P2Pibes

Unha solución descentralizada á compartición de ficheiros.

Pedro Guijas Bravo
Arturo Ramos Rey
Eliseo Bao Souto
Manuel Corujo Muiña
Diego Fresco Rivera

Contidos

1

Proxecto

Requisitos funcionais e non funcionais.

2

Arquitectura

Representación C4 e tácticas aplicadas..

3

Implementación

Estrutura, elementos destacados, probas e documentación.

4

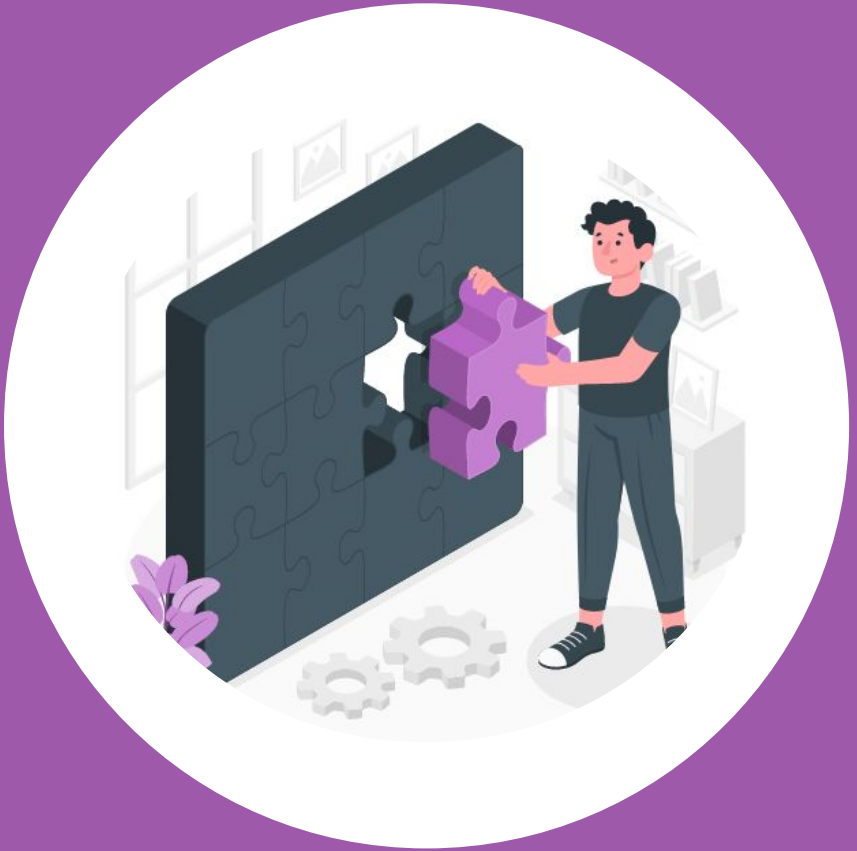
Demostración

Consideracións previas...

- Proxecto adaptado ao tempo da práctica
- Entorno local, non distribuido real
    - Todos os nodo en rede e accesibles.
- Pouca seguridade para un caso real

1

Proxecto



Que é P2Pibes?

1

Compartición de ficheiros

Distribuir e prover acceso a información almacenada dixitalmente.

2

Descentralización

Dispersar poderes e responsabilidades dunha entidade única central.

3

P2P

Rede interconectada na que desaparecen os conceptos cliente/servidor clásico e aparecen os nodos.

Requisitos funcionais e non funcionais

1. **Engadido de Nodos**
2. **Subida de ficheiros**
3. **Borrado de ficheiros**
4. **Busca de ficheiros**
5. **Descarga de ficheiros**
6. **Consulta de información**

1. **Rendemento**

   Velocidade ao procesar peticións.

