Capstone project 2025 - Move code and host server

> I have my current code in a state that i want it now i need to move it from my personal device to my work computer and host a server on my work computer and then run my program on it can you walk me through your suggestions and steps to complete this process so i can have my program start being tested by actual users.

Got it. Here's a clean, practical way to move your app to the work PC, host it there, and let real users start testing. I'll assume your capstone is a Django app with SQLite (fits what you've told me: Django + QR + SNMP). If any detail differs, follow the same pattern but swap in your stack.

0) Pre-flight checklist on the work PC

1. You have local admin rights.
2. You have, or can request, either a stable hostname or a reserved IP.
3. You can open a port in Windows Defender Firewall.
4. Optional but nice: a short internal DNS name, like `printapp.berea.local`.

1) Move the code from personal device to work PC

Pick one, use Git if you can.
• Git (recommended):
a) Create a private repo (GitHub, GitLab, internal).
b) On work PC: `git clone <repo-url>`
• Zip: zip your project folder, copy via OneDrive or USB, unzip to `C:\Apps\capstone`.
Either way, you want a folder like `C:\Apps\capstone` with `manage.py`, the app packages, and a `requirements.txt`.

2) Install prerequisites on the work PC (Windows)

1. Install Python 3.11 or 3.12 from python.org, add to PATH.
2. Install Git (if you used Git).
3. In PowerShell (Admin):

```mathematica
cd C:\Apps\capstone
python -m venv .venv
.\.venv\Scripts\Activate.ps1
python -m pip install --upgrade pip
pip install -r requirements.txt
```

3) Configure environment for Django

Create `.env` in your project root (or use your settings mechanism). Example:

```
ini

DJANGO_SECRET_KEY=change-me
DJANGO_DEBUG=False
DJANGO_ALLOWED_HOSTS=workpcname,10.40.140.76,printapp.berea.local
# If using SMTP or other creds, put them here
```

Update `settings.py` to read from `.env` if you aren't already. Add your work PC's hostname or IP to `ALLOWED_HOSTS`.

4) Prepare the database and static files

Inside the venv:

```
css

python manage.py migrate
python manage.py collectstatic --noinput
python manage.py createsuperuser
```

If you're bringing an existing SQLite DB from your personal machine, copy it into a dedicated data folder, for example `C:\Apps\capstone\data\db.sqlite3`, and point your `DATABASES` setting there. Back it up first.

5) Quick smoke test on the LAN (simple, same machine only)

From the venv:

```
nginx

python manage.py runserver 0.0.0.0:8000
```

Open the firewall for that port so other machines on campus can hit it:

```
pgsql

netsh advfirewall firewall add rule name="Capstone Test 8000" dir=in action=allow protocol=TCP localport=8000
```

From another PC, browse to `http://<workpcname>:8000/`. If you can log in and click around, great. This server is for testing only, not production-grade.

6) Pilot-ready server on Windows (stable, no containers)

Use Waitress as the WSGI server and optionally put Caddy or Nginx in front later.

1. Install Waitress:

```nginx
pip install waitress
```

2. Create **`run_waitress.py`** in your project root:

```python
import os
from waitress import serve

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "yourproject.settings")  # change this
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()

if __name__ == "__main__":
    serve(application, host="0.0.0.0", port=8000, threads=8)
```

3. Test it:

```nginx
python run_waitress.py
```

Keep the firewall rule on port 8000. Users can now test at **`http://<workpcname>:8000/`**.

Optional reverse proxy and cleaner URL:
- Install Caddy for Windows, create a **`Caddyfile`**:

```css
printapp.berea.local:80 {
    reverse_proxy localhost:8000
}
```

Run Caddy as a service, and then have users browse **`http://printapp.berea.local/`**. If you only have IP, you can still proxy by IP.

7) Run it as a Windows Service (so it survives logouts and reboots)

Use NSSM (Non-Sucking Service Manager), which is simple and reliable on Windows.

1. Download **`nssm.exe`** and place it in **`C:\Apps\nssm\`**.
2. In PowerShell (Admin):

```makefile
C:\Apps\nssm\nssm.exe install CapstoneApp
```

In the dialog:
- Path: **`C:\Apps\capstone\.venv\Scripts\python.exe`**
- Arguments: **`C:\Apps\capstone\run_waitress.py`**
- Startup directory: **`C:\Apps\capstone`**

• Set “Startup type” to Automatic.
• In the “I/O” tab, set Log files, for example:

- stdout: `C:\Apps\capstone\logs\stdout.log`
- stderr: `C:\Apps\capstone\logs\stderr.log`
  Click Install service, then:

```
sql

sc start CapstoneApp
```

Check: `http://<workpcname>:8000/` works after you log off and after a reboot.

8) Data, backups, and logs

• SQLite file: keep it in `C:\Apps\capstone\data\`.
• Nightly backup: use Task Scheduler to copy it to a secure share:

- Action: `powershell.exe`
- Args: `Copy-Item C:\Apps\capstone\data\db.sqlite3 \\fileserver\backups\capstone\db-$(Get-Date -f yyyyMMdd).sqlite3`
  • Logs: rotate weekly. Easiest is a scheduled task that zips and truncates `stdout.log` and `stderr.log`.

