Service Layer Design

Jesse Lindahl

Service Layer Overview

The current plan for the backend service layer of the “Fit Quick” application is to build it out using the Django REST framework. This Django REST framework will give me the capability to build up a REST API for this project to allow for proper data utilization between the frontend and backend of the application.

Service Layers and Endpoints Overview

Listed below are the various elements that will make up my applications service layers and endpoints. These elements are all conceptual at this time and will likely see changes over the development cycle for this project, but this is my current vision for how to bring my MVP to life through the lens of service layers and user endpoints. Many of the URL’s and examples listed below are placeholders or nonfunctional at this time, but could eventually be utilized for the final version of the application.

User:

1. Login

   Method: GET

   URL: https://sweet-poetry-0043.bss.design/login.html

   In order for a user to gain access to the application they will need to either create an account or login to an existing one that has previously been set up. This login page will be a simple forum that will prompt users for a username and password for the application. Once the login credentials have been confirmed, the user will be given access to the application and sent to the starting home page.

   Example Request: request.GET -url https://sweet-poetry-0043.bss.design/login.html

   Successful Response:

   HTTP 200 OK
   Allow: GET, POST
   Content-Type: application/json
   Vary: Accept

```
{
  “Login”: [
    {
     "id": 0,
     "username": "string",
     "password": "string",
     "userStatus": 0
    }
   ]
}
```

   Error Response:

   HTTP 404 Not Found
   Allow: GET, POST
   Content-Type: application/json
   Vary: Accept

```
{
  “code”: 404,
  "message": “No account recognized.”,
}
```

Diagram:

2. Dashboard Navigation

Method: PUT

URL: https://sweet-poetry-0043.bss.design/home.html

Upon successfully logging into an account, the user should be able to easily navigate the dashboard tabs to access the various pages of the application. There will only be a limited number of service endpoints a user can access within this layer of the application, but the ability to do so will be critical for the MVP of this project. A user's ability to successfully navigate the home page and dashboard tabs is the first step in ensuring all service endpoints can be reached properly.

Example Request: request.PUT -url https://sweet-poetry-0043.bss.design/home.html

Successful Response:

HTTP 200 OK
Allow: GET, PUT
Content-Type: application/json
Vary: Accept

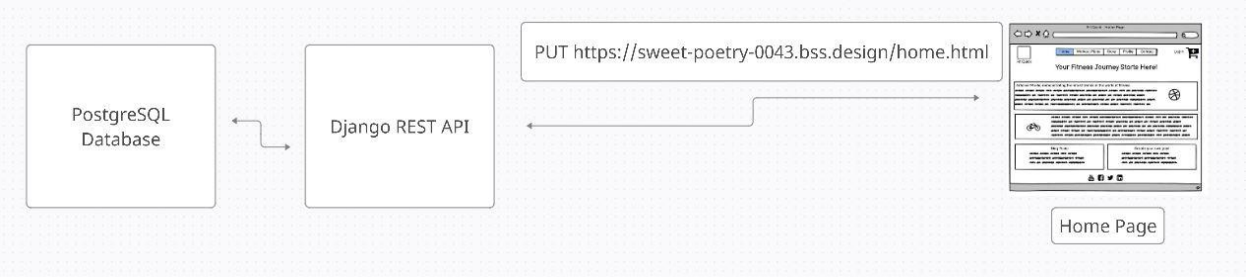
```
{
 “home_page”: [
   {
    "id": 111111,
    "home": "https://sweet-poetry-0043.bss.design/home.html",
    "plans": "https://sweet-poetry-0043.bss.design/workoutplans.html",
    "store": "https://sweet-poetry-0043.bss.design/store.html",
    "contact": "https://sweet-poetry-0043.bss.design/contact.html",
    "cart": "https://sweet-poetry-0043.bss.design/cart.html",
    "profile": "https://sweet-poetry-0043.bss.design/profile.html",
   ]
 }
}
```


Error Response:

HTTP 400 Bad Request
Allow: GET, PUT
Content-Type: application/json
Vary: Accept


```
{
  “code”: 400,
  "message": “Invalid selection.”,
}
```


Diagram:



Blog:

1. Blog Post

   Method: POST

   URL: https://sweet-poetry-0043.bss.design/home.html

Directly on the home page, users will have the ability to create customized blog posts centered around fitness recommendations and questions. This particular service endpoint will begin and end right on the home screen as the blogging feature will exist as a forum near the middle of page. Users will have the capability to create, share, search for and comment on posts from the forum section. Blog posts will be viewable and interactable by other application users who have created an account. Users can also edit and delete their own posts from this service endpoint.

