What will we do now?

- **Create the full folder structure** for backend, frontend, and logs.
- **Set permissions** so the right users can run the app and edit code.
- **Add placeholder files** for backend (app.py, .env) and frontend (HTML, JS, media).
- **Install Python and set up a virtual environment** with all required dependencies (Flask, Stripe, Supabase, Gunicorn).
- **Install and configure NGINX** as our web server to serve static files and proxy API requests to the backend.
- **Set up a temporary HTTP-only config** so Let's Encrypt can issue our SSL certificate.
- **Create a minimal index page** for certificate validation.
- **Enable our NGINX site** and disable the default config.
- **Test and reload NGINX** to ensure the config works.
- **Open firewall ports** for web traffic (HTTP & HTTPS).
- **Install Certbot** to handle SSL certificates.
- **Request the SSL certificate** using the temporary HTTP config.
- **Replace temporary NGINX config with the final HTTPS config** for secure connections.
- **Test and reload NGINX again** to apply the secure setup.
- **Replace placeholder files with the real app code** (backend + frontend).
- **Create a systemd service** to run Flask via Gunicorn, ensuring it starts on boot and restarts if it crashes.
- **Apply backend permissions** so the service runs under the correct user and file security is in place.
- **Start and enable the systemd service** so the app is live 24/7.

This is the folder structure we need to create on our cloud server:

```
/opt/gptsweetheart/                 # Flask app (backend)
├── app.py                          # main backend entry
├── .env                            # environment variables
├── venv/                           # Python virtual env
├── logs/
│   └── gunicorn.log                # backend logs

/var/www/gptsweetheart/             # Static site (frontend)
├── sign-up.html                    # signup page
├── login.html                      # login page
├── redirect.html                   # email redirect/activation page
├── dashboard.html                  # user dashboard
├── assets/                         # css/js/images for frontend
│   └── js/                         # frontend JavaScript files
│       ├── auth.js
│       ├── sign-up.js
│       ├── login.js
│       ├── dashboard.js
│       └── redirect.js
└── media/                          # generated images served at /media/*
```

For that we use the following bashes:

# 1) Main project folder (backend root)
sudo mkdir -p /opt/gptsweetheart
sudo chown "$USER:$USER" /opt/gptsweetheart

# 2) Backend folders + dummy files
sudo mkdir -p /opt/gptsweetheart/logs
sudo touch /opt/gptsweetheart/app.py \
/opt/gptsweetheart/.env \
/opt/gptsweetheart/logs/gunicorn.log
sudo chown -R "$USER:$USER" /opt/gptsweetheart

# 3) Frontend folders + dummy HTML & JS files
sudo mkdir -p /var/www/gptsweetheart/assets/js /var/www/gptsweetheart/media

# 4) HTML placeholders
sudo touch /var/www/gptsweetheart/sign-up.html \
/var/www/gptsweetheart/login.html \
/var/www/gptsweetheart/redirect.html \
/var/www/gptsweetheart/dashboard.html

# 5) JS placeholders
```
sudo touch /var/www/gptsweetheart/assets/js/auth.js \
        /var/www/gptsweetheart/assets/js/sign-up.js \
        /var/www/gptsweetheart/assets/js/login.js \
        /var/www/gptsweetheart/assets/js/dashboard.js \
        /var/www/gptsweetheart/assets/js/redirect.js
```

# 6) Web server owns everything for serving
```
sudo chown -R www-data:www-data /var/www/gptsweetheart
sudo find /var/www/gptsweetheart -type d -exec chmod 755 {} \;
sudo find /var/www/gptsweetheart -type f -exec chmod 644 {} \;
```

# 7) Give YOU edit rights to HTML & JS files
```
sudo chown "$USER:$USER" /var/www/gptsweetheart/*.html
sudo chown "$USER:$USER" /var/www/gptsweetheart/assets/js/*.js
```

# 8) Create the .env file in the right folder
```
cd /opt/gptsweetheart && nano .env
```

# 9) Paste that with YOUR values and then safe and close with Control + O, Enter, Control + X
```
SUPABASE_URL=https://your-supabase-project-url.supabase.co
SUPABASE_SERVICE_ROLE_KEY=your-supabase-service-role-key

STRIPE_SECRET_KEY=sk_test_your_stripe_secret_key
STRIPE_PRICE_ID_PRO=price_your_stripe_price_id
STRIPE_WEBHOOK_SECRET=whsec_your_stripe_webhook_secret

FRONTEND_URL=https://gptsweetheart.com
```

# 10) Navigate to the right folder
```
cd /opt/gptsweetheart
```

# 11) Python + venv install
```
sudo apt-get update
sudo apt-get install -y python3-venv python3-pip
```

# 12) Create venv + install deps
```
python3 -m venv venv
source venv/bin/activate
pip install --upgrade pip
pip install flask flask-cors supabase python-dotenv stripe gunicorn
```