9) Email, file uploads, and static

• Confirm `STATIC_ROOT` is served by Django through Waitress for now. For greater performance later, let Caddy or Nginx serve `STATIC_ROOT` directly.
• If you have user uploads, set `MEDIA_ROOT` to a persistent path and test permissions.

10) User testing plan (quick and realistic)

1. Create three to five pilot users across different buildings.
2. Seed test data that matches real printers and tickets.
3. Capture feedback in a shared doc.
4. Monitor `stderr.log` for tracebacks and `stdout.log` for slow requests.
5. If you’re doing SNMP polling, run it on a schedule or as a Django management command via Task Scheduler, not on request threads.

11) Basic rollback plan

• Keep your old ZIP of the working version.
• If a deploy breaks, stop the service:

```
arduino

sc stop CapstoneApp
```

- Replace the code with the last known good ZIP, reactivate the venv, and restart:

```sql
sc start CapstoneApp
```

12) Deploy updates cleanly

When you're ready to ship changes:

```pgsql
cd C:\Apps\capstone
git pull            (or replace from ZIP)
.\.venv\Scripts\Activate.ps1
pip install -r requirements.txt
python manage.py migrate
python manage.py collectstatic --noinput
sc stop CapstoneApp
sc start CapstoneApp
```

13) Security and housekeeping (campus-friendly)

- Set `DEBUG=False` in production.
- Keep `ALLOWED_HOSTS` tight.
- Store secrets only in `.env` and restrict NTFS permissions on that file.
- Open only the ports you need.
- If you later expose this beyond campus, put Caddy in front with HTTPS and rate limiting.

If you want, I can generate the exact `settings.py` env loader snippet, a sample `.env`, the `run_waitress.py`, a minimal `Caddyfile`, and an NSSM one-pager you can hand to IT.

> I dont think i have permissions on my schools network to have a stable/ reserved IP or open a port in the firewall should i just pay to host it somewhere for the time being

Short answer: yes, use external hosting for now. Since you can't reserve an IP or open inbound ports on campus, the cleanest path is to run your app off-campus and point testers to a public URL. Here are three workable options, in order of "least red tape".

Option A: Quick cloud VPS (reliable, cheap, fully under your control)

Good for a pilot with real users; costs about $6–$12/month.

1. Create a small Ubuntu VM (DigitalOcean, Lightsail, Linode; 1 vCPU, 1–2 GB RAM).
2. Point a subdomain to it (optional but nicer): `printapp.yourdomain.com`.
3. On the VM:

```
bash

sudo apt update && sudo apt -y install python3-venv python3-pip git nginx
sudo ufw allow 'OpenSSH'
sudo ufw allow 'Nginx Full'
sudo ufw enable
git clone <your repo> /srv/capstone && cd /srv/capstone
python3 -m venv .venv && source .venv/bin/activate
pip install --upgrade pip && pip install -r requirements.txt gunicorn
python manage.py collectstatic --noinput
python manage.py migrate
```

4. Gunicorn service:

```
pgsql

sudo tee /etc/systemd/system/capstone.service >/dev/null <<'EOF'
[Unit]
Description=Capstone Django
After=network.target
[Service]
User=www-data
WorkingDirectory=/srv/capstone
Environment="DJANGO_SETTINGS_MODULE=yourproject.settings"
EnvironmentFile=/srv/capstone/.env
ExecStart=/srv/capstone/.venv/bin/gunicorn yourproject.wsgi:application --bind 127.0.0.1:8000 --workers 3
Restart=always
[Install]
WantedBy=multi-user.target
EOF
sudo systemctl daemon-reload
sudo systemctl enable --now capstone
```

5. Nginx reverse proxy with HTTPS:

```
nginx

sudo tee /etc/nginx/sites-available/capstone >/dev/null <<'EOF'
server {
  listen 80;
  server_name printapp.yourdomain.com;
  location / { proxy_pass http://127.0.0.1:8000; proxy_set_header Host $host; proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for; proxy_set_header X-Forwarded-Proto $scheme; }
  client_max_body_size 20m;
}
EOF
sudo ln -s /etc/nginx/sites-available/capstone /etc/nginx/sites-enabled/
sudo nginx -t && sudo systemctl reload nginx
```

Then add HTTPS with Certbot later. Share the URL with testers.

Pros: works anywhere, no campus changes needed, stable.
Cons: you pay a few dollars monthly and you manage the box.

Option B: Cloud “PaaS” (simplest deployment, slightly pricier)

Use Render, Railway, Fly.io, or Azure App Service. You push your repo, set env vars, they build and run it, give you HTTPS. Perfect if you don’t want to touch servers.

Pros: fastest to live; logs, deploys, HTTPS handled.
Cons: free tiers can sleep or throttle; steady usage usually ~$7–$20/month.

