ICOM6034 INDIVIDUAL REPORT

CHENG Lang 3035396393

Group T

I. List of Work done

1. Overall system design
2. Frontend and backend technology selection
3. Implementation of frontend and backend

II. Module Designs

A. Backend

1. File structure

backend/travel_advice/ : root directory of backend project

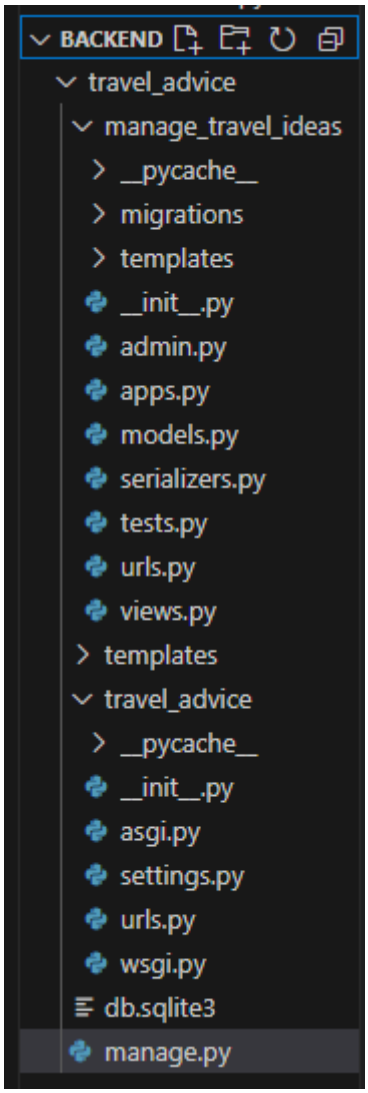
backend/travel_advice/manage_travel_ideas/:

models.py: implementation of models

serializers.py: implementation of model serializers

urls.py: routing under “/idea/”

views.py: implementation of each REST api view



backend/travel_advice/travel_advice:

settings.py: settings used for backend, notably DBMS configurations (DATABASES), Django Rest Framework authentication and permissions (REST_FRAMEWORK), CORS settings.

urls.py: “top level” routings.

Backend/travel_advice/manage.py: start server script

2. Model design

   Each Idea object has the following properties:

   author_token: used to represent the associated author, string

   title: string, has a length limit of 255

   destination: string, has a length limit of 255

   description: string, no length limit

   tags: TaggableManager() object introduced from third party library dj-taggit. It is at the end a list of strings(tags).

   start_date: date

   end_date: date

   created_at: auto generated timestamp

   updated_at: auto generated timestamp

   id: implicitly created primary key

   Models are serialized by Django Rest Framework's serializer so that they can be represented in JSON.

3. Views and APIs

   All views are REST API views.

   For creating ideas:

created_ideas(request)

```python
@api_view(["POST"])
def created_ideas(request):
    """
    CREATE a new idea post
    api: create/
    method: POST
    Example:

    {
        "title": "UPDATED",
        "author_token": "superuser",
        "destination": "GUANGZHOU",
        "description": "This is an idea",
        "tags": [
            "rest",
            "hey"
        ],
        "start_date": "2023-08-01",
        "end_date": "2023-08-03"
    }

    """
    idea = IdeaSerializer(data=request.data)

    #validation for existing data
    # if Idea.objects.filter(**request.data).exists():
    #     raise serializers.ValidationError('This data already exists')

    if idea.is_valid():
        idea.save()
        return Response(idea.data)
    else:
        return Response(status=status.HTTP_404_NOT_FOUND)
```

Simply accepts a JSON representation of an idea object, create it if

data are valid, otherwise returns 404.

For querying/search ideas:

```python
@api_view(['GET'])
def view_ideas(request):
    """
    READ a list of ideas
    api: all/?key=value
    method: GET
    """
    # checking for the parameters from the URL

    if request.query_params:
        params_dict = request.query_params.dict() # dictionary that contains query params, e.g. {id: 4, author_token: "superuser"}
        if "keyword" in params_dict:
            ideas = Idea.objects.filter(Q(destination__icontains=params_dict["keyword"])|Q(tags__name__in=[params_dict["keyword"]])).distinct()
        else:
            ideas = Idea.objects.filter(**request.query_params.dict())
    else:
        ideas = Idea.objects.all()

    # if there is something in ideas else raise error, ideas is a queryset thus many=True
    if ideas:
        serializer = IdeaSerializer(ideas, many=True)
        return Response(serializer.data)
    else:
        return Response(status=status.HTTP_404_NOT_FOUND)
```

Search function is integrated into view_ideas using Q object.