Example Request: request.POST -url https://sweet-poetry-0043.bss.design/home.html

Successful Response:

HTTP 200 OK
Allow: GET, PUT, POST, DELETE
Content-Type: application/json
Vary: Accept

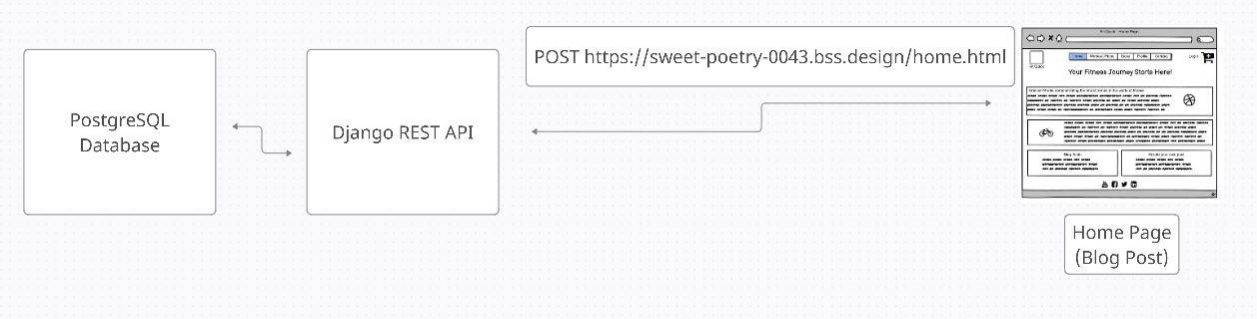

```
{
 “home_page”: [
   {
    "id": 222222,
    "post": "string",
    "comment": "string",
    "share": "string",
    "search": "string",
    "delete": "string",
    "edit": "string",
   ]
 }
}
```


Error Response:

HTTP 400 Bad Request
Allow: GET, PUT, POST, DELETE
Content-Type: application/json
Vary: Accept


```
{
  “code”: 400,
  "message": “Blog failed to post. Please try again.”,
}
```

Diagram:



Ecommerce Ordering:

1. Store

   Method: GET

   URL: https://sweet-poetry-0043.bss.design/store.html

   When a user clicks on the 'Store' tab, this service endpoint will redirect them to the ecommerce storefront section of the application. Products and services will be displayed in rows and columns that are recorded on the backend of the application and will be interactable on the frontend by the user. The users will then be able to assemble an order based on the products displayed on the storefront that will then be stored at the shopping cart service endpoint for later use.

   Example Request: request.GET -url https://sweet-poetry-0043.bss.design/store.html

   Successful Response:

   HTTP 200 OK
   Allow: GET, PUT, DELETE
   Content-Type: application/json
   Vary: Accept

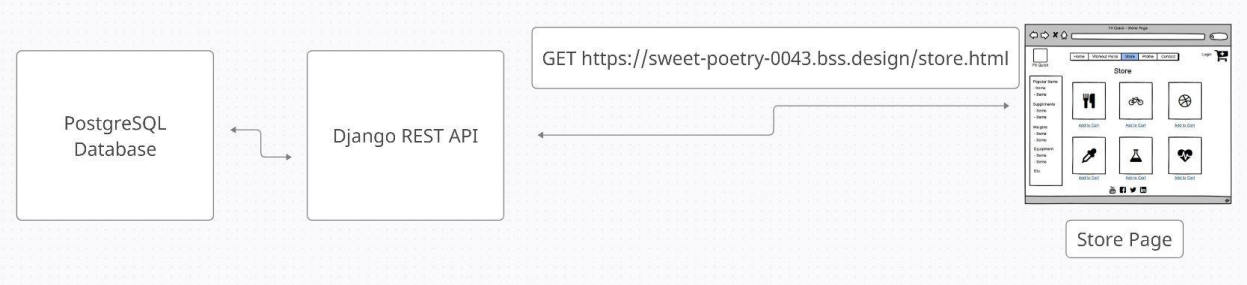
```
{
 “store_page”: [
   {
    "id": 33333,
    "products": "string",
    "quantities": "int",
    "delete": "string",
    "share": "string",
    "add": "string",
   ]
 }
}
```