Option C: Zero-inbound “tunnel” from your work PC (free, if allowed)

If outbound traffic is allowed but inbound is blocked, run a tunnel client that creates a public URL to your local server.

Two common approaches:

1. Cloudflare Tunnel: installs a small service on Windows, maps `yourname.trycloudflare.com` (or your domain) to `http://localhost:8000`.
2. Tailscale/ZeroTier: creates a VPN overlay; only people in your tailnet can access the app.

Pros: no firewall changes, often free.
Cons: may require installing a service on the work PC, and IT policies may forbid it; Tailscale requires testers to install a client.

What I recommend for you

Given your constraints and that you want real users testing soon: go with Option A (small VPS) or Option B (Render/Railway). They’re predictable and won’t fight campus networking rules.

If you want, tell me which route you pick:
• VPS: I’ll give you a ready-to-paste `.env` template, systemd unit filled with your project name, and an Nginx config tuned for Django static/media.
• Render/Railway: I’ll give you the exact build/run commands and env vars, plus a minimal `gunicorn` start command and `STATIC_ROOT` handling.

> out of the options you said for option A to create a small Ubuntu VM can you tell me which one you sugest

I’d go with DigitalOcean for your pilot. It’s the least “fussy,” has tons of copy-pasteable guides, and a $4–$6/month droplet is plenty for a small Django app. Their entry plan includes a pool of outbound transfer and inbound is free, so you won’t get nickeled on bandwidth during testing. DigitalOcean+2

When I’d pick the others
• AWS Lightsail: choose this only if you want an easy on-ramp to the AWS ecosystem or want to use the 90-day trial on the $5 Linux instance tier. Pricing is flat and includes 2 TB transfer at the $5 level. Amazon Web Services, Inc.+1
• Linode/Akamai: comparable price to DO ($5 “Nanode”), with 1 TB transfer and very low egress overage. Good if you care about bandwidth costs. Akamai+1

If you want, I’ll give you a DO-specific, paste-ready runbook: create droplet, harden SSH, install Python+Nginx+Gunicorn, systemd unit, Nginx site, Let’s Encrypt, and a minimal `.env` for Django.

okay i went with digital ocean but i forgot to change my allowed hosts how do i find what that allowed host should be set to and update my site

Here’s the fast way to fix it.

1) Identify what host users are hitting

Pick one and copy the exact value you use in the browser’s address bar.
• If you’re using a domain: **`printapp.yourdomain.com`**
• If you’re using the droplet IP for now: get it with:

```
nginx

curl -4 ifconfig.co
```

2) Update Django settings

If you load env vars from **`.env`**, do this.

Open your **`.env`** on the server:

```
bash

sudo nano /srv/capstone/.env
```

Set both of these, comma separated if you have more than one host:

```
ini

DJANGO_ALLOWED_HOSTS=printapp.yourdomain.com,123.45.67.89
CSRF_TRUSTED_ORIGINS=https://printapp.yourdomain.com,http://123.45.67.89
```

Notes:
• Keep the scheme in **`CSRF_TRUSTED_ORIGINS`** and include https if you’re using TLS.
• If you only use the IP right now, put only the IP. You can add the domain later.

Make sure your **`settings.py`** reads these:

```
python

import os
ALLOWED_HOSTS = [h.strip() for h in os.getenv("DJANGO_ALLOWED_HOSTS","").split(",") if
h.strip()]
CSRF_TRUSTED_ORIGINS = [o.strip() for o in os.getenv("CSRF_TRUSTED_ORIGINS","").split(",") if
o.strip()]
```

If you are not using a **`.env`**, edit **`settings.py`** directly and set:

```
python

ALLOWED_HOSTS = ["printapp.yourdomain.com", "123.45.67.89"]
```

```
CSRF_TRUSTED_ORIGINS = ["https://printapp.yourdomain.com", "http://123.45.67.89"]
```

3) Restart the app

If you used my **`capstone.service`**:

```
lua

sudo systemctl restart capstone
sudo systemctl status capstone --no-pager
```

If you changed your Nginx server_name, also:

```
nginx

sudo nginx -t && sudo systemctl reload nginx
```

4) Verify

• Hit the site from your laptop using the same host you added, for example:

- **`https://printapp.yourdomain.com`**
- **`http://123.45.67.89`**
  • If you still see “DisallowedHost”, check logs:

```
nginx

journalctl -u capstone -e
```

The log will show which Host header Django rejected. Add that exact value to **`DJANGO_ALLOWED_HOSTS`**, restart, and retry.

5) When you switch from IP to domain later

• Add the domain to **`DJANGO_ALLOWED_HOSTS`** and **`CSRF_TRUSTED_ORIGINS`**.
• Keep the old IP in the list for a day or two if users still bookmark it.
• After you enable HTTPS, remove any **`http://`** origin you no longer use.

Optional sanity check:

```
bash

source /srv/capstone/.venv/bin/activate
python manage.py check --deploy
```