Without parameters, it returns all ideas. It can match any attributes for strict searching. (e.g. ?description=”I love Japan”?id=6”)

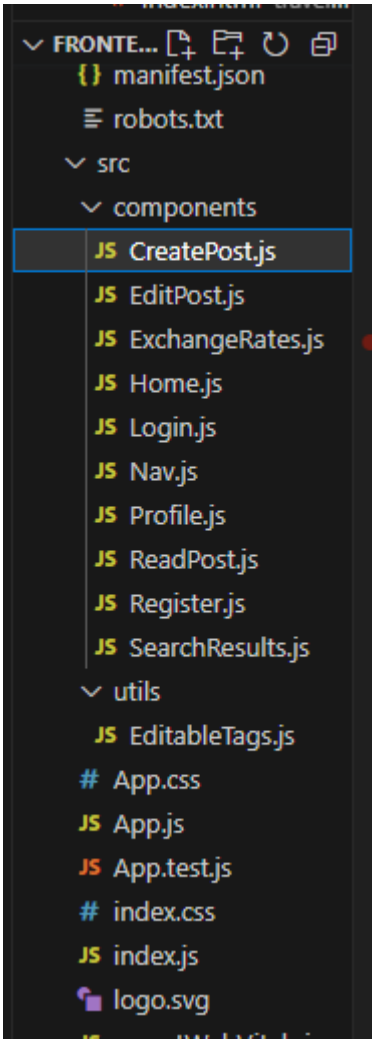
update_ideas accepts largely the same input format as create_idea, only difference is that this time the client needs to provide an id.

delete_ideas accepts a primary key(idea id) as parameter and deletes the correspondent content.

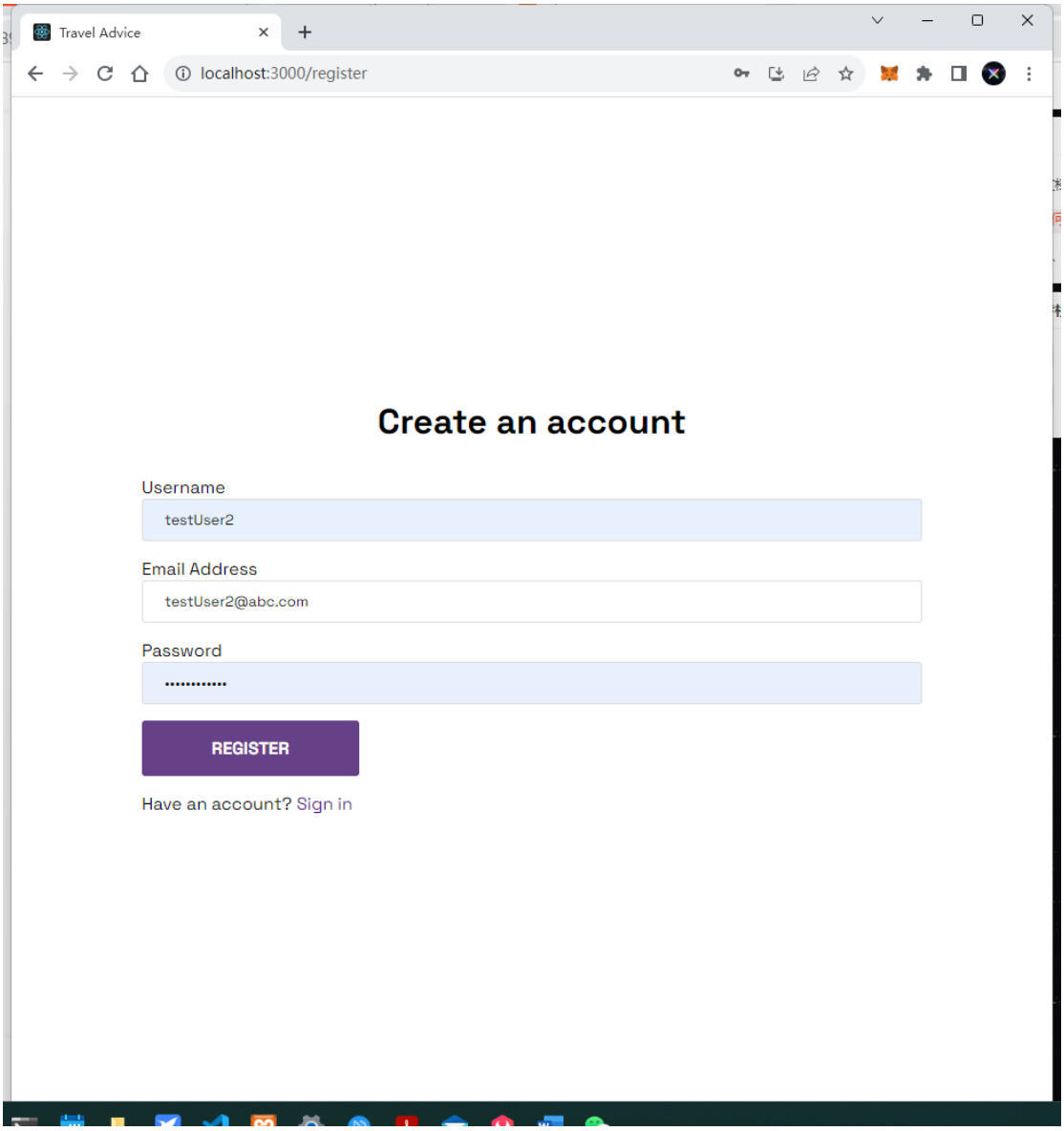
```python
@api_view(['DELETE'])
def delete_ideas(request, pk):
    """
    DELETE an idea
    api: pk/delete
    method: DELETE
    """
    idea = get_object_or_404(Idea, pk=pk)
    idea.delete()
    return Response(status=status.HTTP_202_ACCEPTED)
```

