ft_transcendance

Overall Rules/ questions

0- This project is about creating a website for the Pong contest!

1- can we do without nginx ?

2- Project requirements , read them, understand them

3- contemplate the design of your application with your choices before delving into the code

4- Each part of the subject will explicitly present the authorized third party software you can use.

Specific Goals about the website

1- users will play Pong with others

2- nice user interface

3- check the framework module for a choice of the backend

4- respect the Database module requirements fro the choice of the database to be used with the backend

5- Frontend must respect rules of the FrontEnd module.

6- website must be a single-page application. The user should be able to use the Back and Forward buttons of the browser.

7- website must be compatible with the latest stable up-to-date version of Google Chrome

8- The user should encounter no unhandled errors and no warnings when browsing the website

9- Everything must be launched with a single command line to run an autonomous container provided by Docker . Example : docker-compose up --build

When your computers in clusters run under Linux, you will use Docker in rootless mode for security reasons. This comes with 2 sideways:

- Your Docker runtime files must be located in /goinfre or

/sgoinfre.

- You can’t use so called “bind-mount volumes” between the host and the container if non-root UIDs are used in the container.

Depending on the project, your situation and the context, several fallbacks exist: Docker in a VM, rebuild you container after your changes, craft your own docker image with root as unique UID.

About the Game

1- Both players will use the same keyboard.

The Remote players module can enhance this functionality with remote players.

2- A player must be able to **propose a tournament.** This tournament will consist of multiple players who can take turns playing against each other. You have flexibility in how you implement the tournament, but it must clearly display who is playing against whom and the order of the players.

3- **A registration system** is required: at the start of a tournament, each player must input their alias name. The aliases will be reset when a new tournament begins. However, this requirement can be modified using the **Standard User Management module.**

4- There must be a **matchmaking system:** the tournament system organize the matchmaking of the participants, and announce the next fight.

5- All players must adhere to the same rules, which includes having identical paddle speed.

6- The game itself must be developed in accordance with the default frontend constraints (as outlined above), or you may choose to utilize the FrontEnd module, or you have the option to override it with the Graphics module.

7- While the visual aesthetics can vary, it must still capture the essence of the original Pong (1972).

About Security

1- Any password stored in your database, if applicable, must be strong, and hashed.

2- Your website must be protected against SQL injections/XSS.

3- If you have a backend or any other features, it is mandatory to enable an HTTPS connection for all aspects (Utilize wss instead of ws...).

4- You must implement some form of validation for forms and any user input, either within the base page if no backend is used or on the server side if a backend is employed.

5- Any credentials, API keys, env variables etc... must be saved locally in a .env file and ignored by git.

Modules: min of 7 major mod.

Overview of the 7 chosen:

Each part of the subject will explicitly present the authorized third party software that can be used

Web

- 1- Major module: Use a Framework as backend.
- 2 - Minor module: Use a front-end framework or toolkit.
- 2 - Minor module: Use a database for the backend.

User Management

- 3- Major module: Standard user management, authentication, users across tournaments.

Gameplay and user experience

- 4- Major module: Remote players
- ?Major module: Live chat.

Cybersecurity

◦ 5- Major module: Implement Two-Factor Authentication (2FA) and JWT.

Devops

◦Major module: Infrastructure Setup for Log Management.

◦Minor module: Monitoring system.

◦6- Major module: Designing the Backend as Microservices.

◦Major module: Infrastructure Setup for Log Management.

◦Minor module: Monitoring system.

◦Major module: Designing the Backend as Microservices.

Graphics

◦?Major module: Use of advanced 3D techniques.

Accessibility

◦7- Minor module: Support on all devices.

◦7- Minor module: Multiple language supports.

-1 - Backend is Django

-2 - The designated database for all DB instances in your project is PostgreSQL .

-6 - Major module: Designing the Backend as Microservices.

- This major module aims to enhance the system’s architecture by adopting a microservices design approach, enabling greater flexibility, scalability, and maintainability of the backend components.

