Jokefactory Documentation

Overview

The Jokefactory App is designed for users to read, rate, and upload jokes. It includes a highscore board that updates every minute. The application is Dockerized, with multiple Django instances managed by Traefik for load balancing. It uses PostgreSQL for persistent storage and Redis with Celery for background tasks.

Requirements

- **Load Balancing**: Handled by Traefik, allowing multiple instances of the Django service with failover support. We chose Traefik because it seems to work well with Docker and it has an easy to follow documentation.
- **Dockerized**: The application runs in Docker containers for easy deployment and scalability.
- **Simple Frontend**: The app includes basic HTML with Django templates.
- **Scheduled Tasks**: Celery is used for scheduling and handling background tasks.
- **Persistent Storage**: PostgreSQL is used for storing jokes and related data.
- **Latest Stable Releases**: Uses stable versions of Django, Celery, and other dependencies as specified in `requirements.txt`.

Project Structure

```
jokefactory_project
├── jokefactory
│   ├── models.py
│   ├── tasks.py
│   ├── urls.py
│   └── views.py
├── jokefactory_project
│   ├── celery.py
│   ├── settings.py
│   └── urls.py
├── docker-compose.yml
├── Dockerfile
├── manage.py
├── requirements.txt
└── README.md
```

Docker Configuration

Dockerfile: The `Dockerfile` defines the environment for the Django application, installing necessary dependencies and setting up the application.

```
FROM python:3.9-slim

ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

WORKDIR /code

COPY requirements.txt /code/

RUN pip install -r requirements.txt

COPY . /code/

EXPOSE 8000

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

docker-compose.yml: The `docker-compose.yml` file orchestrates multiple services including Django, PostgreSQL, Redis, and Traefik.

```
services:
  reverse-proxy:
    image: traefik:v2.11
    command:
      - --api.insecure=true
      - --providers.docker
      - --entrypoints.web.address=:80
    ports:
      - "80:80"
      - "8080:8080"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    labels:
      - "traefik.http.routers.api.rule=Host(`localhost`)"
      - "traefik.http.routers.api.service=api@internal"
      - "traefik.http.routers.api.entrypoints=web"
      - "traefik.http.services.api.loadbalancer.server.port=8080"

  whoami:
    image: traefik/whoami
    labels:
      - "traefik.http.routers.whoami.rule=Host(`whoami.docker.localhost`)"

  database:
```

```
    image: 'postgres:16.2'
    restart: always
    container_name: database
    ports:
      - 5432:5432
    environment:
      POSTGRES_USER: username
      POSTGRES_PASSWORD: password
      POSTGRES_DB: jokesDB
    volumes:
      - db_data:/var/lib/postgresql/data

  redis:
    image: redis:latest
    ports:
      - 6379:6379

  web:
    image: django-app:latest
    command: >
      sh -c "python manage.py migrate && python manage.py runserver 0.0.0.0:8000"
    volumes:
      - .:/code
    ports:
      - "8000:8000"
    environment:
      - CELERY_BROKER_URL=redis://redis:6379/0
      - CELERY_RESULT_BACKEND=redis://redis:6379/0
    labels:
      - "traefik.http.routers.web.rule=Host(`localhost`)"
      - "traefik.http.services.web.loadbalancer.server.port=8000"
    depends_on:
      - database
      - redis
    deploy:
      replicas: 3
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure

  celery:
    image: django-app:latest
    command: celery -A jokefactory_project worker --loglevel=info
    volumes:
      - .:/code
    environment:
      - CELERY_BROKER_URL=redis://redis:6379/0
      - CELERY_RESULT_BACKEND=redis://redis:6379/0
    depends_on:
      - web
      - redis
    deploy:
```

```
      replicas: 1

  celery-beat:
    image: django-app:latest
    command: celery -A jokefactory_project beat --loglevel=info
    volumes:
      - .:/code
    environment:
      - CELERY_BROKER_URL=redis://redis:6379/0
      - CELERY_RESULT_BACKEND=redis://redis:6379/0
    depends_on:
      - web
      - redis
    deploy:
      replicas: 1

volumes:
  db_data:
```

Traefik Configuration

Traefik handles routing and load balancing between multiple instances of the Django service. The relevant configuration is included in the `docker-compose.yml` file, specifying entrypoints, providers, and routing rules.

Django Application

settings.py: The Django settings include configuration for databases, static files, and installed applications.

