AI Marketer

AI Marketer is a platform that automates social media marketing using artificial intelligence. Upload images and the AI will generate appropriate captions to help you post across multiple social media platforms.

Getting Started

The instructions below are for running AI Marketer locally. For details on deployment or retrieving environment variables (e.g., API keys, secrets), please refer to the materials provided by Daniel.

Frontend Setup Guide

This guide provides instructions for setting up and running the AI Marketer frontend application in your local environment. The project has a monorepo structure, and this section focuses on the setup within the `ai-marketer-v2-frontend` directory.

1. Navigate to the Frontend Directory

First, from the project's root directory, navigate to the frontend directory:

```
cd ai-marketer-v2-frontend
```

2. Check Prerequisites

To run the frontend application, you'll need:

- **Node.js 18+**: LTS version recommended
- **npm or yarn**: For package management
- **mkcert**: For generating local HTTPS certificates

Verify Node.js and npm are installed:

```
node -v
npm -v
```

3. Install Dependencies

Install all required packages from within the frontend directory:

```
npm install
# or
yarn install
```

4. Set Up HTTPS Certificates

AI Marketer uses HTTPS even in the local development environment for cookie-based authentication and secure communication. You need to generate local certificates.

4.1. Install mkcert

macOS (using Homebrew):

```
brew install mkcert
brew install nss  # Additional installation for Firefox support
mkcert -install
```

Windows (using Chocolatey):

```
choco install mkcert
mkcert -install
```

Linux (Ubuntu/Debian):

```
sudo apt install libnss3-tools
# Install mkcert - refer to https://github.com/FiloSottile/mkcert
mkcert -install
```

4.2. Generate Localhost Certificates

From within the frontend directory, run the following command to generate certificates for localhost:

```
mkcert localhost
```

This command creates `localhost.pem` (certificate) and `localhost-key.pem` (private key) files in the current directory. These files are used when running the frontend application with HTTPS.

5. Environment Variable Setup

Create an environment variable setup file:

```
cp .env.local.example .env.local
```

Then edit the `.env.local` file to set the necessary environment variables:

```
# Backend Server URL - Specify the URL where the backend service is running
NEXT_PUBLIC_API_URL=https://localhost:8000

# Image Domains - List of domains where images are hosted (comma-separated)
NEXT_PUBLIC_IMAGE_DOMAINS=localhost
```

> **Important**: If you're running the backend server locally, make sure the port number in `NEXT_PUBLIC_API_URL` matches the port of your backend server. The default is `8000`.

6. Run the Development Server

After completing all the setup, start the development server with:

```
npm run dev
# or
yarn dev
```

This command runs the HTTPS server at `https://localhost:3000`.

You can access the AI Marketer frontend application in your browser at `https://localhost:3000`. When accessing for the first time, your browser may display a warning about the self-signed certificate, which you can safely ignore and proceed.

7. Troubleshooting

7.1. Certificate-Related Issues

If certificate warnings persist:

- Verify mkcert is correctly installed
- Confirm the `mkcert -install` command ran successfully
- Check that the generated certificate files are in the frontend directory

7.2. Backend Connection Issues

If you encounter API connection errors:

- Verify the backend server is running
- Check the `NEXT_PUBLIC_API_URL` value in your `.env.local` file is correct
- Ensure the backend CORS settings allow requests from your frontend URL

7.3. Dependency Issues

If you encounter dependency-related errors:

```
npm clean-install
# or
yarn install --force
```

7.4. Port Conflicts

If port 3000 is already in use, you can specify a different port:

```
npm run dev -- -p 3001
# or
yarn dev -p 3001
```

Frontend Project Structure

The frontend application is built with Next.js and has the following structure:

```
src/
├── app/                # Next.js App Router pages
│   ├── (auth)/         # Authentication routes (login, register, etc.)
│   ├── (protected)/    # Protected routes (dashboard, posts, settings, etc.)
│   └── layout.tsx      # Root layout
├── components/         # Reusable UI components
│   ├── auth/           # Authentication-related components
│   ├── common/         # Generic UI components
│   ├── layout/         # Layout components
│   ├── post/           # Post creation & editing components
│   ├── styles.ts       # Common styles and utilities
│   └── sections/       # Homepage section components
├── constants/          # Application constants
├── context/            # React context providers
├── hooks/              # Custom React hooks
├── types/              # TypeScript type definitions
└── utils/              # Utility functions
```

Backend Setup Guide

This guide provides instructions for setting up and running the AI Marketer backend application in your local environment.