2. Escalabilidade

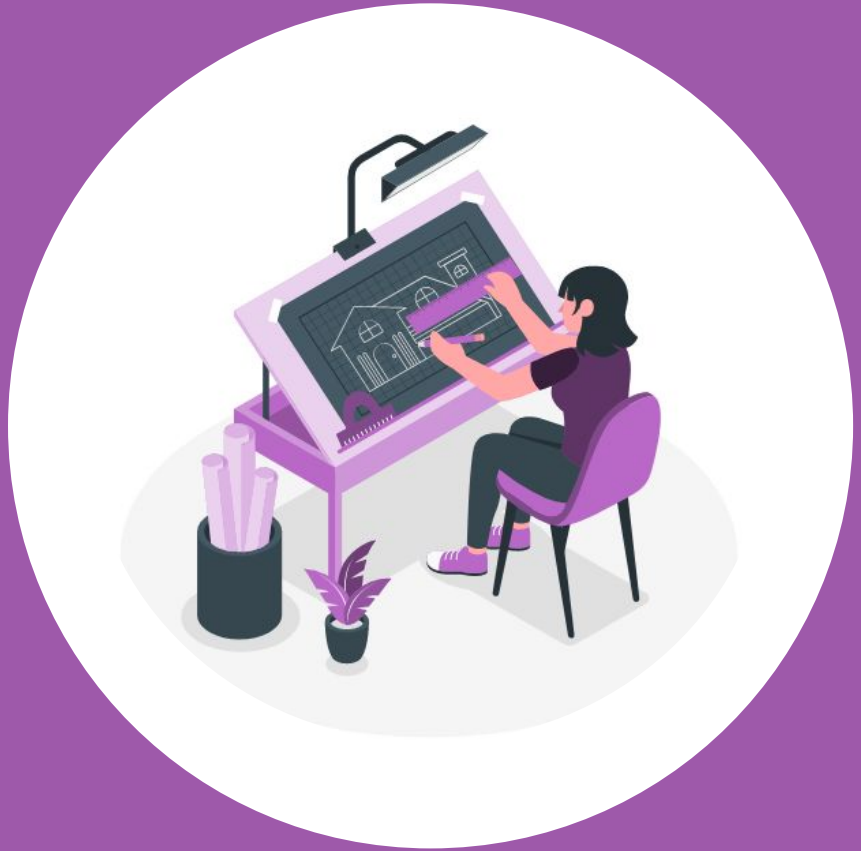
   Posibilidade de gran expansión.

3. Balanceo

   Reparto da carga de traballo.