Note that only authenticated user can use these CRUD operations.

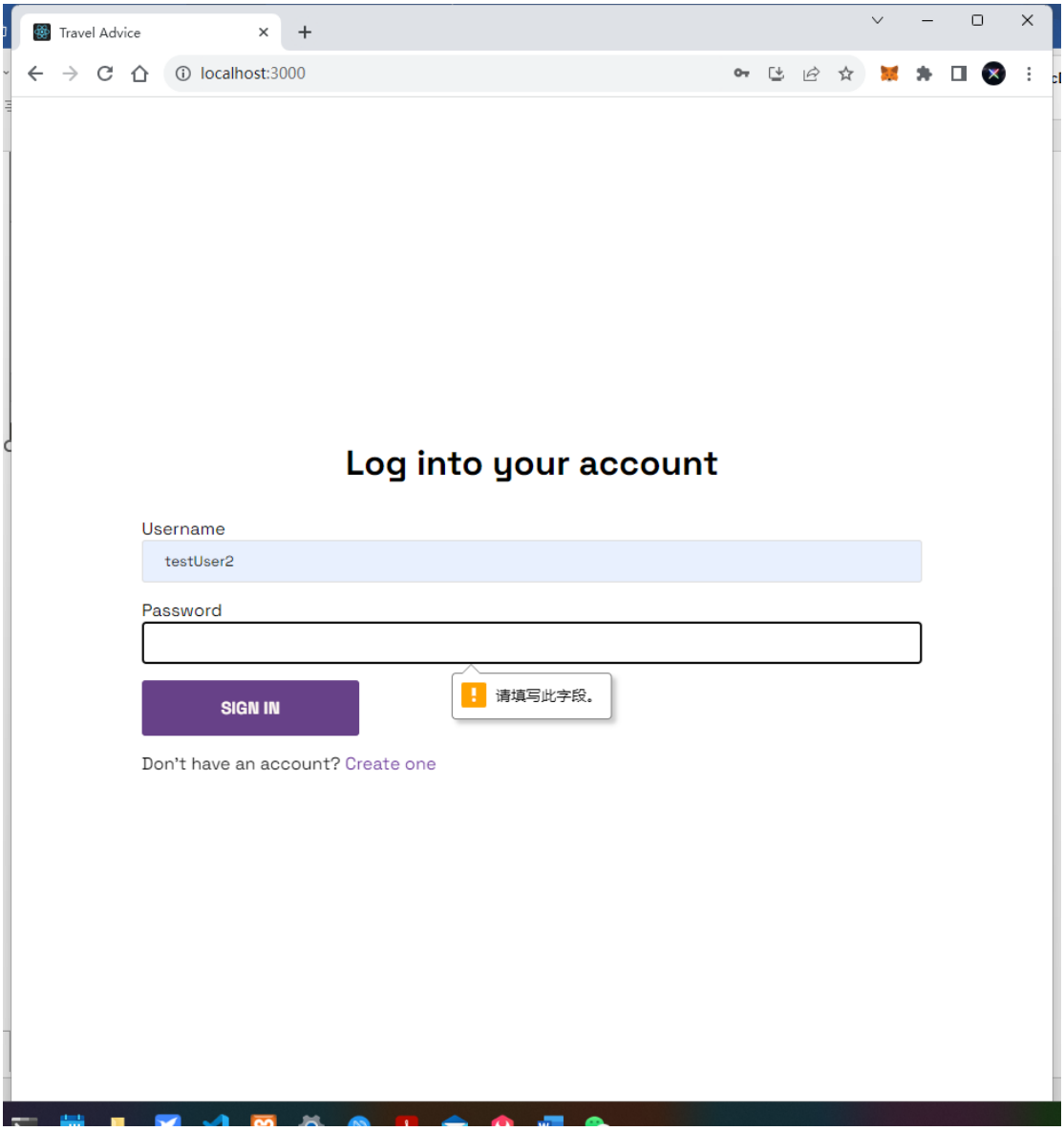
B. Frontend



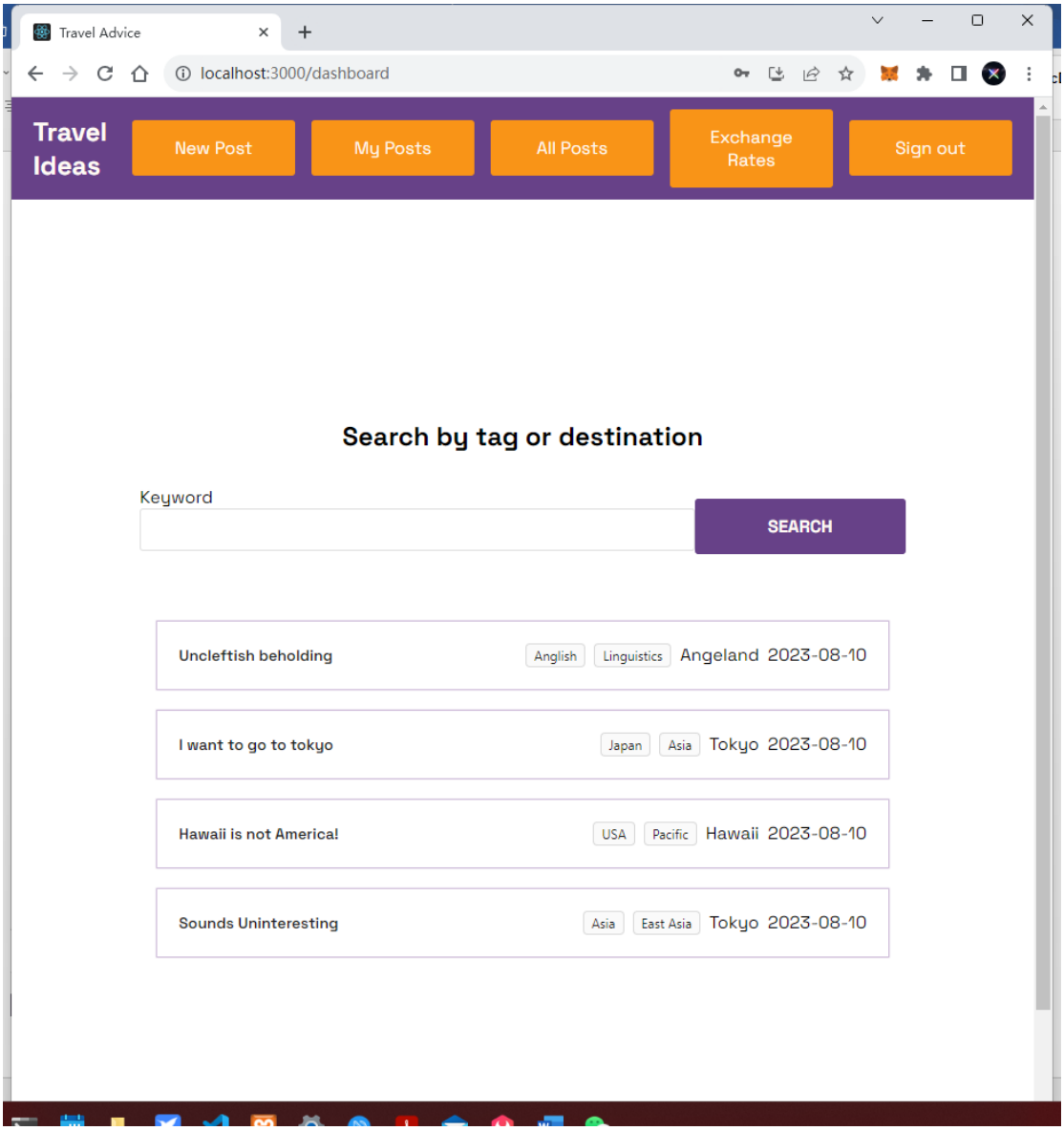
Each component under /src/components, except for Nav.js, correspond to a page. Nav is the nav bar the presents in every page after logging in.



