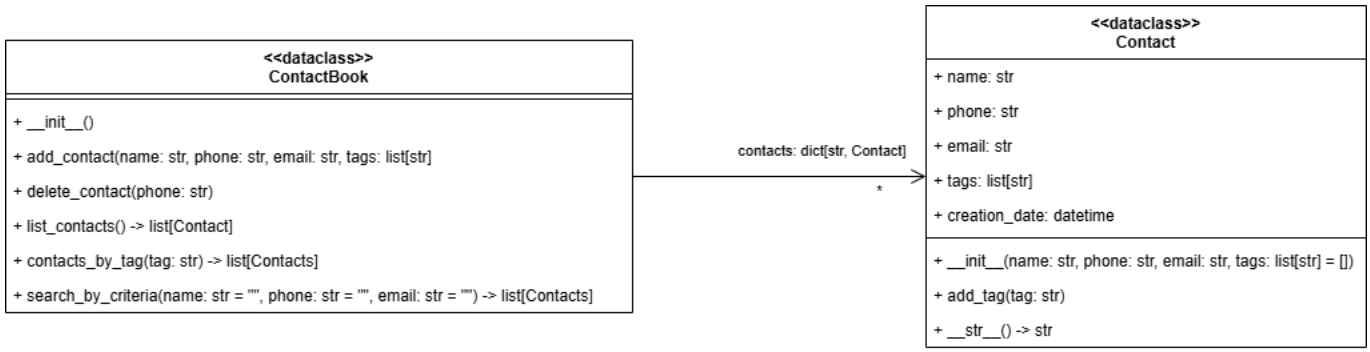
Contact Book

This is a practice application for OOP concepts in Python. The application is a simple notebook with the following features:

- **Add a new contact:** Allows users to add a new contact by specifying the name, phone number, email, and tags (e.g., friend, coworker, etc.). The system assigns a unique phone number to each contact.
- **List all contacts:** Displays all contacts currently stored in the contact book.
- **Search by multiple criteria:** Provides the functionality to search for contacts using combinations of name, phone number, and/or email address.
- **Delete a contact:** Users can delete a contact from the contact book by providing the phone number of the contact.
- **Search by tag:** Filters contacts based on a specific tag assigned to them (e.g., work, friend, etc.).

The domain model for the application consists of the following classes:



Your task is to implement the model in the `contactbook/model/contacts.py` file. Consider the following guidelines:

1. Class `Contact`

- The class should be implemented as a `dataclass`.
- The class should have the following attributes:
  - `name` of type `str` initialized with a parameter in the constructor.
  - `phone` of type `str` initialized with a parameter in the constructor.
  - `email` of type `str` initialized with a parameter in the constructor.
  - `tags` of type `list[str]` initialized with a parameter in the constructor, but with an empty list as the default value.
  - `creation_date` of type `datetime`. This attribute is **not** initialized with a parameter in the constructor but should be set to the current date and time when the object is created.

    > **Hint:** You can use the `field` function from the `dataclass` module and the `datetime.now()` function from the `datetime` module.

- The class should have an instance method `add_tag` that receives a parameter `tag` of type `str` and adds it to the `tags` attribute. The method should not add the tag if it already exists in the list.

- The class should have and instance method `__str__` that returns a `str` with the following format:

```
Name: {name}
Phone: {phone}
Email: {email}
Tags: {tags}
Created on: {creation_date}
```

  Where `{name}` should be replaced by the value of the `name` attribute, `{phone}` by the value of the `phone` attribute, `{email}` by the value of the `email` attribute, `{tags}` by the value of the `tags` attribute separated by commas, and `{creation_date}` by the value of the `creation_date` attribute.

2. Class `ContactBook`

- The class should be implemented as a `dataclass`.
- The class should have an instance attribute `contacts` of type `dict[str, Contact]` which is not initialized with a parameter in the constructor but with an empty dictionary as the default value.
- The class should have an `add_contact` method that receives the following parameters:
  - `name` of type `str`.
  - `phone` of type `str`.
  - `email` of type `str`.
  - `tags` of type `list[str]`.

  The method should create a new `Contact` object with the provided parameters and add it to the `contacts` dictionary using the `phone` as the key.
- The class should have a `delete_contact` method that receives a `phone` parameter of type `str` and removes the contact with the provided phone number from the `contacts` dictionary.
- The class should have a `list_contacts` method that returns a `list[Contact]` with all the contacts in the `contacts` dictionary.
- The class should have a `contacts_by_tag` method that receives a `tag` parameter of type `str` and returns a `list[Contact]` with all the contacts that have the provided tag in their `tags` attribute.
- The class should have a `search_by_criteria` method that receives the following parameters:
  - `name` of type `str`.
  - `phone` of type `str`.
  - `email` of type `str`.

  The method should return a `list[Contact]` with all the contacts that match the provided criteria. If a parameter is an empty string, it should not be considered in the search. For example, if the `name` parameter is an empty string, the method should return all contacts that match the `phone` and `email` criteria, but not the `name` criteria. The search should ignore the case of the strings when comparing them and should consider a match if the search criteria is a substring of the contact's attribute.