```
import argparse
import asyncio
import json
from typing import AsyncGenerator

import uvicorn
from executor import GenerationExecutor
from fastapi import FastAPI, Request
from fastapi.responses import JSONResponse, Response, StreamingResponse
from swarms import Fuyu

TIMEOUT_KEEP_ALIVE = 5  # seconds.
TIMEOUT_TO_PREVENT_DEADLOCK = 1  # seconds.
app = FastAPI()
executor: GenerationExecutor | None = None


@app.get("/stats")
async def stats() -> Response:
    assert executor is not None
    return JSONResponse(json.loads(await executor.aget_stats()))


@app.get("/health")
async def health() -> Response:
    """Health check."""
```

```
    return Response(status_code=200)


@app.post("/generate")
async def generate(request: Request) -> Response:
    assert executor is not None
    """Generate completion for the request.

    The request should be a JSON object with the following fields:
    - prompt: the prompt to use for the generation.
    - stream: whether to stream the results or not.
    - other fields: the sampling parameters (See `SamplingParams` for details).
    """
    request_dict = await request.json()

    streaming = request_dict.pop("streaming", False)

    model_name = request.query_params.get("model_name")
    max_new_tokens = request.query_params.get("max_new_tokens")

    model = Fuyu(
        model_name=model_name,
        max_new_tokens=max_new_tokens,
        args=args,  # Injecting args into the Fuyu model
    )
    response = model.run(
```

```
    request_dict.pop("prompt"), request_dict.pop("max_num_tokens", 8), streaming
)

async def stream_results() -> AsyncGenerator[bytes, None]:
    async for output in response:
        yield (json.dumps({"text": output.text}) + "\n").encode("utf-8")

if streaming:
    return StreamingResponse(stream_results(), media_type="text/plain")

# Non-streaming case
await response.await_completion()

# Return model configurations as JSON
model_config = {
    "model_name": model.model_name,
    "max_new_tokens": model.max_new_tokens,
    "args": {
        "model_dir": args.model_dir,
        "tokenizer_type": args.tokenizer_type,
        "max_beam_width": args.max_beam_width,
    },
}

return JSONResponse(
    {"model_config": model_config, "choices": [{"text": response.text}]}
```

```
    )



async def main(args):
    global executor

    executor = GenerationExecutor(
        args.model_dir, args.tokenizer_type, args.max_beam_width
    )
    config = uvicorn.Config(
        app,
        host=args.host,
        port=args.port,
        log_level="info",
        timeout_keep_alive=TIMEOUT_KEEP_ALIVE,
    )
    await uvicorn.Server(config).serve()



if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("model_dir")
    parser.add_argument("tokenizer_type")
    parser.add_argument("--host", type=str, default=None)
    parser.add_argument("--port", type=int, default=8000)
    parser.add_argument("--max_beam_width", type=int, default=1)
```

```
args = parser.parse_args()


asyncio.run(main(args))
```