Night Owl Developers Documentation

Sprint 2: October 11 , 2021 - October 22, 2021

Overview

- Project goal: To build a web application that functions as an “Airbnb for pets”, with features that facilitate interactions between pet owners and pet service providers.
- Application features for this sprint:
    - Frontend: Service Page, Product Page, Media Page, Create Media Page, Detail Service Page, Detail Product Page, and Detail Media Page.
    - Backend: DB tables setup, APIs for services, products and comments, authentication logic, user session logic, filtering and sorting
- Team organization:
    - Frontend: Frank, Ivan, Jonathan, Meng
    - Backend: Gary, Jesse, Tyler

Frontend:

The frontend is built with React. Libraries used include Bootstrap. Below are the frontend features prioritized in Sprint 2.

Registration page: Users can register an account on the website by providing a username, password, email, and location.

Login page: Users can login to the website by providing their username and password.

Account page: After users are logged in, they can access their accounts page. On this page they can see options to edit their account information such as password, email, and location.

HomePage Search: Users can search for service providers by using Location, price range, pet breed, number of pets.

Media Page: Users can view a list of media posts.

Create Media Page: Users can send post requests to the backend to create media posts.

Product Page: Users can see a list of Product page cards based on selected parameters, filters and sort them accordingly.

Service Page:Users can see a list of Service page cards based on selected parameters, filters and sort them accordingly.

Detail Media Page: Detail Media Page shows the detail description, pictures and comments of the media page.

Detail Product Page: Detail Product Page shows the detail description, pictures and comments of the Product page.

Detail Service Page: Detail Service Page shows the detail description, pictures and comments of the Service page.

Backend:

The backend server logic is built with Express and Node. Other Node modules used include Passport.

PostgreSQL is used as our database management system. Below are the tables currently used by our application:

users

uid	integer
username	varchar(150)
password	varchar(150)
email	varchar(250)
location	varchar(150)

services

service_id	integer
provider_id	varchar(200)
provider_name	varchar(200)
provider_phone	varchar(200)
provider_email	varchar(200)

provider_avatar	varchar(200)
service_pic_url	varchar(200)[]
service_title	varchar(200)
service_detail	varchar(200)
service_facility	varchar(200)
location	varchar(200)
service_rating	decimal
price_per_day	decimal
service_pet_breed	varchar(200)

products

product_id	integer
product_detail	varchar(200)
product_name	varchar(200)
product_origin	varchar(200)
product_category	varchar(200)
product_pet_breed	varchar(200)
product_price	decimal
product_type	varchar(200)
product_pic_url	varchar(200)[]
product_rating	decimal

comments

comment_id	integer
comment_type	varchar(200)
foreign_id	varchar(200)
comment_detail	varchar(200)

author_name	varchar(200)
author_profile_pic_url	varchar(200)
comment_time	varchar(200)
comment_replies	varchar(200)[]

DB API

Server location: http://localhost:3001

User API

API Method	Description	Request Body Format
GET /api/user	Gets all rows of users from the database.	
GET /api/user/:uid	Gets the user with the given uid.	

POST /api/user	Create and insert a new user into the database.	{
		“username”:”value”,
		“password”:”value”,
		“email”:”value”,
		“location”:”value
		}

PUT /api/user/:uid	Edit the user’s password, email, or location fields. Username cannot be changed. At least one of these fields must be in the request body.	{
		“password”:”value”,
		“email”:”value”,
		“location”:”value
		}

Auth API

API Method	Description	Request Body Format

POST /api/auth/login	Login as a user using the login credentials.	{
		username: string,
		password: string
		}

POST /api/auth/logout	Log out of the user’s account.	

GET /api/auth/user	Check if current user is logged in, and if so, get the user details	

Services API

API Method	**Description**	**Request Body Format**
GET /api/services	Gets all rows of services from the database. Allows users to filter and sort on specific columns using query params.	Query params: - locations - pet_breeds - minPrice - maxPrice - sortBy - sortDirection
GET /api/services/:service_id	Get the service with the given service_id	

POST /api/services	Create an insert a new service into the database	{
		“provider_id”:”value”,
		“provider_name”:”value”,
		“provider_phone”:”value”,
		“provider_email”:”value”,
		“provider_avatar”:”value”,
		“service_pic_url”:[”value1”, “value2”],
		“service_title”:”value”,
		“service_detail”:”value”,
		“service_facility”:[”value1”, “value2”]
		“location”:”value”,
		“rating”:”value”,
		“price_per_day”:”value”
		}

PUT /api/services/:service_id	Edits the service’s information. Must contain at least one of the fields to edit.	{
		“provider_phone”:”value”
		}

DELETE /api/services/:service_id	Removes the given service from the database.	

Products API

API Method	Description	Request Body Format
GET /api/products	Gets all rows of products from the database. Allows users to filter and sort on specific columns using query params.	Query params: - categories - pet_breeds - minPrice - maxPrice - sortBy - sortdirection
GET /api/products/product_id	Gets the product with the given product_id from the database.	

POST /api/products	Create and insert a new product into the database	{
		“product_detail”:”value”,
		“product_name”:”value”,
		“product_origin”:”value”,
		“product_price”:”value”,
		“product_type”:”value”,
		“product_pic_url”:”value”,
		“product_rating”:”value”
		}

PUT /api/products/product_id	Edits the product’s information. Must contain at least one of the fields to edit.	{
		“product_rating”:”value”
		}

DELETE /api/products/product_id	Deletes the product with the given product_id from the database.	

Comments API

API Method	Description	Request Body Format

GET /api/comments	Gets all comments belonging to a service, product, or media.
	
	comment_type can be [“product”, “service”, “media”] and foreign_id is an integer.
	
	For example, the request body to the right would get all comments belonging to the product with product_id of 5.
		{
		“comment_type”:”product”,
		“foreign_id”:5
		}

POST /api/comments	Creates an inserts a new comment into the database	{
		“comment_type”:”value”,
		“foreign_id”:”value”,
		“comment_detail”:”value”,
		“author_name”:”value”,
		“author_profile_pic_url”:”value”,
		“comment_time”:”value”,
		“comment_replies”:[[reply1], [reply2]]
		}

PUT /api/comments/:comment_id	Edits the comment’s information. At least one field must be in the request body.	{
		“comment_detail”:”value”
		}

DELETE /api/comments/:comment_id	Deletes the comment with the given comment_id from the database.	