# 13) Install Nginx
```
sudo apt update
sudo apt install -y nginx
```

# 14) TEMP HTTP-only server block for Certbot

```
sudo tee /etc/nginx/sites-available/gptsweetheart >/dev/null <<'EOF'
server {
    listen 80;
    listen [::]:80;
    server_name gptsweetheart.com www.gptsweetheart.com;

    root /var/www/gptsweetheart;
    index index.html;

    location /.well-known/acme-challenge/ {
        alias /var/www/gptsweetheart/.well-known/acme-challenge/;
    }
}
EOF
```

# 15) Create webroot + minimal index

```
sudo mkdir -p /var/www/gptsweetheart
echo "OK" | sudo tee /var/www/gptsweetheart/index.html >/dev/null
```

# 16) Enable site (idempotent)

```
sudo ln -sf /etc/nginx/sites-available/gptsweetheart /etc/nginx/sites-enabled/gptsweetheart
```

# 17) Disable default site if present

```
sudo rm -f /etc/nginx/sites-enabled/default
```

# 18) Test & reload

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

# 19) Firewall (ok if already allowed)

```
sudo ufw allow 'Nginx Full'
```

# 20) Install Certbot

```
sudo apt-get update
sudo apt-get install -y certbot python3-certbot-nginx
```

# 21) Issue certs (webroot avoids plugin touching your config)

```
sudo certbot certonly --webroot \
  -w /var/www/gptsweetheart \
  -d gptsweetheart.com -d www.gptsweetheart.com
```

# 22) Replace server block with production HTTPS config

```
sudo tee /etc/nginx/sites-available/gptsweetheart >/dev/null <<'EOF'
server {
    listen 80;
    listen [::]:80;
    server_name gptsweetheart.com www.gptsweetheart.com;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name gptsweetheart.com www.gptsweetheart.com;

    ssl_certificate     /etc/letsencrypt/live/gptsweetheart.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/gptsweetheart.com/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;

    root /var/www/gptsweetheart;
    index index.html;

    location / {
        try_files $uri $uri.html =404;
    }

    location /api/ {
        proxy_pass http://127.0.0.1:5002;
        proxy_http_version 1.1;
        proxy_set_header Host              $host;
        proxy_set_header X-Real-IP         $remote_addr;
        proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Authorization     $http_authorization;
        proxy_read_timeout 180s;
    }

    location /media/ {
        alias /var/www/gptsweetheart/media/;
        add_header Cache-Control "public, max-age=31536000, immutable";
    }
}
EOF
```

# 23) Test & reload

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

# 24) Replace the dummy files
Replace the dummy files on the server with the real files (app.py and the html pages)

# 25) Create the systems unit (already updated, just paste it)

```
sudo tee /etc/systemd/system/gptsweetheart.service >/dev/null <<'EOF'
[Unit]
Description=GPTSweetheart Flask (gunicorn)
After=network.target

[Service]
User=www-data
Group=www-data
WorkingDirectory=/opt/gptsweetheart
Environment="PATH=/opt/gptsweetheart/venv/bin"
ExecStart=/opt/gptsweetheart/venv/bin/gunicorn -w 3 -b 127.0.0.1:5002 app:app --timeout 120
Restart=always
RestartSec=3

[Install]
WantedBy=multi-user.target
EOF
```

# 26) Permission and start (already updated, just paste it)

```
sudo chown -R www-data:www-data /opt/gptsweetheart
sudo find /opt/gptsweetheart -type d -exec chmod 750 {} \;
sudo chmod 640 /opt/gptsweetheart/*.py
```

# 27) Start service

```
sudo systemctl daemon-reload
sudo systemctl enable --now gptsweetheart
sudo systemctl status gptsweetheart --no-pager
```

Here you find an explanation to each step we did

1. **Create backend root**
   We make a dedicated folder for the Flask app at /opt/gptsweetheart. Giving ownership to your user lets you create and edit files there without using sudo every time.

2. **Scaffold backend files**
   We create the basic pieces the backend expects: an app.py, a .env for secrets, and a logs/ directory for Gunicorn logs. Owning the whole tree ensures your editor and git commands can write to it.

3. **Create frontend directories**
   The public website lives in /var/www/gptsweetheart. We prepare assets/js for your JavaScript and media for generated files so Nginx can serve them directly.

4. **Add HTML placeholders**
   We create empty HTML pages the course will later replace. This guarantees that paths like /sign-up or /dashboard already exist and can be served by Nginx while you're wiring things up.

5. **Add JS placeholders**
   Same idea for the JavaScript files each page imports (including auth.js). Placeholders prevent 404s in the browser until you paste the real code.

6. **Give the web server ownership + sane perms**
   We set www-data (Nginx's user) as the owner so it can read everything it serves. Directories get 755 and files get 644—classic static-site permissions that avoid permission errors.

