Assignment: Build a Dynamic Content Management API

Goal: Your task is to build a "headless" Content Management System (CMS) API using FastAPI. The API must be able to accept and validate different types of content ("Blog Post," "Author Profile," etc.), each with its own unique structure, using a single endpoint.

Scenario

You are the lead backend engineer for "FlexiCMS," a new API-first content platform. Your system allows administrators to define different "Content Models" in a UI. For example, they might create a model for a "Blog Post" with fields for a title and author, and another for an "Author Profile" with fields for a name and a biography.

Your job is to build the API endpoint that content editors will use to submit entries for these models. The endpoint must dynamically validate the incoming data against the correct Content Model.

Your Tasks

Task 1: Basic FastAPI Setup

- Create a new Python file (e.g., `main.py`).
- Set up a basic FastAPI application instance.

Task 2: Define Your Content Models

- In your Python file, create a dictionary to act as your "database" for Content Model definitions. Call it `CONTENT_MODELS`.
- Define the structure for the following two models. Use a mix of data types and requirements:
  - Blog Post: Must have a `title` (string) and `author_id` (integer). It can also have an optional `tags` field (a list of strings).
  - Author Profile: Must have a `full_name` (string) and `biography` (string).

Example Structure:

```
# Fictional IDs: 'blog' for Blog Post, etc.
CONTENT_MODELS = {
   "blog_post": {
      "name": "Blog Post",
      "fields": {
         "title": (str, ...),         # Required
         "author_id": (int, ...),     # Required
         "tags": (list[str], None)    # Optional
      }
```

```
    },
    # ... add the definition for the Author Profile
}
```

Task 3: Implement the Dynamic Model Factory

- Create a dependency function (e.g., `get_content_entry_model`) that takes a `model_id: str` as an argument.
- Inside this function:
    1. Look up the model's information in your `CONTENT_MODELS` dictionary.
    2. If the model is not found, raise an `HTTPException` with a `404` status code.
    3. Use `pydantic.create_model` to generate a Pydantic class based on the `fields` defined for that model.
    4. Return the newly created model class.

Task 4: Create the Content Submission Endpoint

- Create a `POST` endpoint at the path `/entries/{model_id}`.
- This endpoint should depend on the model factory function from Task 3.
- It should accept a request body and allow FastAPI to validate it against the dynamically generated model.
- If validation is successful, return a confirmation, such as `{"message": "Content entry created successfully!", "model_id": model_id, "data": ...}`.

Task 5: Add Advanced Validation

- Let's add more specific rules. Modify your `CONTENT_MODELS`.
- Use `pydantic.Field` to add the following constraints:
    - The `title` of a "Blog Post" must have a minimum length of 3 characters and a maximum length of 100.
    - The `biography` for an "Author Profile" must have a maximum length of 500 characters.

Bonus Challenges

1. Unique Entry ID: Add a non-optional `entry_id` field to every content model you generate. Use a `default_factory` to automatically generate a unique ID for each new submission (e.g., using `uuid.uuid4`).
2. Externalize Definitions: Store your `CONTENT_MODELS` in a separate `models.json` file. Read and parse this file when your application starts. This is a crucial step toward building a truly dynamic system where models can be changed without touching the application code.