Error Response:

HTTP 404 Not Found
Allow: GET, PUT, DELETE
Content-Type: application/json
Vary: Accept


```
{
  “code”: 404,
  "message": “Selection not found.”,
}
```


Diagram:



2. Cart

Method: PUT

URL: https://sweet-poetry-0043.bss.design/cart.html

When an order has been assembled, the shopping cart service endpoint will need to be called on to properly complete the ordering process. The shopping cart service endpoint can be reached from any screen on the application and will have all the product data stored in it that a

user has selected from the ‘Store’ tab. A user will need to review their order within the cart and provide final confirmation once they are satisfied with their selection.

Example Request: request.GET -url https://sweet-poetry-0043.bss.design/cart.html

Successful Response:

HTTP 200 OK
Allow: GET, PUT, DELETE
Content-Type: application/json
Vary: Accept

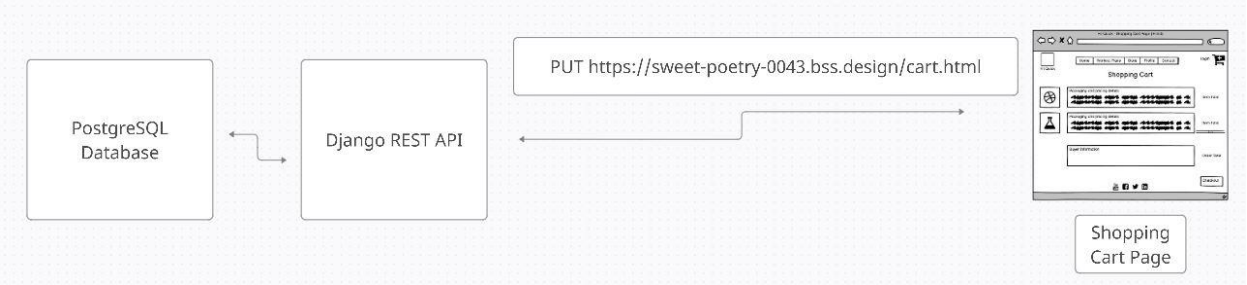
```
{
 “cart_page”: [
   {
    "id": 44444,
    "products": "string",
    "quantities": "int",
    "delete": "string",
    "confirmation": "string",
    "user_details": "string",
    "payment": "int",
   ]
 }
}
```

Error Response:

HTTP 400 Bad Request
Allow: GET, PUT, DELETE
Content-Type: application/json
Vary: Accept

```
{
  “code”: 400,
  "message": “Process Failed.”,
}
```

Diagram:



Workout Plans:

1. Workout Plans

   Method: GET

   URL: https://sweet-poetry-0043.bss.design/plans.html

   When users interact with the content within the “Workout Plans” tab, this service endpoint will be called upon to present the workout plans stored on the backend of the application. Depending on the workout plans that are selected, this service endpoint will execute upon itself by presenting a user with the data they are looking for as it exists within the applications database. This endpoint will conclude when a user has downloaded, shared or completed a search for a given workout plan.