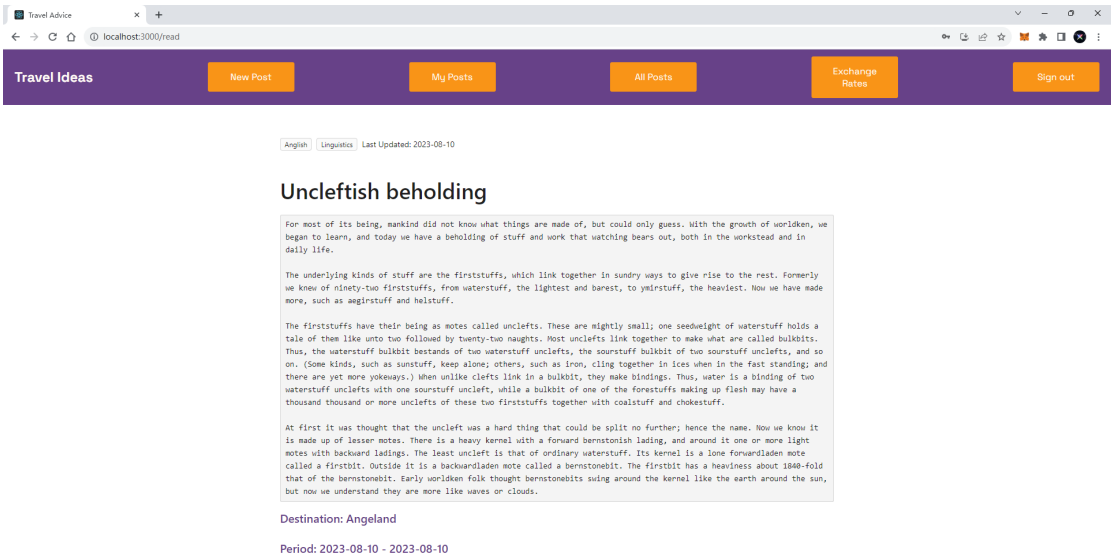
Register.js is a simple registration page. It checks for empty and illegal inputs.



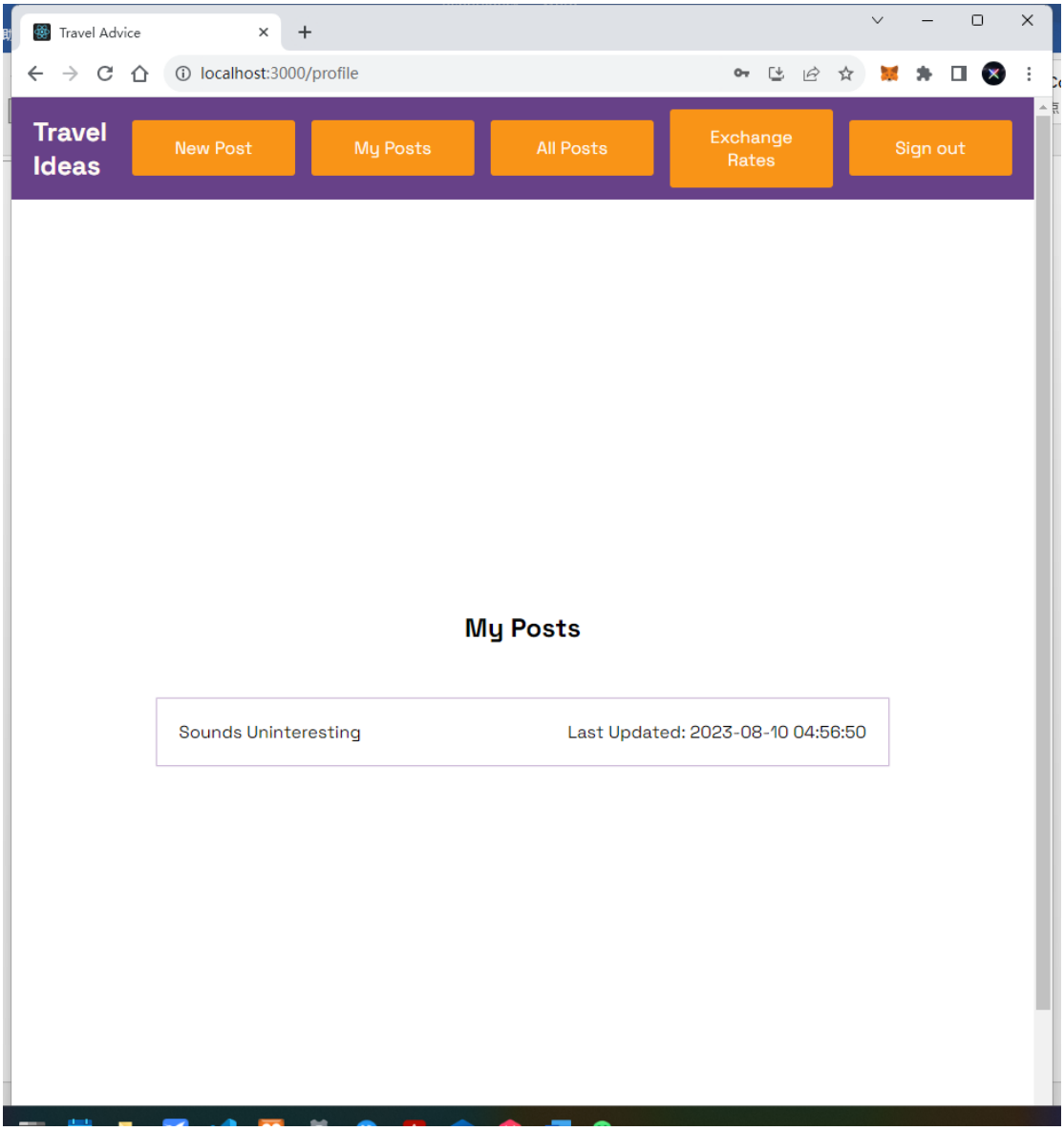
Login.js is largely the same in design. It receive a user token from calling "/ideas/rest-auth/login", stores the token at local storage and use it for authentication later.



