Documentación prueba Desarrollador Panzofi

William Florez

2024-2

Introducción

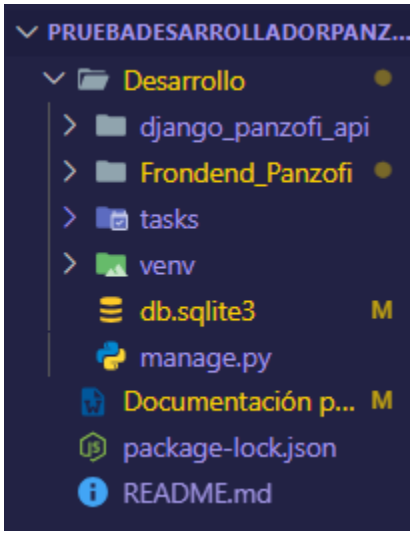
El actual documento se destina a la documentación de las tecnologías, procesos y funciones de la aplicación web desarrollada como prueba técnica, las tecnologías utilizadas fueron REACT que funciona como un framework para facilitar el desarrollo de frondend y DJANGO para la gestión del Backend.

Programas:

- DJANGO:
- REACT:

Estructura del proyecto

En este apartado se explorará la función de los archivos y como se ha distribuido para una navegación sencilla.



La carpeta desarrollo es la carpeta que almacena todo el proyecto. La carpeta “django_panzofi_api” se destina a albergar la lógica interna de django para diferenciar documentos de REACT y DJANGO, además que se creó el archivo .sqlite3 el cual funciona como base de datos del proyecto. La carpeta “Frondend_Panzofi” alberga todo lo relacionado con el frondend o la vista del usuario, La carpeta venv fue creada para funcionar como un entorno virtual para la implementación del proyecto, la base de datos(backend) se inicializa usando el comando:

```
python manage.py runserver
```

Además que para iniciar el entorno virtual se debe ejecutar el comando

```
./venv/Script/activate
```

La carpeta “Task” se creó para la comunicación entre el frondend y el backend. A continuación, se explicarán las carpetas y sus documentos que son utilizados además de explicar su funcionamiento.

BACKEND / DJANGO

Archivo Setting.py aquí se da la autorización para comunicarse con la url de REACT asi las peticiones del frondend pueden ser respondidas por el backend además que se define los programas que son instalados y las carpetas que será utilizadas:

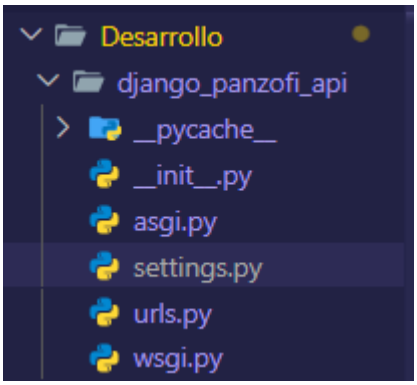


Ilustración 1CarpetaDango

```python
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders', # para comunicarse con el backend
    'rest_framework', # creación de API
    'tasks'     # Carpeta con los archivos frondend y logicas
]
```

Ilustración 2Django/Settings programas instalados

```
# CORS authorization
# URL del servidor REACT
CORS_ALLOWED_ORIGINS = ["http://localhost:5173"]
```

