Taller de ruby Moove-It

Fausto Santinelli

9 de agosto de 2021

Índice general

1. Introducción.

La presente documentación tiene como objetivo dar una descripción detallada de los diferentes elementos de la aplicación realizada para el taller de ruby propuesto por Moove-It, la cual fue desarrollada a lo largo de una semana.

2. Antecedentes.

2.1. Desarrollo.

Para el desarrollo tanto del servidor como del cliente se utilizo Ruby, sin utilización de ninguna gema salvo rspec para la realización de pruebas.

Con el fin de poder cumplir con las características del taller, se partió de una carpeta vacía y se empezó a crear la estructura del proyecto a medida que era requerido.

Con el objetivo de poder manejar variables de entorno, también se creo dentro de la carpeta Config un archivo llamado "local_env" en el cual se podrán cambiar la dirección IP y el puerto del servidor, así como también el tiempo máximo que esperara respuestas el servidor por parte de un usuario conectado o el tiempo en el que se ejecutara el KeyPurger.

2.2. Repositorios

El repositorio del taller se encuentra en la siguiente direccion:

- https://github.com/FausSR/TallerRubyMooveIt

2.3. Requerimientos Funcionales.

A continuación se pasa a listas los requerimientos funcionales, es decir, las funcionalidades de las que se espera que el sistema disponga.

ID Requerimiento	Descripción
RF 1. Comando Set.	Los usuarios podrán enviar comandos de tipo storage a la aplicación con los parámetros "**set** key flags exptime bytes" seguido de un comando con los parámetros "data block". Esto deberá crear un registro clave-valor con la key, el valor, las flags, el largo en bytes y el tiempo de expiración correspondiente al mismo.
RF 2. Comando Add.	Los usuarios podrán enviar comandos de tipo storage a la aplicación con los parámetros "**add** key flags exptime byte" seguido de un comando con los parámetros "data block". Esto deberá añadir los datos correspondientes de los parámetros en una key no existente o eliminada. En caso de no existir la key correspondiente se retornara "STORED", en caso contrario se retornara "NOT-STORED".
RF 3. Comando Replace.	Los usuarios podrán enviar comandos de tipo storage a la aplicación con los parámetros "**Replace** key flags exptime byte" seguido de un comando con los parámetros "data block". Esto deberá añadir los datos correspondientes de los parámetros a una key existente. En caso de existir la key correspondiente se retornara "STORED", en caso contrario se retornara "NOT-STORED".
RF 4. Comando Append.	Los usuarios podrán enviar comandos de tipo storage a la aplicación con los parámetros "**Append** key flags exptime byte" seguido de un comando con los parámetros "data block". Esto deberá añadir los datos correspondientes de los parámetros a una key existente, poniendo el valor del data block al final del valor existente dentro de la key y sumando el numero de bytes correspondientes al registro. En caso de existir la key correspondiente se retornara "STORED", en caso contrario se retornara "NOT-STORED".

ID Requerimiento	Descripción
RF 5. Comando Prepend.	Los usuarios podrán enviar comandos de tipo storage a la aplicación con los parámetros "**Prepend** key flags exptime byte" seguido de un comando con los parámetros "data block". Esto deberá añadir los datos correspondientes de los parámetros a una key existente, poniendo el valor del data block al principio del valor existente dentro de la key y sumando el numero de bytes correspondientes al registro. En caso de existir la key correspondiente se retornara "STORED", en caso contrario se retornara "NOT-STORED".
RF 6. Comando Cas.	Los usuarios podrán enviar comandos de tipo storage a la aplicación con los parámetros "**cas** key flags exptime byte cas-unique" seguido de un comando con los parámetros "data block". Esto deberá añadir los datos correspondientes de los parámetros a una key existente solo en caso de que el valor de cas-unique enviado por parámetro sea igual al almacenada en la aplicación. En caso de existir la key correspondiente y tener el mismo valor de cas se retornara "STORED", en caso de existir un registro con la misma key pero con un valor de cas diferente se deberá retornar "EXISTS" y en caso de no encontrar ninguna key se deberá retornar "NOT-FOUND".
RF 7. Comando Get.	Los usuarios podrán enviar comandos de tipo retrieval a la aplicación con los parámetros "**get** key*". Esto deberá retornar los datos almacenados dentro de las keys existentes con los parámetros "VALUE key flags bytes", seguido de otro mensaje con los valores almacenados dentro de la key correspondiente. El usuario podrá mandar tantas keys como quiera y el servidor tendrá que responder con los datos correspondientes a cada una, pero al finalizar de enviar estos datos se tendrá que retornar "END". En caso de no existir la key correspondiente al parámetro no se retornara nada.

