MongoDB Production Setup Guide for Travian Whispers

This guide explains how to prepare the MongoDB database infrastructure for production use in the Travian Whispers application. The guide covers connection configuration, security, indexing, backup procedures, and removing mock data.

1. MongoDB Server Setup

1.1 Installation & Configuration

For production, use one of these deployment options:

- **MongoDB Atlas** (recommended for ease of management)
- **Self-hosted MongoDB** (for complete control)
- **MongoDB Replica Set** (for high availability)

1.2 Security Configuration

Configure MongoDB with proper authentication:

bash Copy

```bash
# Create admin user
use admin
db.createUser({
  user: "admin",
  pwd: "strong-password-here",
  roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
})

# Create application user
use whispers
db.createUser({
  user: "travian_app",
  pwd: "strong-app-password-here",
  roles: [ { role: "readWrite", db: "whispers" } ]
})
```

Update MongoDB configuration (`mongod.conf`) to enable authentication:

yaml Copy

```yaml
security:
  authorization: enabled
```

1.3 Network Security

- Enable TLS/SSL for encrypted connections
- Configure firewall to restrict MongoDB access
- Use a VPN or private network for database connections when possible

2. Application Configuration

2.1 Update Environment Variables

Store sensitive database information in environment variables or a secure configuration service.

Create a `.env` file:

Copy

```
MONGODB_URI=mongodb://travian_app:strong-app-password-here@mongodb.example.com:27017/w
MONGODB_DB_NAME=whispers
```

2.2 Update MongoDB Connection Code

Review and update the connection code in `database/mongodb.py`:

python Copy

```python
# Import environment variables
import os
from dotenv import load_dotenv
load_dotenv()

# In MongoDB.connect method:
connection_string = connection_string or os.getenv('MONGODB_URI')
db_name = db_name or os.getenv('MONGODB_DB_NAME')
```

2.3 Connection Pooling Settings

Adjust connection pooling based on expected load:

python Copy

```python
self.client = MongoClient(
    connection_string,
    serverSelectionTimeoutMS=5000,
    connectTimeoutMS=10000,
    socketTimeoutMS=45000,
    maxPoolSize=100,  # Adjust based on expected concurrent connections
    waitQueueTimeoutMS=2500,
    retryWrites=True,
    retryReads=True
)
```

3. Create Required Indexes

Ensure all necessary indexes are created at application startup by implementing the `create_indexes` method from `database/mongodb.py`.

Create a script to initialize all indexes:

python Copy

```python
# scripts/init_db.py
from database.mongodb import MongoDB
from database.models.user import User
from database.models.ip_pool import IPAddress
from database.models.proxy_service import ProxyService
from database.models.transaction import Transaction
# Import other models as needed

def create_all_indexes():
    """Create all required indexes for the application."""
    db = MongoDB()
    db.connect()

    # Create MongoDB built-in indexes
    db.create_indexes()

    # Create model-specific indexes
    IPAddress().create_indexes()
    ProxyService().create_indexes()

    print("All indexes created successfully.")

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

Run this script during deployment.

4. Setup Backup Procedures

4.1 Configure Regular Backups

Implement scheduled backups using the `database/backup.py` module:

python Copy

```python
# In your application startup:
from apscheduler.schedulers.background import BackgroundScheduler
from database.backup import create_backup

def schedule_backups():
    scheduler = BackgroundScheduler()
    # Daily backup at 2:00 AM
    scheduler.add_job(create_backup, 'cron', hour=2, minute=0)
    scheduler.start()
```

4.2 Backup Retention

Configure retention policy in settings:

python Copy

```python
# Access with settings model
settings_model = Settings()
retention_days = settings_model.get_setting('backup.retention_period', 30)

# Use in backup cleanup
from database.backup import cleanup_old_backups
cleanup_old_backups(keep_last=retention_days)
```

4.3 Backup Storage

For production, configure backup storage options:

- Local backup with rotation
- Cloud storage (AWS S3, Google Cloud Storage)
- Separate backup server

5. Removing Mock Data

5.1 Identify Mock Data Sources

The main sources of mock data in the codebase are:

1. Hardcoded data in route handlers
2. Default methods creating test data
3. Sample data in controller functions

5.2 Remove Mock Data in User Routes

In `web/routes/user.py`, remove mock data:

python Copy

```python
# Replace mock activity logs with actual database queries
@user_bp.route('/activity-logs')
@login_required
def activity_logs():
    # Get user data
    user_model = User()
    user = user_model.get_user_by_id(session['user_id'])

    if not user:
        flash('User not found', 'danger')
        session.clear()
        return redirect(url_for('auth.login'))

    # Get activity logs from database (replace mock data)
    from database.models.activity import UserActivity
    activity_model = UserActivity()
    activity_logs = activity_model.get_recent_activities(session['user_id'], limit=10)

    # Render activity logs template
    return render_template(
        'user/activity_logs.html',
        logs=activity_logs,
        current_user=user,
        title='Activity Logs'
    )
```

5.3 Remove Mock Data in Admin Routes

In `web/routes/admin.py`, replace hardcoded stats with database queries:

python Copy

```python
# Replace mock recent activity logs:
from database.models.log import ActivityLog
log_model = ActivityLog()
recent_activity = log_model.get_recent_logs(limit=5)
```

5.4 Default Plan Creation

The `SubscriptionPlan.create_default_plans()` method should only be called during initial setup, not in regular operation:

python Copy

```python
# Call this only during initial database setup, not on every application start
if subscription_model.collection.count_documents({}) == 0:
    subscription_model.create_default_plans()
```

6. Production Deployment Checklist

Before deploying to production:

- [x] Set up MongoDB with authentication and TLS
- [x] Configure connection string with environment variables
- [x] Ensure all indexes are created
- [x] Set up regular backup procedures
- [x] Configure proper logging
- [x] Remove all mock data
- [x] Test the system with production database

7. Monitoring and Maintenance

For ongoing monitoring:

1. Set up MongoDB monitoring (CPU, memory, disk usage)
2. Monitor query performance with `explain()`
3. Review slow queries and optimize as needed
4. Regularly review and update indexes
5. Monitor connection pool usage and adjust as needed

Appendix: Required Collections

The following collections should exist in your MongoDB database:

1. `users` - User accounts
2. `subscriptionPlans` - Available subscription plans
3. `transactions` - Payment transactions

4. `villages` - User villages
5. `autoFarm` - Auto-farm configurations
6. `troopTrainer` - Troop training configurations
7. `ipAddresses` - IP pool management
8. `proxyServices` - Proxy provider configurations
9. `activityLogs` - User activity logs
10. `systemLogs` - System activity logs
11. `settings` - Application settings
12. `backupRecords` - Backup history