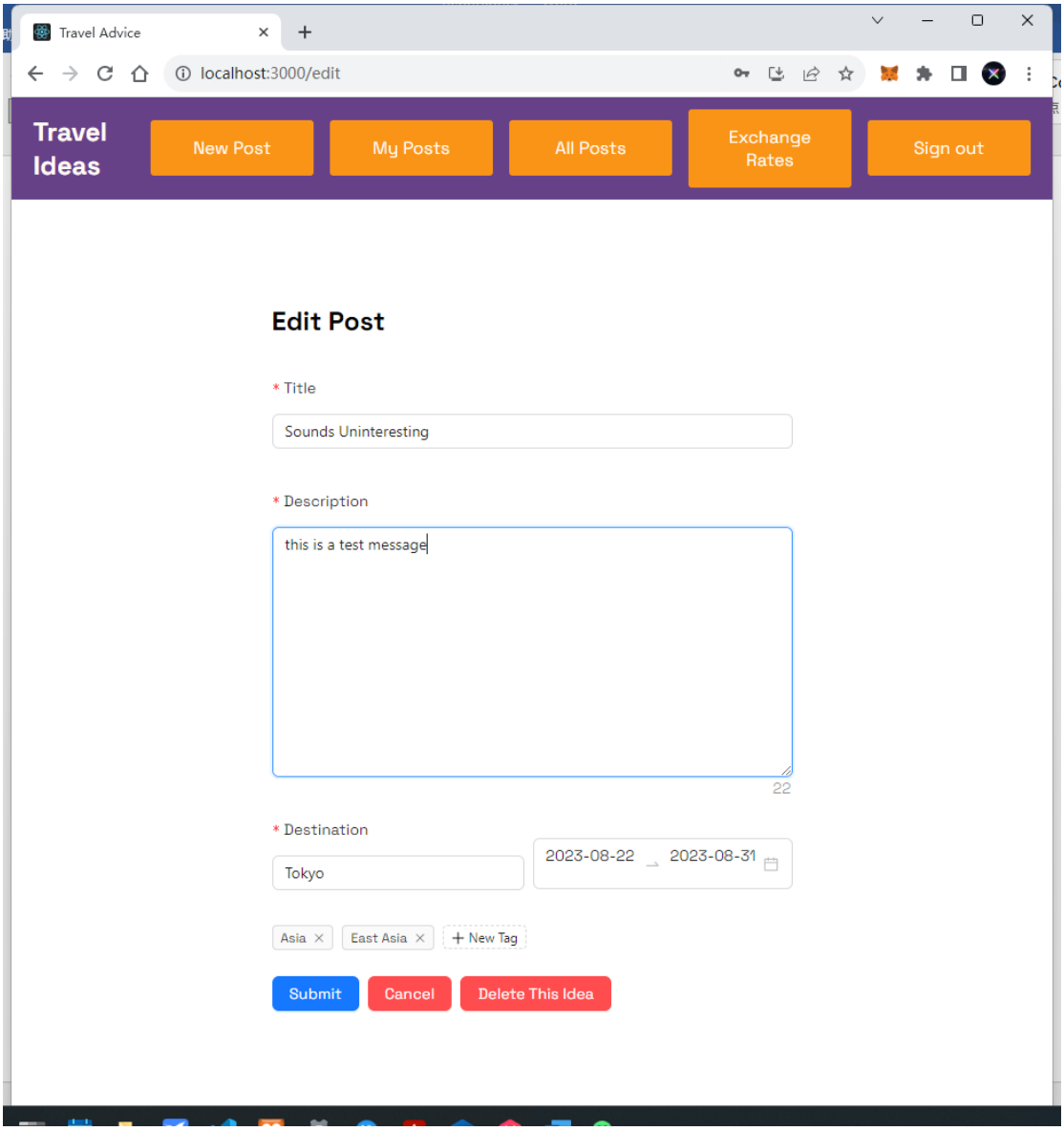
Home.js, rendered either by logging in or clicking “All Posts”