2

Arquitectura

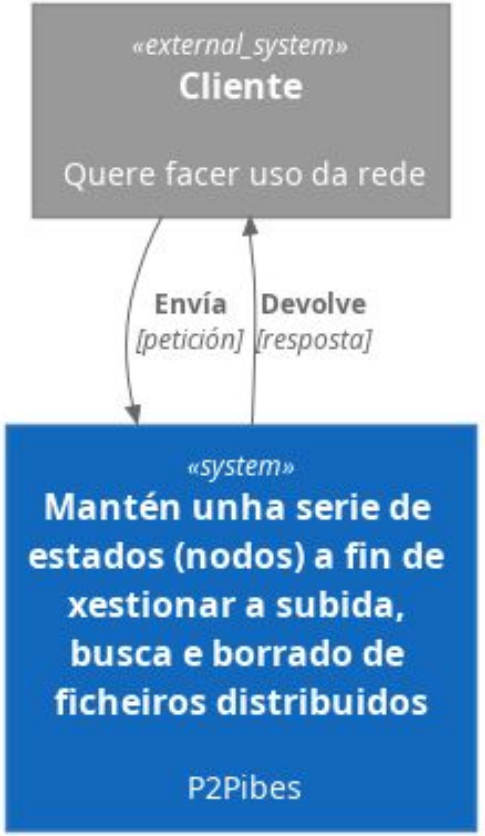


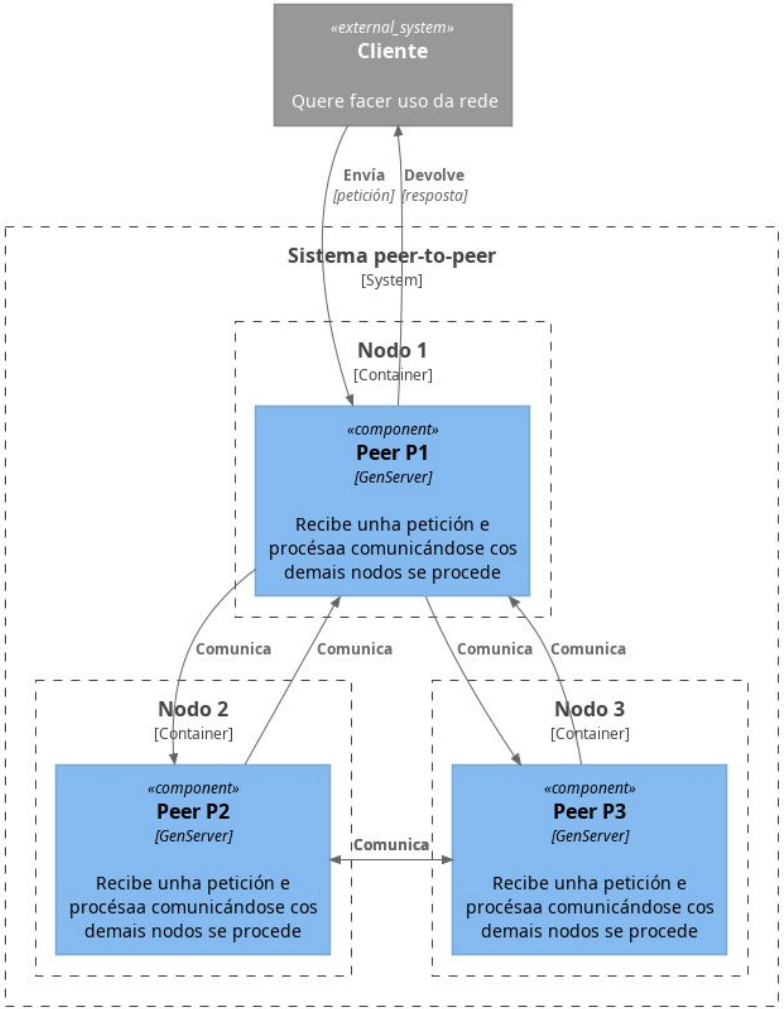
Representación

C4

C4: Nivel 1

Contexto



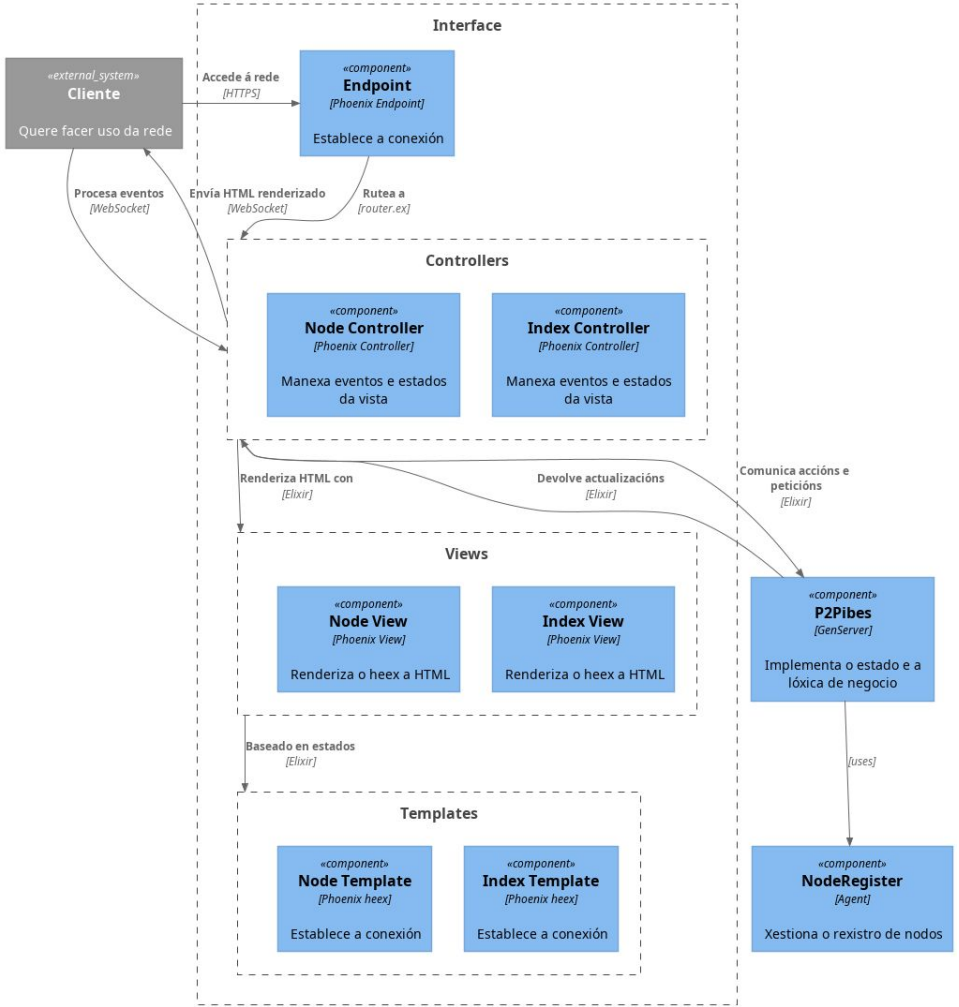


C4: Nivel 2

Contedor

C4: Nivel 3

Compoñentes



Tácticas aplicadas de cara ao rendemento

1

TTL

Tempo de vida asignado para limitar a duración máxima dunha petición.

2

Timeouts

Timeouts en GenServer para limitar accións demasiado duradeiras