7. **Give yourself edit rights to source files**
   We switch only the HTML/JS files back to your user so you can edit them comfortably. The web server still owns the tree, but you won't fight permissions while iterating.

8. **Open the env file**
   You'll store secrets and configuration (Supabase, Stripe, frontend URL) in .env. Keeping secrets here keeps them out of your codebase and makes local edits simple.

9. **Paste your real credentials**
   Supabase and Stripe keys, plus FRONTEND_URL, are what your app needs to talk to external services and build correct redirect links. Saving them now makes the rest of the setup deterministic.

10. **Move to the backend folder**
    We cd into /opt/gptsweetheart so the following Python/venv commands run in the right place. It's a small step that avoids path mistakes.

11. **Install Python tooling**
We install the Python virtual environment tools and pip, which let us isolate dependencies for this app (no system-wide pollution). This keeps your server clean and reproducible.

12. **Create venv and install deps**
The virtual environment holds Flask, Supabase client, Stripe, CORS, and Gunicorn. Installing now ensures the service will have everything it needs when we start it.

13. **Install Nginx**
Nginx will serve the static frontend and reverse-proxy /api/* to your Flask app. It's fast, battle-tested, and lets us keep the backend private on 127.0.0.1.

14. **Temporary HTTP-only Nginx (for Certbot)**
We write a **minimal port-80** config so Let's Encrypt can verify your domain using the webroot method. This is intentionally simple (no TLS yet) because Certbot first needs to reach http:// to prove you own the domain. You'll **replace this later in Step 22** with the full HTTPS setup after certificates exist.

15. **Ensure the webroot and a basic index**
We create /var/www/gptsweetheart/index.html with "OK" so you can confirm the site serves over HTTP. This also gives Certbot a writable webroot to drop its challenge files.

16. **Enable the site**
We symlink the config into sites-enabled so Nginx will actually load it. This mirrors how Ubuntu/Debian keep Nginx vhosts organized.

17. **Disable the default site**
Turning off the default prevents unexpected content from being served at your domain. It also removes ambiguity if the default config overlaps with your hostnames.

18. **Test and reload Nginx**
nginx -t catches typos before a reload, and reloading applies the new HTTP config without downtime. After this, http://yourdomain should show "OK".

19. **Allow Nginx through the firewall**
We open UFW for "Nginx Full" (ports 80/443). If it's already allowed, the command is harmless; if not, it prevents head-scratchers later.

20. **Install Certbot**
Certbot is the Let's Encrypt client that will issue free TLS certificates. We'll use it in "webroot" mode so it doesn't rewrite our config unexpectedly.

21. **Issue the certificates (webroot)**
Certbot places a challenge file in your webroot and validates over plain HTTP. Once it verifies ownership, it writes your certs into /etc/letsencrypt/... so you can enable HTTPS.

22. **Replace with production HTTPS config**
Now that certs exist, we switch to the **final** Nginx config: automatic **HTTP→HTTPS**

redirect, proper TLS (using the certs from Step 21), static file serving, and a **/api** reverse-proxy to Gunicorn (including forwarding the Authorization header and preserving the /api prefix). This is the configuration you'll run in production.

23. **Test and reload Nginx (HTTPS)**
We validate the new HTTPS config and reload Nginx so it starts serving TLS right away. If there's a typo, nginx -t tells you before you break anything.

24. **Replace placeholders with the real app**
Drop in your real app.py, HTML, and JS. At this point the web server and reverse proxy are ready to run the actual code paths.

25. **Create the systemd unit**
We register a gptsweetheart.service that runs Gunicorn with your app and binds it to 127.0.0.1:5002. Systemd will manage restarts and start it on boot.

26. **Set backend permissions for the service user**
The unit runs as www-data, so we give www-data ownership and secure permissions under /opt/gptsweetheart. This avoids "permission denied" when Gunicorn tries to read code or logs.

27. **Start and enable the service**
We reload systemd to pick up the new unit, enable it to start on boot, and start it now. The status command confirms the app is running with three workers and listening on localhost port 5002.

Summary

This setup builds a **production-ready SaaS deployment** from scratch:

- **Backend** lives in /opt/gptsweetheart (Flask + Gunicorn).
- **Frontend** lives in /var/www/gptsweetheart (served directly by NGINX).
- **NGINX** serves static files and proxies API requests to the backend.
- **HTTPS** is enabled via Let's Encrypt, using a two-step config process:
    1. **Step 14:** Temporary HTTP-only config so Certbot can verify the domain.
    2. **Step 22:** Final HTTPS config with redirect and secure proxying.
- **systemd** keeps the app running 24/7 and restarts it automatically if needed.
- Permissions are set so the web server user can run the app, while you still have edit rights to relevant frontend files.

The result is a clean, secure, and maintainable SaaS prototype that can handle real traffic, survive restarts, and serve both static and dynamic content efficiently.