Ilustración 3Settings rutas de solicitudes permitidas

El programa “URL.py” de la carpeta de DJANGO es donde se define unafracción de las rutas url a las cuales respondería y la carpeta de donde puede complementar la url. En la siguiente foto se muestra que responderá a diferentes url, lo que significa es lo siguiente:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('tasks/', include('tasks.urls')),
    path('PruebaTec/', include('tasks.urls'))
]
```

Ilustración 4Django Rutas

ruta	Significado
path('admin/', admin.site.urls)	Se redirige a la página administradora de DJANGO
path('tasks/', include('tasks.urls'))	Usado esta preposición en la url conecta con las extensiones de URL definidas en la carpeta TASK
path('PruebaTec/', include('tasks.urls')	Similar a task/ pero la diferencia esta en la nomenclatura, es mas una diferencia estetica

FRONDEND / REACT

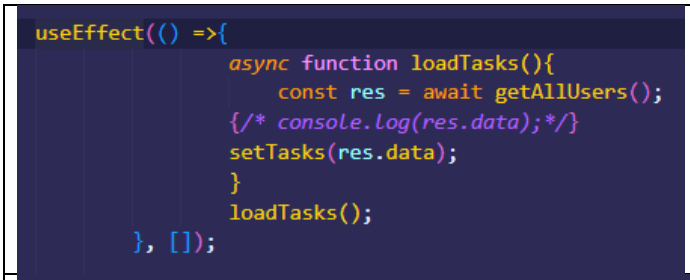
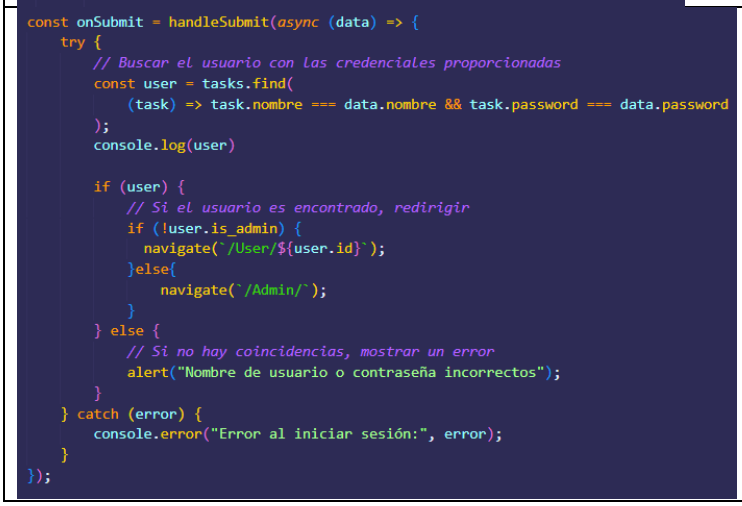
En REACT la lógica e interfaz se almaceno en una carpeta llamada “SRC”, aquí están las carpetas de:

- API: aquí se alberga los documentos encargados de realizar las solicitudes GET, PATCH, DELETE en la base de datos, todo partiendo de una misma url que combina la extensión definida en el archivo url de DJANGO y las que se definen en REACT.
- COMPONENTS: son fragmentos de código que cumplen una función especifica para ser implementadas en las paginas principales, además que estas pueden ser reutilizadas en caso de ser necesario.
- PAGES: carpeta donde se almacenan las interfaces de los usuarios admin, user y login.

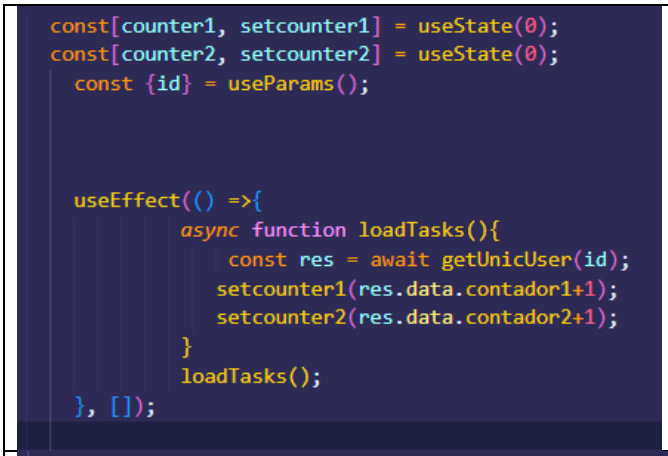
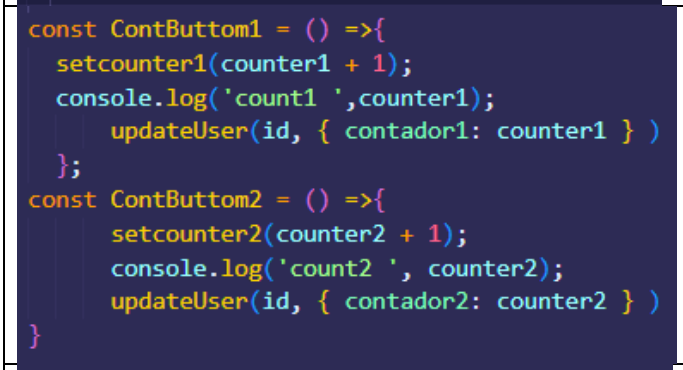