```
import os

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

SECRET_KEY = 'secret-key'
DEBUG = True

ALLOWED_HOSTS = []

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'jokefactory',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
```

```
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'myproject.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'myproject.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'jokesDB',
        'USER': 'user',
        'PASSWORD': 'password',
        'HOST': 'db',
        'PORT': 5432,
    }
}

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

```
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')

CELERY_BROKER_URL = 'redis://redis:6379/0'
CELERY_RESULT_BACKEND = 'redis://redis:6379/0'
```

views.py: Defines the logic for handling HTTP requests in the Django application.

```
import random
from django.http import JsonResponse
from django.shortcuts import get_object_or_404, render, redirect
from .forms import JokeForm
from .models import Joke, Ranking

def joke_list(request):
    jokes = Joke.objects.all()
    return render(request, 'joke_list.html', {'jokes': jokes})

def joke_ranking(request):
    ranking = Ranking.objects.all().order_by('rank')[:10]
    return render(request, 'joke_ranking.html', {'ranking': ranking})

def joke_create(request):
    if request.method == 'POST':
        form = JokeForm(request.POST)
        if form.is_valid():
            joke = form.save(commit=False)
            joke.save()
            return redirect('joke_list')
    else:
        form = JokeForm()
    return render(request, 'joke_form.html', {'form': form})

def joke_list(request):
    jokes = Joke.objects.all().order_by("-rating")[:20]
    joke = get_rand_joke()

    return render(request, 'joke_list.html', {'jokes': jokes,'joke': joke})

def rate_minus(request, joke_id):
    jokes = Joke.objects.all().order_by("-rating")[:20]
    old_joke = get_object_or_404(Joke, id=joke_id)
    old_joke.minus_rating += 1
    old_joke.save()
    joke = get_rand_joke()
```

```
    return render(request, 'joke_list.html', {'jokes': jokes,'joke': joke})

def rate_plus(request, joke_id):
    jokes = Joke.objects.all().order_by("-rating")[:20]
    old_joke = get_object_or_404(Joke, id=joke_id)
    old_joke.rating += 1
    old_joke.save()
    joke = get_rand_joke()

    return render(request, 'joke_list.html', {'jokes': jokes,'joke': joke})

def rate_twoplus(request, joke_id):
    jokes = Joke.objects.all().order_by("-rating")[:20]
    old_joke = get_object_or_404(Joke, id=joke_id)
    old_joke.rating += 2
    old_joke.save()
    joke = get_rand_joke()

    return render(request, 'joke_list.html', {'jokes': jokes,'joke': joke})

def get_rand_joke():
    jokes = Joke.objects.all()
    jokes_count = jokes.count()

    if jokes_count == 0:
        return None
    else:
        joke = jokes[random.randint(0, jokes_count-1)]
        return joke

def rate_joke(joke, rate):
    joke.rating += rate
    joke.save()
```

models.py: Defines the data model for jokes. Each joke has a title, content, author, rating, minus rating and a created-at date.

```
from django.db import models

class Joke(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    author = models.CharField(max_length=100)
    rating = models.IntegerField(default=0)
    minus_rating = models.IntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```

```
class Ranking(models.Model):
    joke = models.ForeignKey(Joke, on_delete=models.SET_NULL, null = True)
    rank = models.IntegerField()