3

Caché

Gardado de datos recentes para poder responder máis rapidamente.

Só internas, non compartidas para non obter resultados desactualizados.

4

Async

Procesamento asíncrono para evitar esperas activas.

5

Rexistro

Dada a interconexión da rede, resulta importante evitar posibles ciclos.

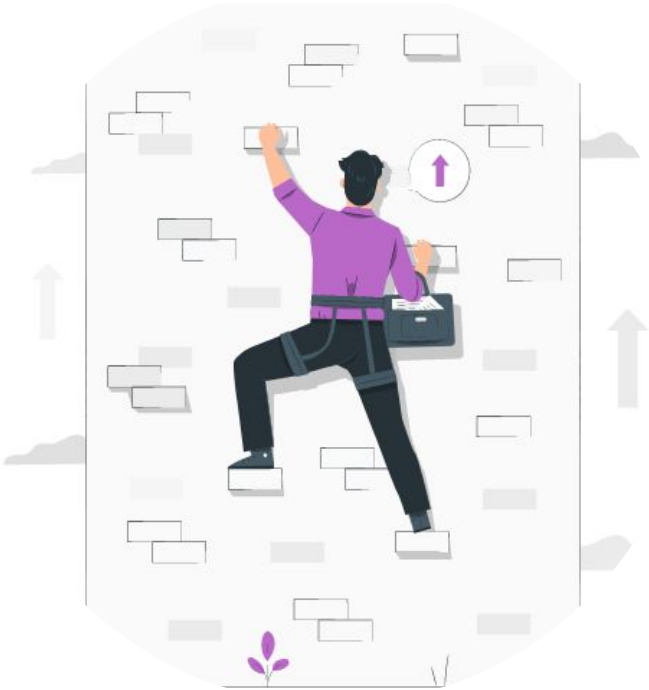
3

Implementación

Estrutura

Proxecto Phoenix con estrutura complexa.