Key features and objectives include:

- Divide the backend into smaller, loosely-coupled microservices, each responsible for specific functions or features.
- Define clear boundaries and interfaces between microservices to enable independent development, deployment, and scaling.
- Implement communication mechanisms between microservices, such as REST-ful APIs or message queues, to facilitate data exchange and coordination.
- Ensure that each microservice is responsible for a single, well-defined task or business capability, promoting maintainability and scalability.

Likely steps for the 3 to work together:

Step 1- Install Django and the Django REST Framework.

To start developing your Django-based microservices for the Pong game, you'll first need to install Django and the Django REST Framework.

▼ Step a: Create and Activate a Virtual Environment

It's a good practice to create a virtual environment for your Python projects. This keeps dependencies required by different projects separate by creating isolated environments for them. Open your terminal or command prompt and run the following commands:

```
# Install virtualenv if you don't have it
pip install virtualenv

# Create a virtual environment named 'venv' (or any othe
r name you prefer)
virtualenv venv

# Activate the virtual environment
# On Windows
venv\\Scripts\\activate
# On Unix or MacOS
source venv/bin/activate
```

▼ Step b: Install Django and Django REST Framework

With the virtual environment activated, install Django and Django REST Framework using pip:

```
pip install django djangorestframework
```

▼ Step c: Start a New Django Project

Create a new Django project. This project will act as one of your microservices. In a real-world scenario, you would repeat this process for each microservice. For this example, let's create the User Service:

```
django-admin startproject user_service
cd user_service
```

▼ Step d: Create a New Django App

Within your Django project, you can create apps that handle different parts of your application logic. Since we're focusing on the User Service, let's create an app that handles user-related operations:

```
python manage.py startapp accounts
```

▼ Step e: Set Up Django REST Framework

To set up the Django REST Framework in your project, you need to add it to the `INSTALLED_APPS` in your project's `settings.py` file:

```
INSTALLED_APPS = [
    ...
    'rest_framework',
    'accounts',  # Make sure to add your newly created a
pp here
]
```

▼ Step f: Create Your First Model and Serializer

Let's create a simple `User` model in `accounts/models.py` (Django already comes with a built-in User model, but this is just for example):

```
from django.db import models

class User(models.Model):
    username = models.CharField(max_length=100)
    email = models.EmailField()

    def __str__(self):
        return self.username
```

Then, create a serializer for the User model in a new file `accounts/serializers.py`:

```
from rest_framework import serializers
from .models import User

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['id', 'username', 'email']
```

▼ Step g: Create a View and URL Route

Create a view in `accounts/views.py` to handle API requests:

```
from rest_framework import viewsets
from .models import User
from .serializers import UserSerializer

class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer
```

Then, define a URL route in `user_service/urls.py`:

```
from django.urls import include, path
from rest_framework import routers
from accounts.views import UserViewSet

router = routers.DefaultRouter()
router.register(r'users', UserViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

▼ Step h: Run Your Server

Finally, run your Django server:

```
python manage.py runserver
```

Now, your User Service is set up with a basic API endpoint to create, read, update, and delete user instances. Access `http://127.0.0.1:8000/users/` to see the API in action.

Repeat similar steps to set up other microservices for your project, adjusting the models, views, and URLs as per the specific requirements of each service (e.g., Game Service, Matchmaking Service, Leaderboard Service).

Step 2- Install and Configure PostgreSQL

Step 3- Apply Migrations

Step 4- Designing Microservices:

Step 5- Microservices Communication:

Step 6- Role of Django in the Gaming:

WebSocket for real-time communication between the client and server to update game states dynamically.

Django Channels is an excellent extension for working with WebSockets in Django.

Step 7- Deployment Considerations

Step 8- **Continuous Integration and Continuous Deployment (CI/CD):** Set up a pipeline to automate the testing and deployment process, ensuring that each update to the codebase is validated and deployed efficiently.