   Example Request: request.GET -url https://sweet-poetry-0043.bss.design/plans.html

   Successful Response:

   HTTP 200 OK
   Allow: GET
   Content-Type: application/json
   Vary: Accept

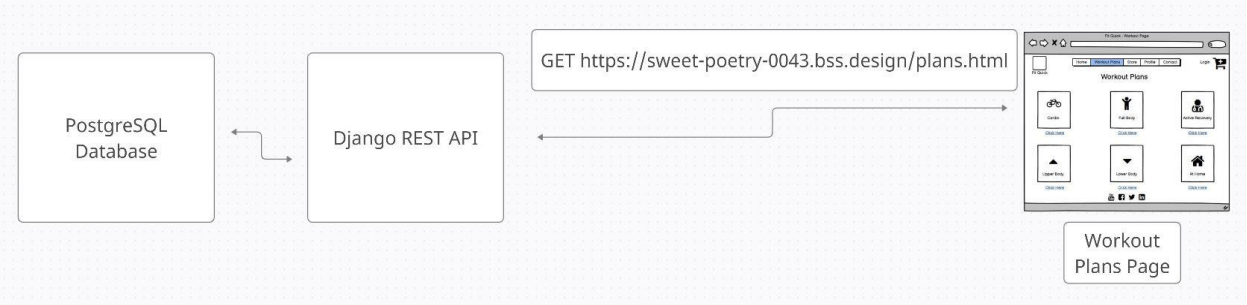
```
{
 “plans_page”: [
   {
    "id": 55555,
    "plan": "string",
    "search": "string",
    "share": "string",
    "download": "string",
   ]
 }
}
```


Error Response:

HTTP 404 Not Found
Allow: GET
Content-Type: application/json
Vary: Accept


```
{
  “code”: 404,
  "message": “Selection not found.”,
}
```


Diagram:



Contact:

1. Contact Forum

   Method: POST

   URL: https://sweet-poetry-0043.bss.design/contact.html

When users interact with the “Contact” tab, this service endpoint will be called upon to submit a message request to the application administrator. This endpoint will interact with the user by given them the ability to write a message in a dedicated forum that can be submitted through the frontend features of the application and recorded onto the backend.

Example Request: request.GET -url https://sweet-poetry-0043.bss.design/contact.html

Successful Response:

HTTP 200 OK
Allow: POST
Content-Type: application/json
Vary: Accept

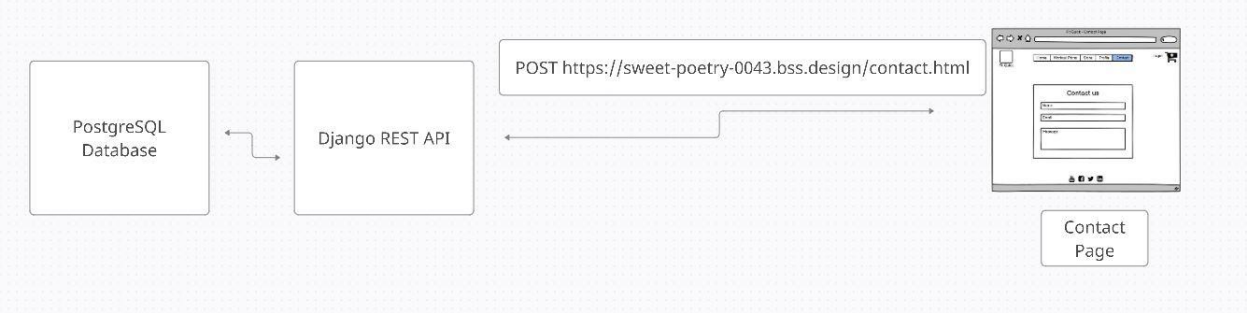
```
{
 “contact_page”: [
   {
    "id": 4546,
    "email_address": "string",
    "userID": "string",
    "message": "string",
    "post": "string",
   ]
 }
}
```

Error Response:

HTTP 404 Not Found
Allow: POST
Content-Type: application/json
Vary: Accept

```
{
  “code”: 400,
  "message": “Invalid Entry.”,
}
```

Diagram:



Conclusion

The service endpoints listed above serve to highlight the features needed to complete the MVP for this application and as a result have the capability of increasing in scope over the course of the project. I am hoping I will have the ability to expand upon these service layers and endpoints as the project starts to come together, but there should also be no issues executing on my vision for the MVP as this design is currently structured. The service endpoints should all have very tight and obvious endpoints for how the user will interact with them across the application, so my ultimate goal is to keep these layers as simple as can be.