Django CRUD App Development & AWS EC2 Deployment Guide

This comprehensive guide documents the complete process of developing a Django CRUD application with MySQL and deploying it to AWS EC2.

Table of Contents

Phase I: Local Django Setup

1. Application Registration

Action: Added `testapp` to `INSTALLED_APPS` in `aws/settings.py`

```
INSTALLED_APPS = [
    # ... other apps
    'testapp',
]
```

Result: ✅ Django project recognized the new application

2. MySQL Database Configuration

Action: Updated `DATABASES` setting in `aws/settings.py`

```
DATABASES = {
```

```
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'django_crud_db',
        'USER': 'root',
        'PASSWORD': '',  # Initially empty
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Result: ✅ Django configured for MySQL connection

3. Model Creation

Action: Defined `Task` model in `testapp/models.py` **Result:** ✅ Data structure established for task management

4. CRUD Views Implementation

Action: Created class-based views in `testapp/views.py`:

- `TaskListView` - Display all tasks
- `TaskCreateView` - Create new tasks
- `TaskUpdateView` - Edit existing tasks
- `TaskDeleteView` - Delete tasks

Result: ✅ Core CRUD functionality implemented

5. URL Configuration

Action:

- Defined URL patterns in `testapp/urls.py`
- Updated main `aws/urls.py` to include testapp URLs

Result: ✅ Application routing established

6. Template Creation

Action: Created HTML templates in `testapp/templates/testapp/`:

- `base.html` - Common layout
- `task_list.html` - Task display

- `task_form.html` - Create/edit form
- `task_confirm_delete.html` - Delete confirmation

Result: ✅ User interface prepared

Phase II: Database Configuration & Migration Issues

1. MySQL Database Creation

Question: Does `django_crud_db` need to be created in MySQL Workbench? **Answer:** Yes, created using:

CREATE DATABASE django_crud_db;

Result: ✅ Database created successfully

2. Password Configuration Update

Action: Updated MySQL password in `settings.py` to `'subbu@143'` **Result:** ✅ Database credentials updated

3. Migration Challenges

Issue: Access denied error: `(1045, "Access denied for user 'root'@'localhost' (using password: NO)")`

Troubleshooting Steps:

1. Verified MySQL Workbench connection ✅
2. Confirmed database and tables exist ✅
3. Deleted and recreated `migrations` directory
4. Re-ran `makemigrations` and `migrate`

Result: ⚠️ Local database connection issues identified (user access related)

Phase III: Version Control with GitHub

1. Git Repository Initialization

git init

2. .gitignore Creation

Action: Created `.gitignore` to exclude:

- Virtual environment files
- `.pyc` files
- `__pycache__` directories
- Other unnecessary files

3. Initial Commit

