Service Endpoint

DogPro application's service endpoints:

1. /users

   Methods: `POST`, `GET`, `PUT`, `DELETE`

   MongoDB Collection: `Users`

2. /users/{userID}/dogs

   Methods: `POST`, `GET`, `PUT`, `DELETE`

   MongoDB Collection: `Users` (Embedded `dogProfiles` array in user document)

3. /dogs/{dogID}/diet

   Methods: `GET`, `PUT`

   MongoDB Collection: `Users` (Nested `feedingSchedule` object in `dogProfiles` array)

4. /dogs/{dogID}/potty

   Methods: `GET`, `PUT`

   MongoDB Collection: `Users` (Nested `pottySchedule` object in `dogProfiles` array)

5. /users/{userID}/login

   Methods: `POST`

   MongoDB Collection: `Users`

6. /training-resources

   Methods: `GET`

   MongoDB Collection: `TrainingResources`

Endpoints Description

1. /users - Utilized to create (`POST`), fetch (`GET`), update (`PUT`), and delete (`DELETE`) user information. Corresponds to the `Users` collection in MongoDB.

2. /users/{userID}/dogs - Used to add (`POST`), fetch (`GET`), update (`PUT`), and remove (`DELETE`) dogs owned by a user. Interacts with the `dogProfiles` field in the `Users` collection.

3. /dogs/{dogID}/diet - Used to fetch (`GET`) and update (`PUT`) the diet details of a dog. Engages with the `feedingSchedule` object within the `dogProfiles` field in the `Users` collection.

4. /dogs/{dogID}/potty - Used to fetch (`GET`) and update (`PUT`) the potty schedule of a dog. Communicates with the `pottySchedule` object within the `dogProfiles` field in the `Users` collection.

5. /users/{userID}/login - Used for user login (`POST`). Maps to the `Users` collection.

6. /training-resources - Utilized to fetch (GET) all training resources for a dog owner. This endpoint interacts with the TrainingResources collection in MongoDB and retrieves articles, and videos that aid the dog training process. No modification (POST, PUT, DELETE) methods are supported, as this collection is treated as a read-only repository of resources.

Example Requests and Responses

1. /users - POST - Register a new user

Request:

```
{
   "name": "User1",
   "email": "user1@email.com",
   "password": "password"
}
```

Response:

```
{
```

```
    "status": "success",
    "data": {
      "_id": ObjectId(),
      "name": "User1",
      "email": "user1@email.com"
    }
  }
```


2. /users/{userID}/dogs - POST - Add a new dog for a user

Request:


```
{
    "dogName": "Rex",
    "dogBreed": "Labrador",
    "dogAge": 3,
    "dogWeight": 30,
    "feedingSchedule": {
      "breakfast": "8:00 AM",
      "lunch": "1:00 PM",
      "dinner": "7:00 PM"
    },
    "pottySchedule": {
      "morning": "7:00 AM",
      "afternoon": "3:00 PM",
      "night": "10:00 PM"
    },
    "trainingList": ["Sit", "Stay", "Fetch"]
}
```

Response:


```
{
   "status": "success",
   "data": {
      "_id": ObjectId(),
      "name": "User1",
      "email": "user1@email.com",
      "dogProfiles": [
         {
            "dogName": "Rex",
            "dogBreed": "Labrador",
            "dogAge": 3,
            "dogWeight": 30,
            "feedingSchedule": {
               "breakfast": "8:00 AM",
               "lunch": "1:00 PM",
               "dinner": "7:00 PM"
            },
            "pottySchedule": {
               "morning": "7:00 AM",
               "afternoon": "3:00 PM",
               "night": "10:00 PM"
            },
            "trainingList": ["Sit", "Stay", "Fetch"]
         }
      ]
   }
}
```

3. /users/{userID}/dogs/{dogID}/feedingSchedule - PUT - Update the feeding schedule of a dog

Request:


```
{
   "feedingSchedule": {
      "breakfast": "7:00 AM",
      "lunch": "12:00 PM",
      "dinner": "6:00 PM"
   }
}
```


Response:


```
{
   "status": "success",
   "data": {
      "_id": ObjectId(),
      "dogName": "Rex",
      "dogBreed": "Labrador",
      "dogAge": 3,
      "dogWeight": 30,
      "feedingSchedule": {
         "breakfast": "7:00 AM",
         "lunch": "12:00 PM",
         "dinner": "6:00 PM"
      },
      "pottySchedule": {
         "morning": "7:00 AM",
```

```
        "afternoon": "3:00 PM",
        "night": "10:00 PM"
      },
      "trainingList": ["Sit", "Stay", "Fetch"]
    }
  }
```