    def __str__(self):
        return self.joke.title if self.joke else "Deleted Joke"
```

Scheduled Tasks

Scheduled tasks are handled by Celery. The `celery.py` file configures the Celery application. Tasks are defined in the Django application and scheduled using Celery Beat. We scheduled two tasks, that are run every minute: Update Top 10 Ranking table and delete all jokes, that have a minus-ranking of 4 or more. Celery is the go-to task scheduler for Django, thats why we chose to use it.

Persistent Storage

Persistent storage is managed using PostgreSQL, as defined in the `docker-compose.yml` file. The volume `postgres_data` ensures that the database persists across container restarts. This seemed like the easiest solution to combine with Docker.

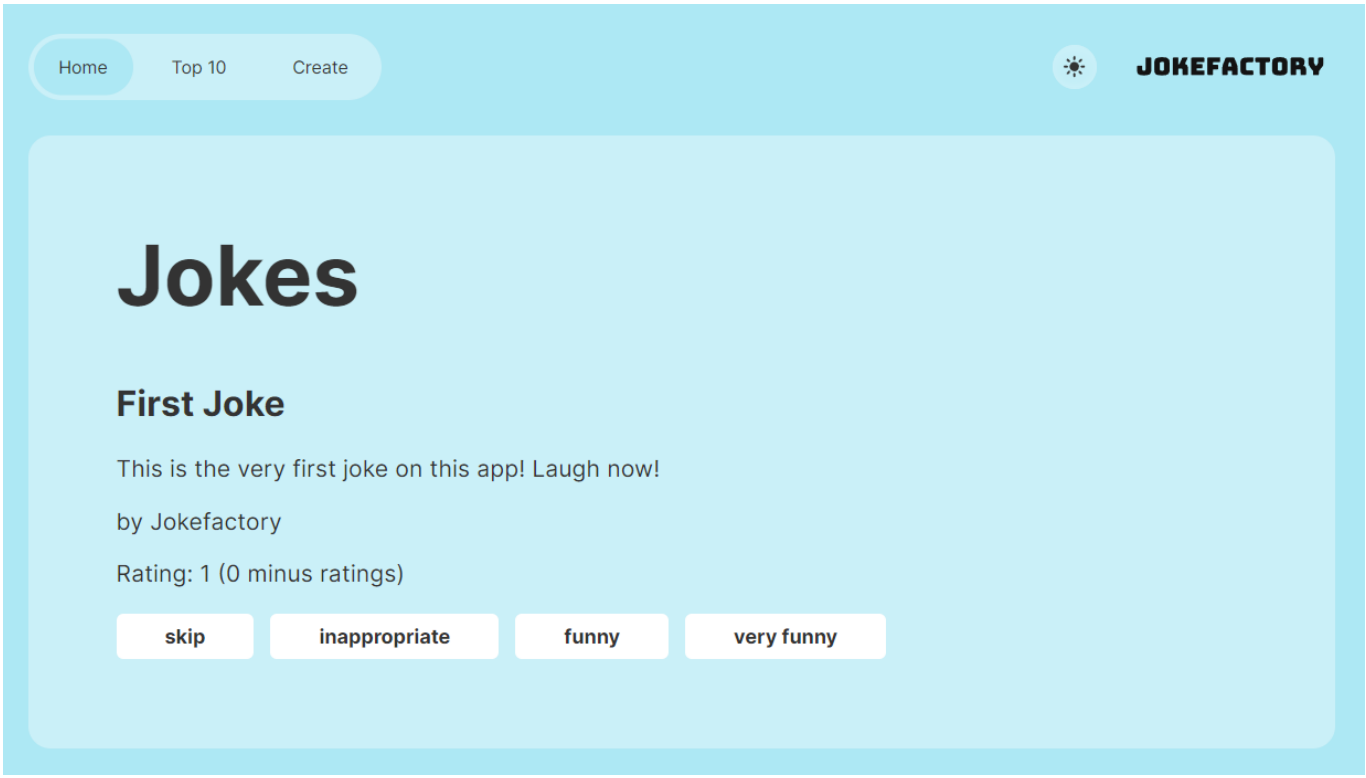
Frontend

The frontend consists of basic HTML pages. It is implemented with Django templates rendering HTML with forms for submitting and displaying jokes. We chose Django because we already have some experience using it.

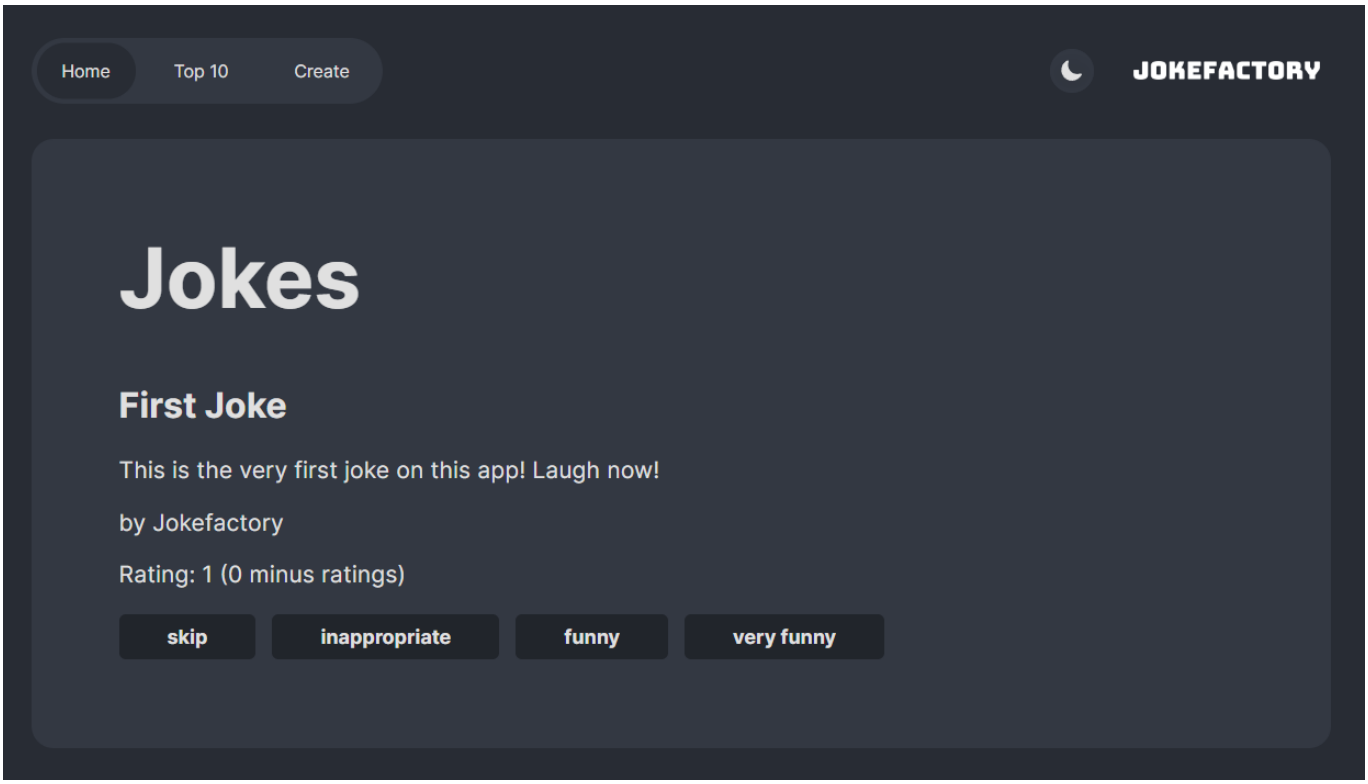
There are three different views/pages in our app: Home, Top 10 and Create. On the home page, a random joke of the submitted jokes is displayed, which you can rate as funny/unfunny. On the Top 10 Page, you can see the 10 best rated jokes (this ranking table is refreshed every minute by our task scheduler). The create page is a simple form, where the user can submit his own jokes to the app, which then can be rated by others.