- C4 (diagramas e código)
- lib
  - p2pibes_web
    - controllers
    - templates
    - views
    - router.ex
  - p2pibes.ex
  - node_register.ex
- test
- README.md



Elementos destacados

Neste proxecto, faise uso de elementos destacados (e ben coñecidos) de Elixir como son:

- GenServer
- Agent

Vexámolos!

Agent

Útil para o manexo do rexistro de nodos.

Módulo `NodeRegister`

```elixir
defmodule NodeRegister do
  use Agent

  def start_link do
    Agent.start_link(fn -> [] end, name: :node_reg)
  end

  def get_nodes do
    if !Enum.member?(Process.registered(), :node_reg) do
      start_link()
    end
    Agent.get(:node_reg, fn content -> content end)
  end

  def exist?(node) do
    if !Enum.member?(Process.registered(), :node_reg) do
      start_link()
    end
    Agent.get(:node_reg, fn content -> content end)
    |> Enum.member?(node)
  end

  def add_node(node) do
    if !Enum.member?(Process.registered(), :node_reg) do
      start_link()
    end
    Agent.update(:node_reg, fn list -> [node | list] end)
  end

  def remove_node(node) do
    if !Enum.member?(Process.registered(), :node_reg) do
      start_link()
    end
    Agent.update(:node_reg, fn list -> List.delete(list, node) end)
  end
end
```

GenServer

```elixir
defmodule P2pibes do
  use GenServer

  @ttl 5
  @cache_valid_time 1800


  ################################## INTERFACE ##################################

  # Arrancar Nodo en Red vacia
  def up_node(name, files) do
    {:ok, pid} = GenServer.start_link(__MODULE__, files, name: name)
    NodeRegister.add_node(name)
    pid
  end

  # Arrancar Nodo en Red vacia
  def up_node(name, files, node) do
    my_node = up_node(name, files)
    add_neighbor(name, node)
    add_neighbor(node, name)
    my_node
  end

  # Bajar Nodo
  def down_node(node) do
    GenServer.call(node, :exit)
    NodeRegister.remove_node(node)
    GenServer.stop(node)
  end

  # Neighbors
  def add_neighbor(node1, node2) when node1 != node2,
    do: GenServer.cast(node1, {:add_neighbor, node2})

  def rm_neighbor(node1, node2), do: GenServer.cast(node1, {:rm_neighbor, node2})
  def get_neighbors(node), do: GenServer.call(node, :get_neighbors)

  # Files
  def add_file(node, path), do: GenServer.cast(node, {:add_file, path})
  def rm_file(node, path), do: GenServer.cast(node, {:rm_file, path})
  def get_files(node), do: GenServer.call(node, :get_files)
  def get_file(node, filename), do: GenServer.call(node, {:get_file, filename})
  def download_file(node, path), do: GenServer.call(node, {:download_file, path})

  # Search
  def search(node, filename), do: GenServer.cast(node, {:search, filename})
  def get_results(node, filename), do: GenServer.call(node, {:get_results, filename})
```

```elixir
defmodule P2pibes do
  use GenServer

  @ttl 5
  @cache_valid_time 1800


################################## SERVER ##################################
  @impl true
  def init(files) do
    # Estado -> vecinos, ficheros, log_querys, resultados
    {:ok, {[], files, [], []}}
  end

  # Adds

  # Add Neighbor
  @impl true
  def handle_cast({:add_neighbor, node}, {nodes, files, q_log, results}),
    do: {:noreply, {[node | nodes], files, q_log, results}}

  # Rm Neighbor
  def handle_cast({:rm_neighbor, node}, {nodes, files, q_log, results}),
    do: {:noreply, {List.delete(nodes, node), files, q_log, results}}

  # Add File
  def handle_cast({:add_file, path}, {nodes, files, q_log, results}) do
    # file(nombre,ruta,size,hash)
    if File.exists?(path) do
      if File.dir?(path) do
        # Adds dir
        {:noreply,
         {nodes,
          (new_directory(path) ++ files)
          # Clears duplicated files
          |> Enum.uniq_by(fn x -> x[:hash_ref] end), q_log, results}}
      else
        # Comprobamos duplicados
        if Enum.any?(files, fn x -> x[:path] == new_file(path)[:path] end) do
          {:noreply, {nodes, files, q_log, results}}
        else
          # Adds file
          {:noreply, {nodes, [new_file(path) | files], q_log, results}}
        end
      end
    else
      # No existe
      {:noreply, {nodes, files, q_log, results}}
    end
  end

  ...

  end

  # Exit
  @impl true
  def handle_call(:exit, _from, {nodes, files, q_log, results}) do
    Enum.each(nodes, fn x -> GenServer.cast(x, {:rm_neighbor, self()}) end)
    {:reply, :ok, {[], files, q_log, results}}
  end

  # Get Neighbors
  @impl true
  def handle_call(:get_neighbors, _from, {nodes, files, q_log, results}),
    do: {:reply, nodes, {nodes, files, q_log, results}}

  # Get Files
  @impl true
  def handle_call(:get_files, _from, {nodes, files, q_log, results}),
    do: {:reply, files, {nodes, files, q_log, results}}

...
```