4. /users/{userID}/dogs/{dogID}/pottySchedule - PUT - Update the potty schedule of a dog

Request:


```
{
  "pottySchedule": {
    "morning": "6:30 AM",
    "afternoon": "2:30 PM",
    "night": "9:30 PM"
  }
}
```


Response:


```
{
  "status": "success",
  "data": {
    "_id": ObjectId(),
    "dogName": "Rex",
    "dogBreed": "Labrador",
    "dogAge": 3,
    "dogWeight": 30,
    "feedingSchedule": {
      "breakfast": "7:00 AM",
```

```
            "lunch": "12:00 PM",
            "dinner": "6:00 PM"
        },
        "pottySchedule": {
            "morning": "6:30 AM",
            "afternoon": "2:30 PM",
            "night": "9:30 PM"
        },
        "trainingList": ["Sit", "Stay", "Fetch"]
    }
}
```

5. /users/{userID}/login - POST - User login

Request:

```
{
    "email": "user1@email.com",
    "password": "password"
}
```

Response:

```
{
    "status": "success",
    "data": {
        "_id": ObjectId(),
        "name": "User1",
        "email": "user1@email.com"
    }
}
```

6. /training-resources - GET - Fetch all training resources

Request:

None

Response:

```
{
   "status": "success",
   "data": [
      {
         "_id": ObjectId(),
         "type": "Article",
         "title": "Training your dog to fetch",
         "url": "https://example.com/article"
      },
      {
         "_id": ObjectId(),
         "type": "Video",
         "title": "Teaching your dog to sit",
         "url": "https://example.com/video"
      },
      // more resources...
   ]
}
```

Error Responses Example

1. /users - POST - Create a new account without providing necessary data

Request:

```
{
   "username": "",
```

"email": "user@example.com",

"password": "12345678"

}

Response:

{

"status": failure,

"message": "Username is required."

}

2. /users/{userID} - GET - Fetch a user's account with an invalid userID

Response:

{

"status": failure,

"message": "User not found."

}

3. /users/{userID}/dogs - POST - Add a new dog for a user without providing necessary data

Request:

{

"dogName": "",

"dogBreed": "Labrador",

"dogAge": 3,

"dogWeight": 30,

}

Response:

{

"status": "failure",

"message": "Dog name is required."

}

Diagram of Service Endpoints Interaction with User Interface Pages

1. Home Page -> GET /
2. About Us Page -> GET /about-us
3. Behavioral Resource Page -> GET /training-resources
4. Sign Up Page -> POST /users
5. Login Page -> POST /users/{userID}/login
6. User Profile Page -> GET /users/{userID}, PUT /users/{userID}, DELETE /users/{userID}
7. Add Dog Page -> POST /users/{userID}/dogs
8. Dog Profile Page -> GET /users/{userID}/dogs/{dogID}, PUT /users/{userID}/dogs/{dogID}, DELETE /users/{userID}/dogs/{dogID}
9. Diet Page -> POST /calculateDiet (PHP processing only, no DB interaction)
10. Potty Page -> POST /calculatePotty (PHP processing only, no DB interaction)

The frontend will interact with these endpoints using the fetch API to send and receive data. PHP will handle the diet and potty calculations.

Stretch Feature

Stretch Feature Endpoints

/clicker

Methods: GET

/users/{userID}/dogs/{dogID}/avatar

Methods: POST, GET

Stretch Feature Endpoints Description

/clicker - This endpoint will be used to fetch (GET) a sound file to be used as a dog clicker sound effect. This will not interact with the database but serve static resources.

/users/{userID}/dogs/{dogID}/avatar - This endpoint will be used to upload (POST) and fetch (GET) a profile avatar for a dog. This will interact with the dogProfiles field in the Users collection.

Example Requests and Responses

/clicker - GET - Fetch the dog clicker sound file

Request:

None

Response:

{

"status": "success",

"data": {

"soundFileUrl": "/sounds/clicker.mp3"

}

}

/users/{userID}/dogs/{dogID}/avatar - POST - Upload a profile avatar for a dog

Request:

Form data with an image file

Response:

{

"status": "success",

"data": {

"_id": ObjectId(),

"avatarUrl": "/avatars/dogID.jpg"

}

}

Diagram of Stretch Feature Service Endpoints Interaction with User Interface Pages

Dog Clicker Page -> GET /clicker

Dog Profile Edit Page -> POST /users/{userID}/dogs/{dogID}/avatar

This frontend will interact with these endpoints using the fetch API to send and receive data. The dog clicker sound effect will be fetched and played on the client side. The profile avatar will be uploaded and stored in the MongoDB database.

Note: The only change from the original file is the Stretch Feature section just above to explain what it would take to develop after deployment of MVPs.