1. Navigate to the Backend Directory

From the project's root directory, navigate to the backend directory:

```
cd ai-marketer-v2-backend/backend
```

2. Check Prerequisites

To run the backend application, you'll need:

- **Python 3.11+**: Required for running the Django application
- **Docker and Docker Compose**: For containerization and easy setup
- **WSL2**: Recommended for Windows users (see detailed WSL2 setup in README.md)
- **mkcert**: For generating local HTTPS certificates

Verify that Python and Docker are installed:

```
python --version
docker --version
docker-compose --version
```

3. Set Up HTTPS Certificates

Like the frontend, the backend also uses HTTPS in the local development environment. You need to generate local certificates using mkcert:

```
mkcert localhost
```

This will create `localhost.pem` (certificate) and `localhost-key.pem` (private key) files in the current directory.

4. Environment Variable Setup

Create an environment variable setup file by copying the example file:

```
cp .env.example .env
```

Then edit the `.env` file to set the necessary environment variables:

```
# Debug mode (set to False in production)
DEBUG=True

# Database settings
POSTGRES_DB=postgres
POSTGRES_USER=postgres
POSTGRES_PASSWORD=password
POSTGRES_HOST=db
POSTGRES_PORT=5432

# Development environment settings
DJANGO_ENV=development
FLUSH_DB=False
USE_RENDER_DB=False

# CORS and allowed hosts
CORS_ALLOWED_ORIGINS=https://localhost:3000
FRONTEND_BASE_URL=https://localhost:3000

# Square API settings
SQUARE_ENV=sandbox
SQUARE_BASE_URL_SANDBOX=https://connect.squareupsandbox.com
SQUARE_BASE_URL_PROD=https://connect.squareup.com
SQUARE_REDIRECT_URI=https://localhost:8000/api/businesses/square/callback/

# Celery broker URL
CELERY_BROKER_URL=redis://redis:6379/0

# For Meta integration
FACEBOOK_REDIRECT_URI=https://localhost:3000/settings/social/

# Required secrets (you need to fill these in)
ALLOWED_HOSTS=localhost,127.0.0.1,0.0.0.0
SQUARE_APP_ID_SANDBOX=your-sandbox-app-id
SQUARE_APP_SECRET_SANDBOX=your-sandbox-app-secret
FACEBOOK_APP_ID=your-facebook-app-id
FACEBOOK_SECRET=your-facebook-app-secret
OPENAI_API_KEY=your-openai-api-key
DISCORD_WEBHOOK_URL=your-discord-webhook-url
```

When you first run the application, the entrypoint script will automatically generate values for `SECRET_KEY` and `TWOFA_ENCRYPTION_KEY` and add them to your `.env` file.

5. Start the Backend Server with Docker Compose

From the root directory of the project, run:

```
docker-compose up
```

This command will:

- Build and start the backend container
- Set up a PostgreSQL database container
- Set up a Redis container for Celery tasks
- Start a Celery worker for background tasks

The backend API will be accessible at `https://localhost:8000`.

6. Troubleshooting

6.1. Database Issues

If you encounter database issues, you can reset the database:

```
# Set FLUSH_DB=True in your .env file
docker-compose down
docker-compose up
```

6.2. Permission Issues with Docker

If you encounter permission issues with Docker, try running the commands with sudo:

```
sudo docker-compose up
```

6.3. Certificate Issues

If you encounter certificate issues:

- Verify that the certificate files are in the correct location
- Check that the filenames match what the application expects (`localhost.pem` and `localhost-key.pem`)

6.4. Environment Variable Issues

If certain features don't work due to missing API keys:

- Make sure all required API keys are set in your `.env` file
- For development, you can use placeholder values for some third-party services if you're not using those features

Backend Project Structure

The backend application is built with Django and has the following structure:

```
backend/
├── ai/                 # AI-related functionality (caption generation)
├── businesses/         # Business model and views
├── config/             # Project configuration and settings
│   ├── celery_app.py   # Celery configuration
│   ├── constants.py    # Project constants
│   └── settings.py     # Django settings
├── posts/              # Social media posts functionality
├── promotions/         # Business promotions functionality
├── sales/              # Sales data analysis
├── social/             # Social media integration
├── users/              # User authentication and management
├── utils/              # Utility functions
│   ├── discord_api.py  # Discord integration for image hosting
│   ├── openai_api.py   # OpenAI integration for AI features
│   └── square_api.py   # Square integration for business data
├── manage.py           # Django management script
├── requirements.txt    # Python dependencies
└── Dockerfile          # Docker configuration
```