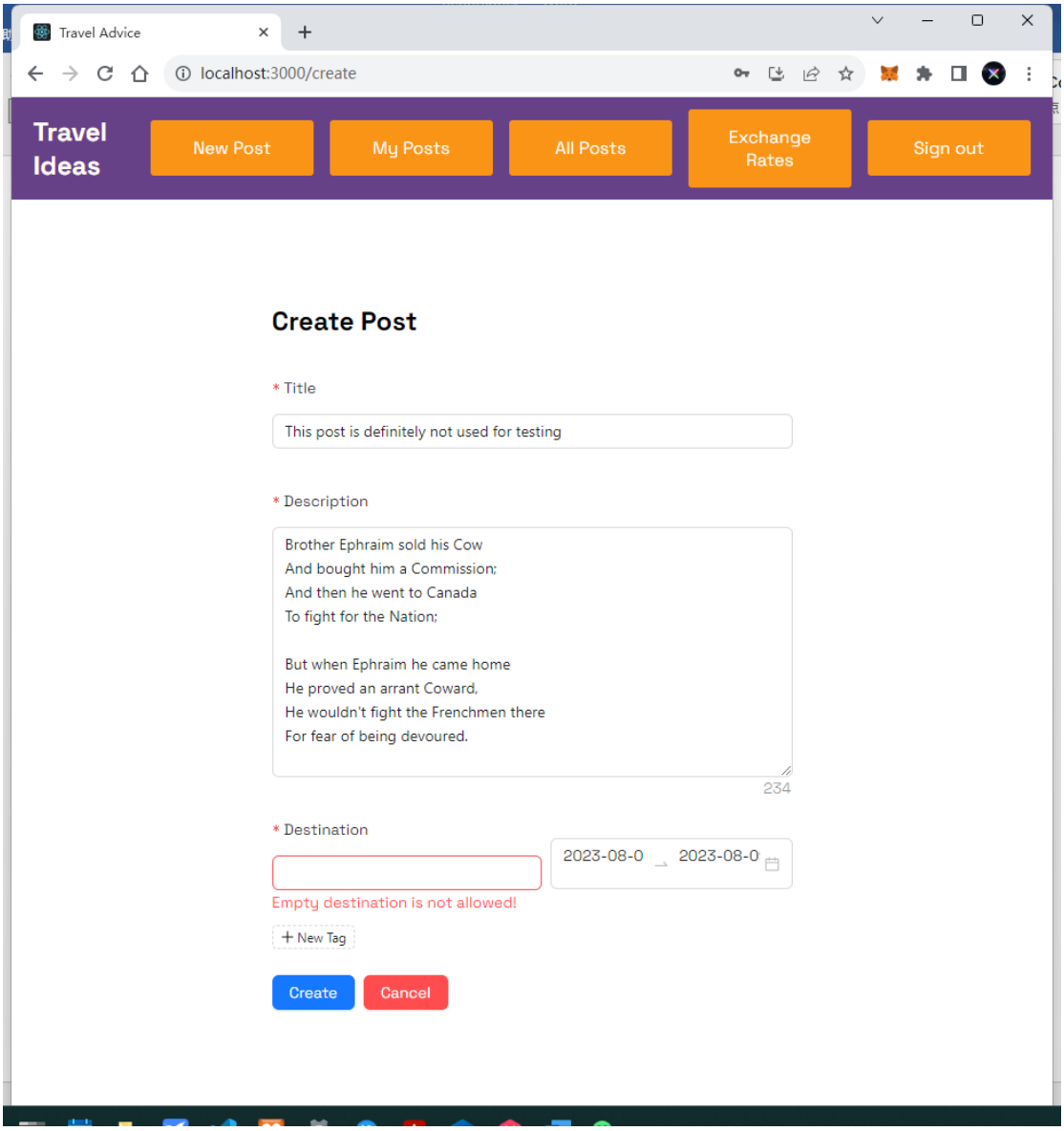
ReadPost.js, entered by clicking on idea items at dashboard or search result



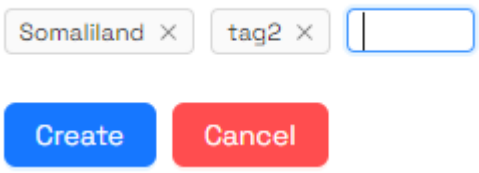
Profile.js, displays the posts authored by the current user.



EditPost.js. Tags are a list of EditableTags implemented in utils/EditableTags.js. Datepicker is always a range. All three buttons can send the user back to the profile page.