ID Requerimiento	Descripción
RF 8. Comando Gets.	Los usuarios podrán enviar comandos de tipo retrieval a la aplicación con los parámetros "**gets** key*". Esto deberá retornar los datos almacenados dentro de las keys existentes con los parámetros "VALUE key flags bytes casunique", seguido de otro mensaje con los valores almacenados dentro de la key correspondiente. El usuario podrá mandar tantas keys como quiera y el servidor tendrá que responder con los datos correspondientes a cada una, pero al finalizar de enviar estos datos se tendrá que retornar "END". En caso de no existir la key correspondiente al parámetro no se retornara nada.
RF 9. Error de comando.	Se deberá retornar "ERROR" en caso de recibir un comando no conocido
RF 10. Error de parámetro de comando.	Se deberá retornar "CLIENT_ERROR error" en caso de recibir alguno de los parámetros incorrectos correspondientes al del comando entregado.
RF 11. Error de servidor.	Se deberá retornar "CLIENT_SERVER error" en caso de que ocurra un problema con el servidor que obligue a cerrar la conexión.
RF 12. Borrado de keys.	Se deberán eliminar las keys que excedan el tiempo de expiración asignado por el usuario.
RF 13. Conexión con la aplicación.	Los usuarios podrán conectarse a la aplicación mediante conexión TCP/IP.

2.4. Requerimientos no Funcionales.

Ahora se muestra la lista de requerimientos no funcionales.

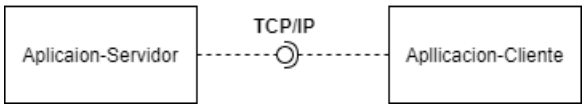
ID Requerimiento	Descripción
RNF 1. Conexión de múltiples usuarios.	El servidor deberá poder interactuar con múltiples usuarios conectados a la vez.
RNF 2. Lenguaje.	La aplicación deberá ser creada utilizando únicamente lenguaje Ruby
RNF 3. No utilización de gemas.	El servidor no podrá utilizar ningún tipo de gema de Ruby, a excepción de rspec la cual se utilizara para la creación de pruebas.
RNF 4. Testing de comandos.	Tendrán que crearse pruebas para cada uno de los comandos creados.
RNF 5. Almacenamiento en memoria.	El servidor deberá almacenar la información correspondiente en memoria.

3. Diseño.

En este capitulo de la documentación se pasa a mostrar los diagramas correspondientes y a explicar el comportamiento del programa creado.

3.1. Arquitectura del Sistema.

Para el taller se crearon dos programas diferentes. El servidor, en el cual estuvo la mayor parte del trabajo, y la demo del cliente la cual solamente ejecuta un programa que se conecta al servidor.