There is some CSS to make the pages visually appealing and we also implemented a dark mode, as we experimented with CSS color variables. The color palette can be changed on the top right corner when pressing the sun/moon symbol.

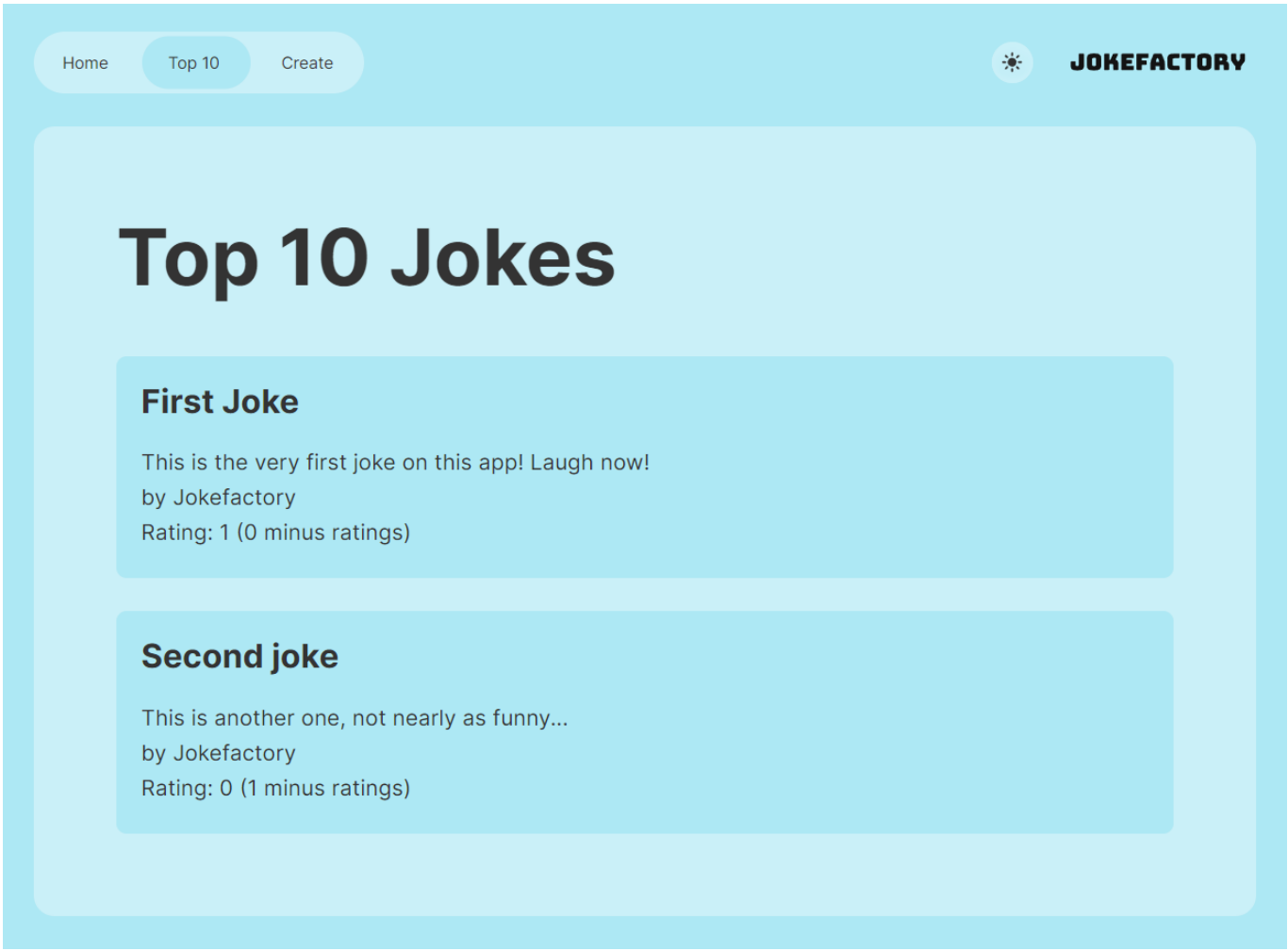
Home Page - Light Mode



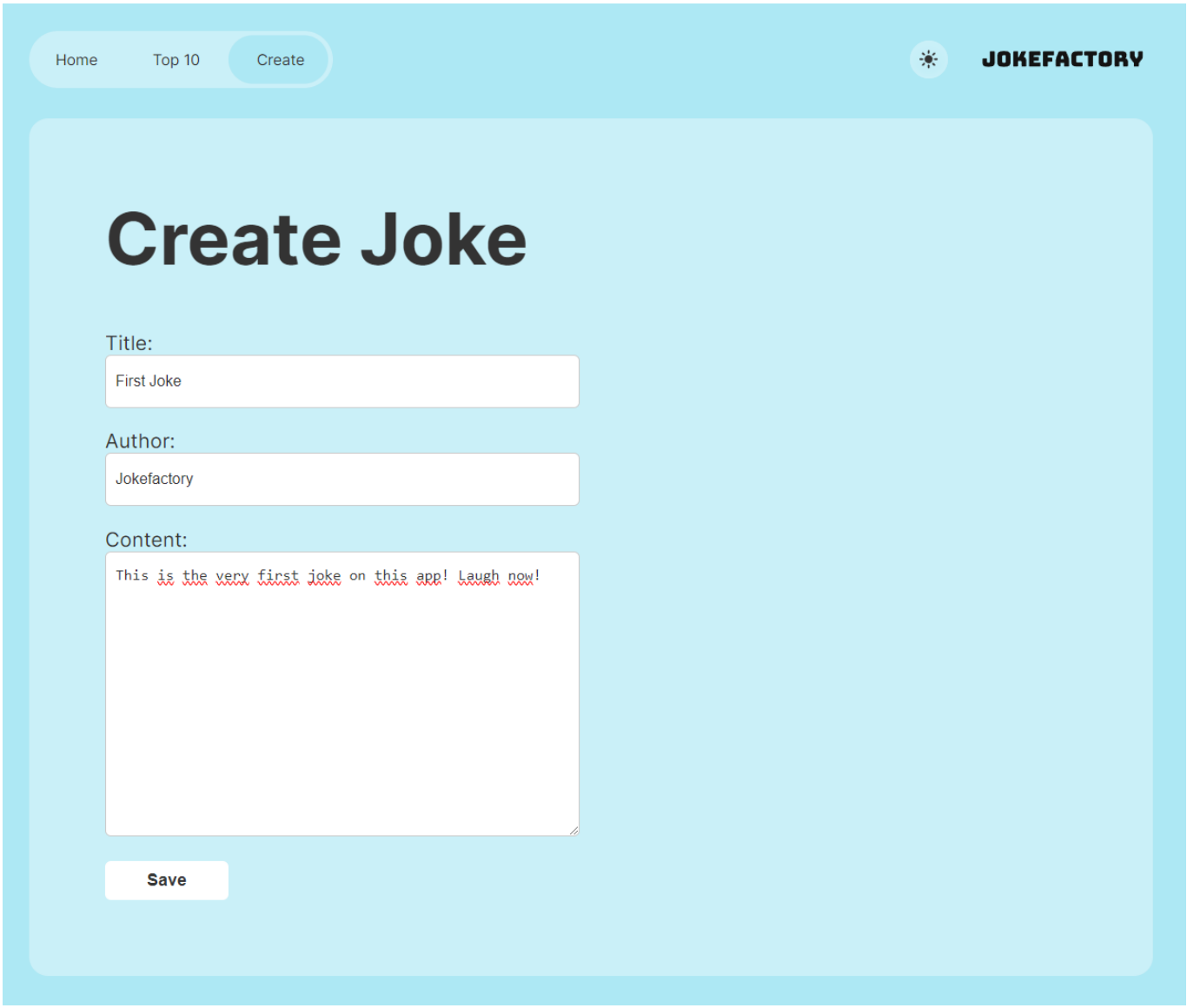
Home Page - Dark Mode



Top 10 Page


Home
Top 10
Create
JOKEFACTORY
Top 10 Jokes
First Joke
This is the very first joke on this app! Laugh now!
by Jokefactory
Rating: 1 (0 minus ratings)
Second joke
This is another one, not nearly as funny...
by Jokefactory
Rating: 0 (1 minus ratings)

Create Page


Home
Top 10
Create
JOKEFACTORY
Create Joke
Title:
First Joke
Author:
Jokefactory
Content:
This is the very first joke on this app! Laugh now!
Save

Deployment Instructions

Navigate to jokefactory_project folder and use these commands:

Using Docker Compose:

1. Build and start services:

```
docker-compose up --build
```

2. Stop services:

```
docker-compose down
```

Using Docker Swarm:

1. Build the Docker image:

```
docker build --progress=plain -t django-app:latest .
```

2. Deploy the stack:

```
docker stack deploy -c docker-compose.yml jokefactory
```

3. Remove the stack:

```
docker stack rm jokefactory
```

Additional Resources

- Traefik Quick Start
- Django Documentation
- Celery Documentation
- Docker Documentation