Probas realizadas

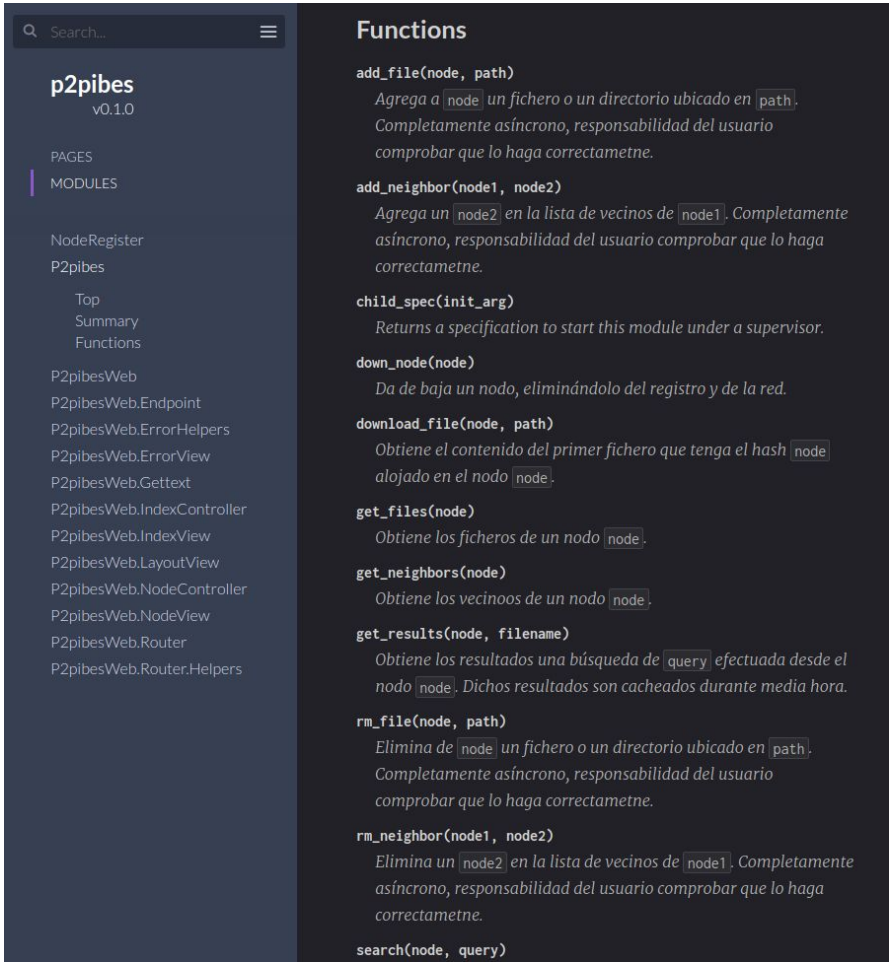
Realizáronse tanto `Doctest` coma probas de unidade “tradicionais”.

```
"""
 Prueba de Creacion de Redes:
 {:ok,node1}=P2pibes.up_node([])
 {:ok,node2}=P2pibes.up_node([],node1)
 {:ok,node3}=P2pibes.up_node([],node1)
 {:ok,node4}=P2pibes.up_node([],node1)
 {:ok,node5}=P2pibes.up_node([],node1)
 P2pibes.get_neighbors(node1)
 P2pibes.get_neighbors(node2)

 P2pibes.down_node(node1)
 P2pibes.get_neighbors(node2)
 P2pibes.get_neighbors(node5)
"""
```

```
defmodule P2pibesTest do
  use ExUnit.Case

  def create_node(files, name), do: GenServer.start_link(P2pibes, files, name: name)


  test "Creacion de nodos" do
    {:ok, node1} = create_node([], :node1)
    {:ok, node2} = create_node([], :node2)
    {:ok, node3} = create_node([], :node3)
    {:ok, node4} = create_node([], :node4)
    {:ok, node5} = create_node([], :node5)
    P2pibes.add_neighbor(node1, node2)
    P2pibes.add_neighbor(node2, node1)
    P2pibes.add_neighbor(node2, node3)
    P2pibes.add_neighbor(node3, node2)
    P2pibes.add_neighbor(node3, node4)
    P2pibes.add_neighbor(node4, node3)
    P2pibes.add_neighbor(node4, node5)
    assert P2pibes.get_neighbors(node1) == [node2]
