting(key, value):
                    success = False

        return success
    except Exception as e:
        logger.error(f"Failed to initialize default settings: {e}")
        return False
```

7. Application Initialization Script

Create a script to properly initialize the database:

python Copy

```python
# scripts/init_database.py
import logging
from database.mongodb import MongoDB
from database.models.subscription import SubscriptionPlan
from database.settings import Settings

# Configure logger
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger('init_database')

def initialize_database():
    """Initialize the database with required collections and default data."""
    logger.info("Starting database initialization")

    # Connect to database
    db = MongoDB()
    if not db.connect():
        logger.error("Failed to connect to MongoDB")
        return False

    # Create indexes
    logger.info("Creating database indexes")
    db.create_indexes()

    # Initialize subscription plans
    logger.info("Initializing subscription plans")
    subscription_model = SubscriptionPlan()
    if subscription_model.collection.count_documents({}) == 0:
        logger.info("No subscription plans found, creating defaults")
        subscription_model.create_default_plans()

    # Initialize settings
    logger.info("Initializing application settings")
    settings_model = Settings()
    settings_model.initialize_default_settings()

    logger.info("Database initialization completed successfully")
    return True
```

```
if __name__ == "__main__":
    initialize_database()
```

8. Testing Database Connection

Create a script to test the database connection:

python Copy

```python
# scripts/test_connection.py
import logging
from database.mongodb import MongoDB

# Configure logger
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger('test_connection')

def test_connection():
    """Test MongoDB connection and database access."""
    logger.info("Testing MongoDB connection")

    db = MongoDB()
    if db.connect():
        logger.info("Successfully connected to MongoDB")

        # Test retrieving a database and collection
        database = db.get_db()
        if database:
            logger.info(f"Successfully accessed database: {database.name}")

            # List collections
            collections = database.list_collection_names()
            logger.info(f"Collections: {', '.join(collections)}")

            return True
        else:
            logger.error("Failed to access database")
            return False
    else:
        logger.error("Failed to connect to MongoDB")
        return False

if __name__ == "__main__":
    test_connection()
```

9. Instructions for Removing Mock Data from Routes

To remove mock data from routes, follow these patterns:

In Admin Dashboard Route

Replace:

python Copy

```python
# Recent activity logs
recent_activity = [
    {
        "timestamp": datetime.now().strftime('%Y-%m-%d %H:%M'),
        "username": "admin",
        "action": "System backup created",
        "status": "Success",
        "status_class": "bg-success"
    },
    # More mock entries...
]
```

With:

python Copy

```python
# Get recent activity logs from database
from database.models.log import ActivityLog
activity_log = ActivityLog()
recent_activity = activity_log.get_system_activity(limit=5)
```

In User Dashboard Route

Replace:

python Copy

```python
# Prepare auto farm data
auto_farm_data = {
    'status': 'active' if user['settings'].get('autoFarm', False) else 'stopped',
    'interval': 60,  # Default interval
    'last_run': 'Never',  # This would be retrieved from activity logs
    'next_run': 'N/A',  # This would be calculated based on last run and interval
    'villages': user['villages']
}
```

With:

python Copy

```python
# Get actual auto farm data
from database.models.auto_farm import AutoFarm
auto_farm_model = AutoFarm()
auto_farm_config = auto_farm_model.get_user_config(session['user_id'])

if auto_farm_config:
    auto_farm_data = {
        'status': auto_farm_config.get('status', 'stopped'),
        'interval': auto_farm_config.get('interval', 60),
        'last_run': auto_farm_config.get('lastRun', 'Never'),
        'next_run': auto_farm_config.get('nextRun', 'N/A'),
        'villages': user['villages']
    }
else:
    # Default values if no config exists
    auto_farm_data = {
        'status': 'stopped',
        'interval': 60,
        'last_run': 'Never',
        'next_run': 'N/A',
        'villages': user['villages']
    }
```

10. Production Configuration Template

Create a production configuration template file:

python Copy

```python
# config/production.py
"""
Production configuration for Travian Whispers application.
"""
import os
from datetime import timedelta

# MongoDB Configuration
MONGODB_URI = os.getenv('MONGODB_URI', 'mongodb://username:password@hostname:27017/whi
MONGODB_DB_NAME = os.getenv('MONGODB_DB_NAME', 'whispers')

# Session Configuration
SECRET_KEY = os.getenv('SECRET_KEY', 'generate-a-secure-random-key')
SESSION_TYPE = 'filesystem'
SESSION_PERMANENT = True
PERMANENT_SESSION_LIFETIME = timedelta(minutes=60)

# Email Configuration
SMTP_SERVER = os.getenv('SMTP_SERVER', 'smtp.example.com')
SMTP_PORT = int(os.getenv('SMTP_PORT', '587'))
SMTP_USERNAME = os.getenv('SMTP_USERNAME', '')
SMTP_PASSWORD = os.getenv('SMTP_PASSWORD', '')
EMAIL_FROM = os.getenv('EMAIL_FROM', 'Travian Whispers <noreply@example.com>')

# Payment Gateway Configuration
PAYPAL_CLIENT_ID = os.getenv('PAYPAL_CLIENT_ID', '')
PAYPAL_SECRET = os.getenv('PAYPAL_SECRET', '')
PAYPAL_MODE = os.getenv('PAYPAL_MODE', 'live')  # 'sandbox' or 'live'

# Backup Configuration
BACKUP_DIR = os.getenv('BACKUP_DIR', 'backups')
BACKUP_RETENTION_DAYS = int(os.getenv('BACKUP_RETENTION_DAYS', '30'))

# Maintenance Mode
MAINTENANCE_MODE = False
MAINTENANCE_MESSAGE = 'We are currently performing scheduled maintenance. Please check

# Security Configuration
DEBUG = False
TESTING = False
```

Save this file and load it appropriately in your application's initialization.