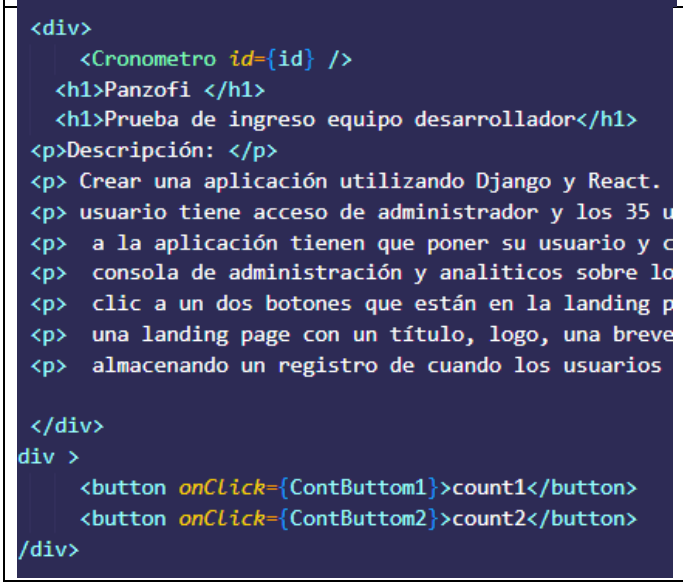
- APP.jsx: este documento se destina también a la navegación para complementar los archivos de url pues si el archivo no se encontrara aquí no se encontraría el archivo al que se hace referencia.

CARPETA PAGES:

Login.jsx: solicita y valida la existencia de usuarios además del tipo, dependiendo el caso se envia a la pagina de usuario o de administrador. Implementa funciones de la carpeta API para verificar lo mencionado anteriormente

	Se hace la solicitud de los datos y se almacena en una variable
	Función que busca la existencia del usuario suministrado y redirige a otras paginas, encaso de ser un usuario corriente se adjunta la ID del usuario para ser usado en la pagina de usuario

UserPAge.jsx: la pagina de usuario posee la id del usuario almacenada en la url para consultar los valores de contador de los botones, también es usado para señalar donde se debe de almacenar los datos de la fecha y tiempo de permanencia en la página.

	Se definen las variables que almacenan los valores previos de los contadores y se recupera el valore de id que se encuentra en la url
	Lógica usada para actualizar el valore del numero de veces que se oprimieron los botones usando la api “updateUser”
	Se invoca el el componente que funciona como cronometro que se alverga en la carpeta COMPONENTS, y botones que ejecuta la función de contadores

AdminPAge.jsx: esta pagina es destinada únicamente para usuarios administrador, aquí se muestra el código almacenado en “COMPONENTS” donde se lista los usuarios y los datos de nombre, fecha, tiempo, contador1, contador2 además de 3 tipos de graficas que representan los contadores.