    assert P2pibes.get_neighbors(node2) == [node3, node1]
    assert P2pibes.get_neighbors(node3) == [node4, node2]
    assert P2pibes.get_neighbors(node4) == [node5, node3]
    assert P2pibes.get_neighbors(node5) == []
  end

  test "Añadido de archivos" do
    {:ok, node1} = create_node([], :node1)
    P2pibes.add_file(node1, "fichero_inexistente.txt")
    assert P2pibes.get_files(node1) == []
    P2pibes.add_file(node1, "mix.exs")
    P2pibes.add_file(node1, "README.md")
    assert P2pibes.get_files(node1) == do_file("README.md") ++ do_file("mix.exs")
    P2pibes.add_file(node1, "mix.exs")
    assert P2pibes.get_files(node1) == do_file("README.md") ++ do_file("mix.exs")
  end

  test "Añadido de directorios" do
    {:ok, node1} = create_node([], :node1)
    P2pibes.add_file(node1, "test/prueba_añadido")

    assert P2pibes.get_files(node1) ==
             do_file("test/prueba_añadido/prueba_recursividad/fichero1.txt") ++
               do_file("test/prueba_añadido/fichero1.txt")
  end

  test "Borrado de archivos" do
    {:ok, node1} = create_node([], :node1)
    P2pibes.add_file(node1, "mix.exs")
    P2pibes.add_file(node1, "README.md")
    P2pibes.rm_file(node1, "README.md")
    assert P2pibes.get_files(node1) == do_file("mix.exs")
  end
```