CreatePosts.js. Default date is the current day.



EditableTags.js

Search by tag or destination

Keyword

yo

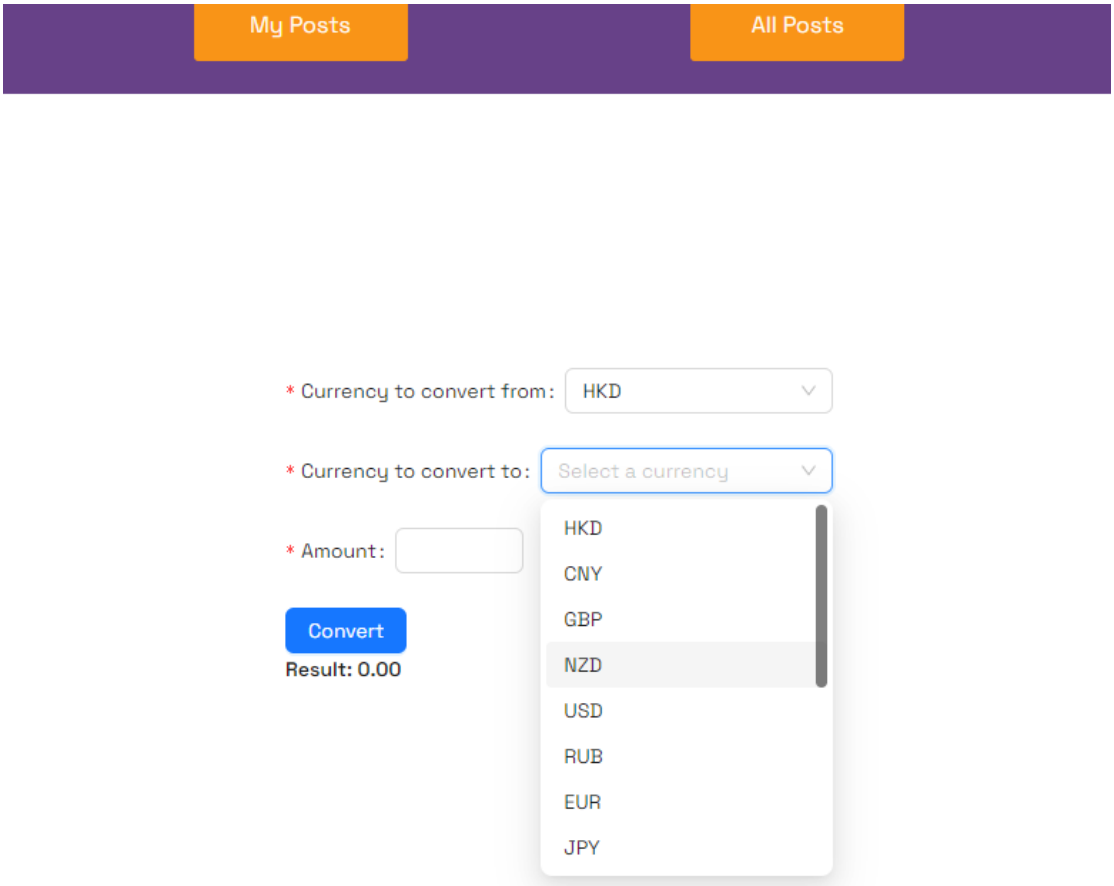
SEARCH

Uncleftish beholding	Anglish	Linguistics	Angeland	2023-08-10
I want to go to tokyo	Japan	Asia	Tokyo	2023-08-10
Hawaii is not America!	USA	Pacific	Hawaii	2023-08-10
This post is definitely not used for testing	tag2	Somaliland	Tokyo	2023-08-10
Korea is the best country in the world	Japan	DPRK	Kyoto	2023-08-10

3 results found

I want to go to tokyo	Japan	Asia	Tokyo	2023-08-10
This post is definitely not used for testing	tag2	Somaliland	Tokyo	2023-08-10
Korea is the best country in the world	Japan	DPRK	Kyoto	2023-08-10

SearchResults.js



ExchangeRates.js

III. How to run project

After setting up the DB using the provided SQL dump, open travel_advice\backend\travel_advice\travel_advice\settings.py and modify DATABASES setting to match your own.

Under travel_advice\backend\travel_advice:

pip3 install -r requirements.txt (recommend using a virtual environment)

python3 manage.py runserver

The server runs at localhost:8000 by default, go to localhost:8000/admin for admin page access. The superuser credentials are:

Username: cl980820

Password: 980820

You may create your own superuser.

Under travel_advice\backend\travel_advice, run **npm install**, and then **npm start** to launch the frontend at localhost:3000.