SQLite to PostgreSQL Fix for Azure Deployment

Problem Summary

The Azure SaaS deployment was incorrectly using SQLite instead of PostgreSQL, causing database errors:

```
ERROR: (sqlite3.OperationalError) no such table: organizations
```

Root Cause Analysis

1. **Dangerous SQLite Fallback in startup.sh**

The startup script had this line:

```
export DATABASE_URL=${DATABASE_URL:-"sqlite:///./azure_app.db"}
```

This defaulted to SQLite if DATABASE_URL wasn't set, even in production.

2. **Multiple SQLite Fallbacks in config.py**

The configuration file had several fallback mechanisms that would silently use SQLite when DATABASE_URL wasn't properly configured.

3. **Missing ENVIRONMENT Variable**

The ENVIRONMENT variable wasn't being set to "production" in Azure, causing the code to treat it as an unknown environment and fall back to unsafe defaults.

Solution Implemented

1. **startup.sh - Strict Validation**

```
# CRITICAL: Do NOT default to SQLite in production
if [ -z "$DATABASE_URL" ]; then
    echo "ERROR: DATABASE_URL environment variable is not set!"
    exit 1
fi

# Validate that we're not using SQLite in production
if [[ "$DATABASE_URL" == sqlite* ]]; then
    echo "ERROR: SQLite database detected in production!"
    exit 1
fi
```

```
# Set ENVIRONMENT to production
export ENVIRONMENT=${ENVIRONMENT:-"production"}
```

2. **app/config.py - Fail Loudly in Production**

- Added strict validation that prevents SQLite usage in production
- Raises `ValueError` if DATABASE_URL is not set in non-development environments
- Added secondary validation to reject SQLite connections in production

3. **app/db/base.py - Triple-Layer Protection**

- Validates DATABASE_URL is set before creating engine
- Checks for SQLite usage in production environments
- Fails with clear error messages instead of falling back to SQLite

Files Modified

1. **startup.sh**
    - Added DATABASE_URL validation on startup
    - Added SQLite detection and rejection
    - Enforces ENVIRONMENT=production
2. **app/config.py**
    - Stricter DATABASE_URL validation
    - Added SQLite rejection in production
    - Better error messages with troubleshooting info
3. **app/db/base.py**
    - Enhanced database engine creation validation
    - Prevents SQLite fallback in production
    - Added detailed logging for debugging

Expected Behavior After Fix

In Production (Azure):

1. If DATABASE_URL is not set → **Application fails to start** with clear error message
2. If DATABASE_URL contains sqlite → **Application fails to start** with clear error message
3. If DATABASE_URL is valid PostgreSQL → **Application starts successfully**

In Development:

1. If DATABASE_URL is not set and ENVIRONMENT is "development" → Falls back to SQLite with warning
2. Otherwise → Uses the configured database

Verification Steps

After deployment, check the Azure logs for:

✅ **SUCCESS indicators:**

```
Environment configuration:
DATABASE_URL: postgresql://...
Database type: PostgreSQL
INFO: Using PostgreSQL database for production environment
```

❌ **FAILURE indicators (if DATABASE_URL not set):**

```
ERROR: DATABASE_URL environment variable is not set!
This is required for production deployment.
```

❌ **FAILURE indicators (if SQLite detected):**

```
ERROR: SQLite database detected in production!
Production deployments must use PostgreSQL.
```

Azure Configuration Requirements

Ensure these Application Settings are configured in Azure:

Setting	Required	Example Value
DATABASE_URL	✅ Yes	postgresql://user:pass@host:5432/db?sslmode=require
ENVIRONMENT	✅ Yes	production
SECRET_KEY	✅ Yes	your-secret-key
OPENAI_API_KEY	✅ Yes	sk-...
GOOGLE_API_KEY	Optional	...
TAVILY_API_KEY	Optional	...

How to Deploy the Fix

1. **Commit the changes:**

```
git add startup.sh app/config.py app/db/base.py
git commit -m "Fix: Prevent SQLite fallback in Azure production
```

```
deployment"
git push origin main
```

2. **Verify Azure Application Settings:**

```
az webapp config appsettings list \
  --name ai-event-planner-saas-py \
  --resource-group <your-resource-group>
```

3. **Ensure DATABASE_URL is set:**

```
az webapp config appsettings set \
  --name ai-event-planner-saas-py \
  --resource-group <your-resource-group> \
  --settings DATABASE_URL="postgresql://..." \
             ENVIRONMENT="production"
```

4. **Monitor deployment logs:**

```
az webapp log tail \
  --name ai-event-planner-saas-py \
  --resource-group <your-resource-group>
```

Troubleshooting

If the app still uses SQLite:

1. Check that DATABASE_URL is set in Azure App Service settings (not just GitHub secrets)
2. Verify ENVIRONMENT is set to "production"
3. Check startup logs for configuration details
4. Restart the app after setting environment variables

If the app fails to start:

1. Check logs for the specific error message
2. Verify DATABASE_URL format is correct
3. Ensure PostgreSQL database is accessible from Azure
4. Check that PostgreSQL connection string includes `?sslmode=require`

Prevention Measures

The fix implements three layers of protection:

1. **Startup Validation** - Prevents app from starting with invalid database config

2. **Config Validation** - Prevents code from loading with SQLite in production
3. **Engine Validation** - Prevents database engine creation with SQLite in production

This "fail loudly, fail early" approach ensures database configuration issues are caught immediately rather than causing runtime errors.

Related Documentation

- `AZURE_DEPLOYMENT_SETUP_GUIDE.md` - Complete Azure setup instructions
- `DEPLOYMENT_GUIDE.md` - General deployment guide
- `GITHUB_SECRETS_SETUP.md` - GitHub secrets configuration

Date

Created: January 25, 2025