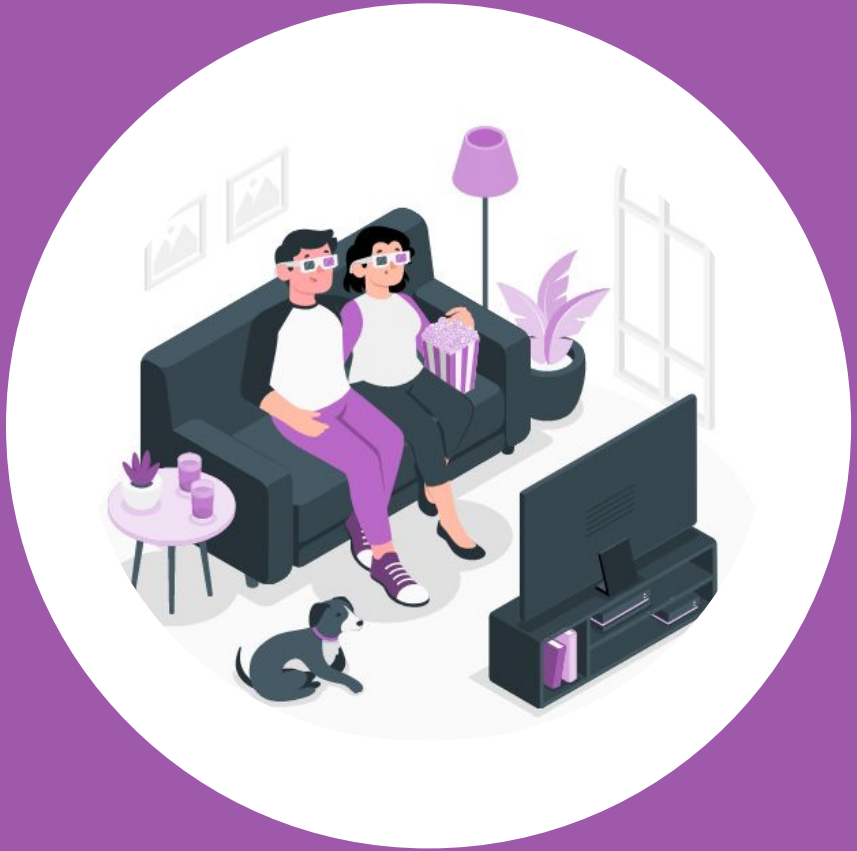
Documentación

```elixir
################ NEIGHBORS ################

  @doc """
  Agrega un `node2` en la lista de vecinos de `node1`. Completamente asíncrono,
responsabilidad del usuario comprobar que lo haga correctametne.
  ## Parameters
    - node1: nombre de un nodo.
    - node2: nombre de un nodo.
  """
  def add_neighbor(node1, node2) when node1 != node2,
    do: GenServer.cast(node1, {:add_neighbor, node2})

  @doc """
  Elimina un `node2` en la lista de vecinos de `node1`. Completamente asíncrono,
responsabilidad del usuario comprobar que lo haga correctametne.
  ## Parameters
    - node1: nombre de un nodo.
    - node2: nombre de un nodo.
  """
  def rm_neighbor(node1, node2), do: GenServer.cast(node1, {:rm_neighbor, node2})

  @doc """
  Obtiene los vecinoos de un nodo `node`.
  ## Parameters
    - node: nombre de un nodo.
  """
  def get_neighbors(node), do: GenServer.call(node, :get_neighbors)

  ################ FILES ################

  @doc """
  Agrega a `node` un fichero o un directorio ubicado en `path`. Completamente asíncrono,
responsabilidad del usuario comprobar que lo haga correctametne.
  ## Parameters
    - node: nombre de un nodo.
    - path: ubicación del fichero o directorio.
  """
  def add_file(node, path), do: GenServer.cast(node, {:add_file, path})

  @doc """
  Elimina de `node` un fichero o un directorio ubicado en `path`. Completamente asíncrono,
responsabilidad del usuario comprobar que lo haga correctametne.
  ## Parameters
    - node: nombre de un nodo.
    - path: ubicación del fichero o directorio.
  """
  def rm_file(node, path), do: GenServer.cast(node, {:rm_file, path})

  @doc """
  Obtiene los ficheros de un nodo `node`.
  ## Parameters
    - node: nombre de un nodo.
  """
  def get_files(node), do: GenServer.call(node, :get_files)
```



4

Demostración



Prestade atención á
live-demo!

fin