```
git add .
git commit -m "Initial commit: Set up Django CRUD app with MySQL"
```

4. GitHub Upload

Repository: https://github.com/shaik786143/django-crud-app.git **Result:** ✅ Codebase successfully hosted on GitHub

---

Phase IV: AWS EC2 Prerequisites

Required Infrastructure:

- ✅ Running EC2 instance
- ✅ SSH key pair for access
- ✅ Security Group configuration (ports 22, 80, 443)
- ✅ Local MySQL installation knowledge

EC2 Environment Setup:

Software Installation:

- Python 3.x
- pip (Python package manager)
- venv (Virtual environment)
- MySQL client

Dependencies File

Action: Generated `requirements.txt`:

pip freeze > requirements.txt

Result: ✅ Project dependencies documented

Phase V: EC2 Deployment & Dependency Management

1. Security Group Verification

Action: Confirmed inbound rules:

- SSH (Port 22) ✅
- HTTP (Port 80) ✅
- HTTPS (Port 443) ✅

2. Dependency Installation Challenges

Issue 1: Python Version Conflicts

Error: `pytz` version compatibility issues **Solution:** Upgraded local packages and regenerated `requirements.txt`

Issue 2: Windows-Specific Packages

Errors:

- `pywin32==306` not found
- `pywinpty` build failures

Solution: Removed Windows-specific packages:

- `pywin32`
- `pywinpty`

Issue 3: Compilation Errors

Errors:

- `cffi` build failure
- `lxml` compilation issues

- `twisted-iocpsupport` errors

Solution: Installed build dependencies:

sudo apt-get update
sudo apt-get install build-essential libffi-dev libxml2-dev libxslt1-dev

3. requirements.txt Corruption

Issue: File appeared corrupted with garbled characters **Solution:**

1. Removed corrupted file: `rm requirements.txt`
2. Manually created clean version with essential packages only
3. Excluded non-essential packages (Scrapy, jupyter, pandas, etc.)

Result: ✅ Dependencies successfully installed

Phase VI: MySQL Server Installation on EC2

1. Connection Error

Error: `(2002, "Can't connect to local MySQL server through socket '/var/run/mysqld/mysqld.sock' (2)")` **Diagnosis:** MySQL server not installed on EC2

2. MySQL Server Installation

sudo apt-get update
sudo apt-get install mysql-server

3. Service Verification

sudo systemctl status mysql

Result: ✅ MySQL server running successfully

Phase VII: Database User Configuration

1. Database Creation

CREATE DATABASE django_crud_db;

2. User Creation Challenge

Initial Attempt:

CREATE USER 'django_user'@'localhost' IDENTIFIED BY 'django_password!';

Error: `bash: !': event not found` (due to `!` in password)

Solution: Modified password to `django_password12`:

CREATE USER 'django_user'@'localhost' IDENTIFIED BY 'django_password12';

3. Privilege Assignment

GRANT ALL PRIVILEGES ON django_crud_db.* TO 'django_user'@'localhost';
FLUSH PRIVILEGES;

Result: ✅ Database user configured with proper permissions

Phase VIII: Final Configuration & Launch

1. Settings Update

Action: Modified `aws/settings.py` on EC2:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'django_crud_db',
        'USER': 'django_user',
        'PASSWORD': 'django_password12',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

2. Database Migration

python manage.py makemigrations
python manage.py migrate

Result: ✅ Database schema applied successfully

3. Administrative Setup

python manage.py createsuperuser

Result: ✅ Admin user created

4. Static Files Collection

python manage.py collectstatic --noinput

Result: ✅ Static files prepared for serving

5. Server Launch

python manage.py runserver 0.0.0.0:8000

Result: ✅ Django application live on port 8000

Phase IX: Application Access & Troubleshooting

1. Access Method

URL Format: `http://YOUR_EC2_IP:8000` **Example:** `http://13.235.83.88:8000`

2. Connection Timeout Issue

Problem: "This site can't be reached" error with `ERR_CONNECTION_TIMED_OUT`

Root Cause: Port 8000 not open in EC2 Security Group

Solution Steps:

1. Navigate to AWS EC2 Console
2. Go to Network & Security → Security Groups
3. Select your instance's security group
4. Edit Inbound Rules
5. Add new rule:
    - **Type:** Custom TCP
    - **Port Range:** 8000
    - **Source:** 0.0.0.0/0 (for testing)
6. Save changes

Result: ✅ Application accessible via browser

Important Notes

⚠️ EC2 Instance Lifecycle

Critical Understanding:

- **Stopping Instance:** Application goes offline immediately
- **IP Address:** Changes when instance is stopped/started (unless using Elastic IP)
- **Data Persistence:** Code and database files remain intact
- **Billing:** Compute charges stop, storage charges continue

🔒 Security Considerations

- Port 8000 opened to `0.0.0.0/0` for testing only
- For production, restrict source IP ranges
- Consider using Elastic Load Balancer and proper web server (Nginx/Apache)
- Implement HTTPS in production environment

🚀 Production Recommendations

- Use Gunicorn or uWSGI instead of Django development server
- Configure proper web server (Nginx/Apache)
- Set up SSL certificates
- Implement proper logging and monitoring
- Use RDS for database instead of local MySQL
- Configure automated backups

Quick Reference Commands

Local Development

```
# Create virtual environment
python -m venv venv
source venv/bin/activate  # Linux/Mac
venv\Scripts\activate     # Windows

# Install dependencies
pip install -r requirements.txt

# Database operations
python manage.py makemigrations
python manage.py migrate

# Run development server
python manage.py runserver
```

EC2 Deployment

```
# Connect to EC2
ssh -i your-key.pem ubuntu@your-ec2-ip

# Start Django server
python manage.py runserver 0.0.0.0:8000

# Check MySQL status
sudo systemctl status mysql
```

Useful Git Commands

```
git add .
git commit -m "Your commit message"
git push origin main
```

This guide serves as a complete reference for your Django CRUD application deployment journey. Keep it handy for future deployments and troubleshooting!