```jsx
import { TaskList } from "../components/TaskList";
import { Graphics } from "../components/Graphics";  // Importamo

/* se instala la biblioteca para grafica
npm install react-chartjs-2 chart.js
*/
export function Admin(){
    return    (
        <div>
          <h1>Admin Page</h1>
          <TaskList /> {/* lista de ususarios*/}
          <Graphics /> {/* grafica con valores de botones*/}
        </div>
      );
    }
```

CARPETA COMPONENTS:

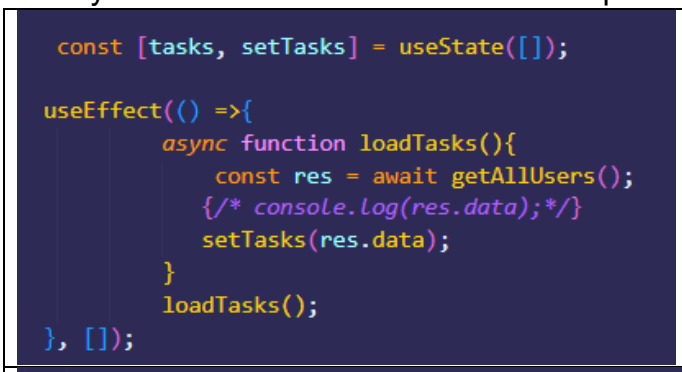
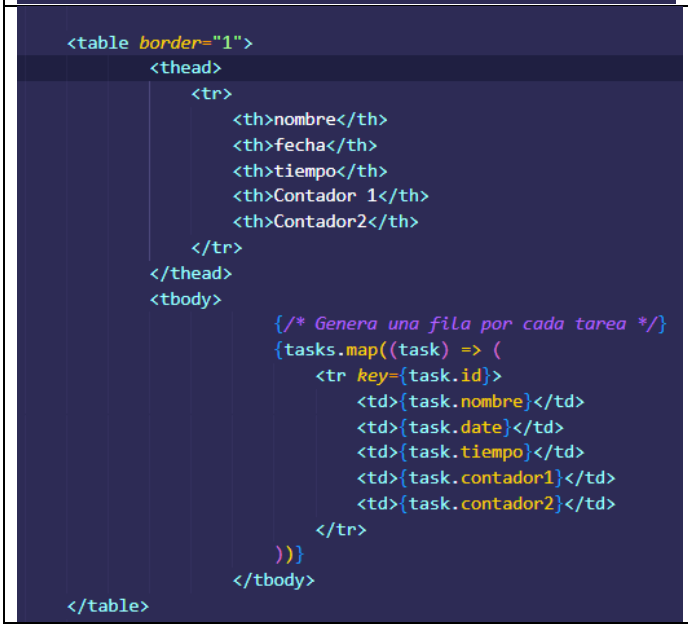
La carpeta components busca almacenar procesos que se pueden reutilizar en otros programas:

- Cronometro.jsx: programa que inicia un contador desde que carga la pagina hasta que se cierra, ademas que se obtiene la fecha en que se ingresa y se reutiliza el api de actualización de datos, en este se utiliza una variable enviada desde la pagina donde es usada, y señala el usuario que se debe modificar.
- Graphics.jsx: recopila los datos de la base de datos y se implementan en funciones para manipular los datos y poder ser graficados

``jsx const chartData = { labels: data.map(user => user.nombre), // nombre de usuarios datasets: [// información que se va a usar { label: "Contador 1", data: data.map(user => user.contador1), // columa contador 1 borderColor: "rgba(75,192,192,1)", backgroundColor: "rgba(75,192,192,0.2)" }, { label: "Contador 2", data: data.map(user => user.contador2), // columa contador 2 borderColor: "rgba(153,102,255,1)", backgroundColor: "rgba(153,102,255,0.2)" },], }; ```	Manipulación de la información de la base de datos para ser graficada

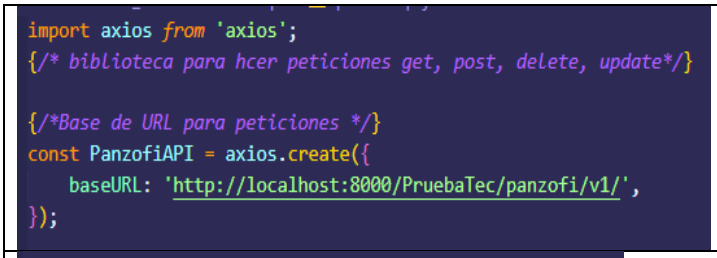
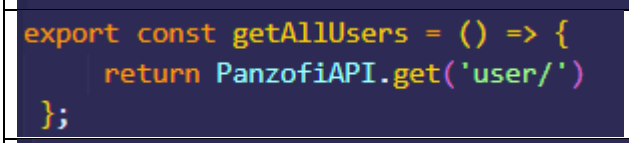
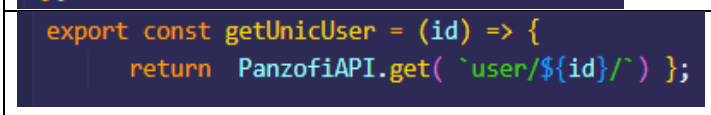
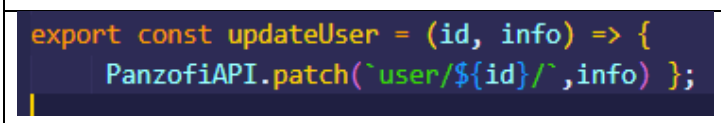
	Tipos de grafica que usan datos especificados anteriormente

- Navigation.jsx: ayuda para la navegación, entre páginas, puede ser eliminada puesto que esta fue echa para pruebas de navegación.
- TaskList.jsx: reutiliza el api para obtener todos los usuarios de la base de datos y se listan en una tabla los datos requeridos

	Api para obtener todos los usuarios
	Tabla que muestra los datos seleccionados

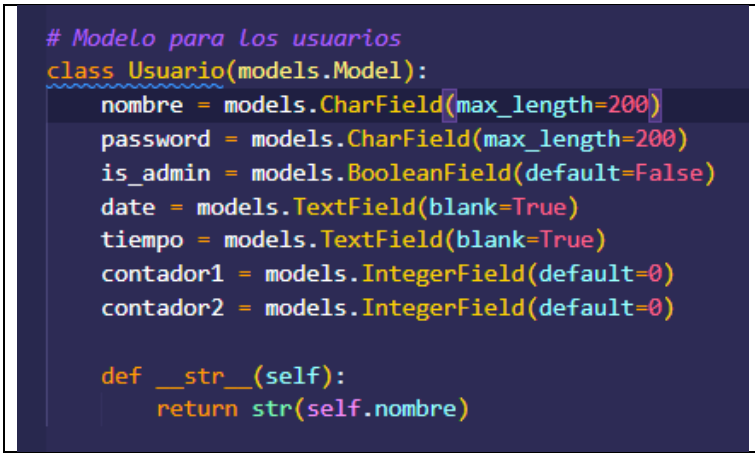
CARPETA API

Se almacenan la estructura de las solicitudes a la base de datos, serán usadas y reutilizadas en los diferentes programas.

	Dirección base de la URL para peticiones de la base de datos
	Obtener todos los datos
	Petición similar a la anterior pero aquí se pide un valor para buscar
	Formato que pide dos variables un id para ubicar al usuario e info que es la información que se va a actualizar

CARPETA TASK

La carpeta Task es usada principalmente para la comunicación entre el frondend y backend al crear las tablas de la base de datos, establecer las bases de las URL, convertir los datos de REACT a JSON para que puedan ser interpretados y almacenados por la base de datos de DJANGO.

	Archivo models.py es el encargado de crear la estructura que almacena la información de los usuarios

- URL.PY este es el archivo del cual se hace referencia en el documento de url de DJANGO,aquí es donde se completa la url con la información de

REACT.

urls.py

Desarrollo > tasks > urls.py > ...

```python
# creación de las rutas necesarias para el frondend
from django.urls import path, include
from rest_framework import routers
from tasks import views

#api version
router = routers.DefaultRouter()
router.register(r'tasks', views.TaskView, 'tasks')
router.register(r'user', views.UserView, 'users')

urlpatterns = [
    path("api/v1/", include(router.urls) ),
    path("panzofi/v1/", include(router.urls)),

]
```

- Serializer.py: documento que transforma la información generada por REACT a formato JSON para ser interpretado por DJANGO

serializer.py

Desarrollo > tasks > serializer.py > ...

```python
#"""archivo para convertir datos django a json"""

from rest_framework import serializers
from .models import Task, Usuario

class TaskSerializer(serializers.ModelSerializer):
    class Meta:
        model = Task
        #fields = ('id', 'title', 'description', 'done')
        fields = '__all__' # para serializar todos los campos


class TaskSerializerUser(serializers.ModelSerializer):
    class Meta:
        model = Usuario
        fields = '__all__' # para serializar todos los campos
```