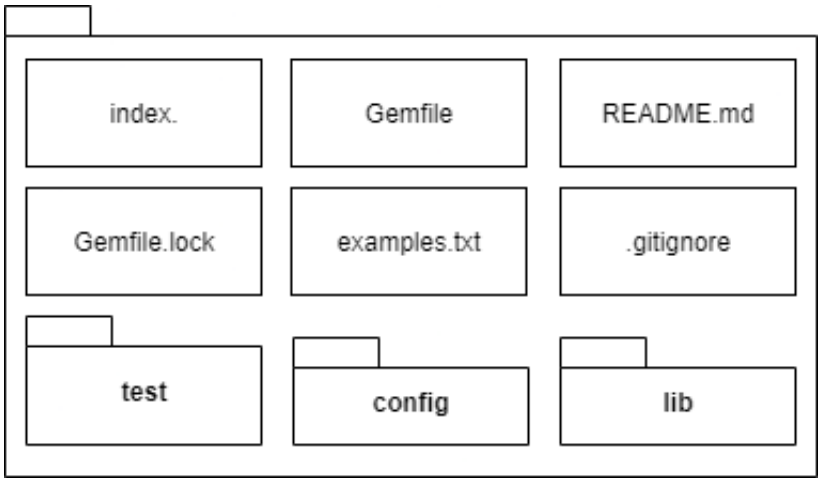
3.2. Servidor.

- Descripción: Es el servidor el cual acepta conexiones de clientes y responde a estos mediante lo establecido en la parte anterior.
- Lenguaje: Ruby.

3.2.1. Vistas de Descomposición.

main

Representación Primaria.

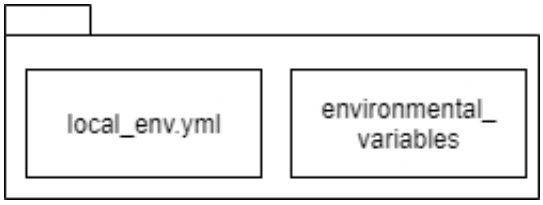


Catalogo de Elementos.

Elemento	Responsabilidades
index	Archivo de ejecución de el servidor. Este archivo crea el almacenamiento que se usara en memoria, crea la clase que leerá las variables de entorno, crea la clase encargada de borrar las keys expiradas y crea la clase encargada de ejecutar el servidor TCP y toda la lógica relacionada .
config	Contiene el archivo con variables de entorno y la clase encargada de cargarlas.
lib	Contiene toda la lógica del servidor.
test	Contiene todas las pruebas de los comandos.
examples.txt	Contiene ejemplos de la ejecución de comandos de la aplicación.
Gemfile	Archivo con todas las gemas que usamos en la aplicación.
Gemfile.lock	Archivo de specs y versiones de gemas.
.gitignore	Evita que archivos se suban al repositorio.
README.md	Archivo con información valiosa sobre el repositorio.

config

Representación Primaria.

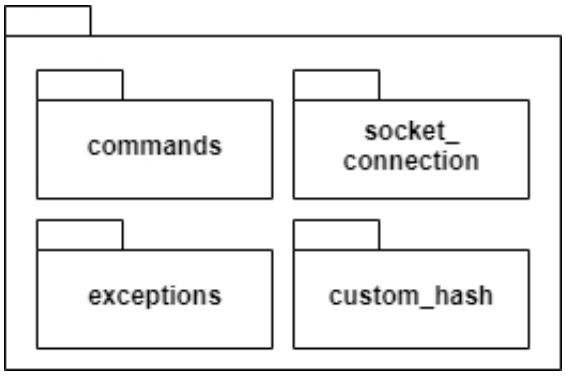


Catalogo de Elementos.

local_env.yml	Contiene las variables de entorno aplicadas al sistema.
environmental_variables	Clase utilizada para leer las variables de entorno.

lib

Representación Primaria.

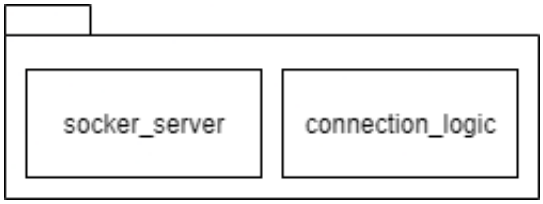


Catalogo de Elementos.

commands	Contiene la lógica de los comandos.
socket_connection	Contiene la lógica del servidor y la lectura de mensajes.
exceptions	Contiene exceptions personalizadas para el servidor.
custom_hash	Contiene la lógica del hash utilizado para almacenar información.

socket_connection

Representación Primaria.

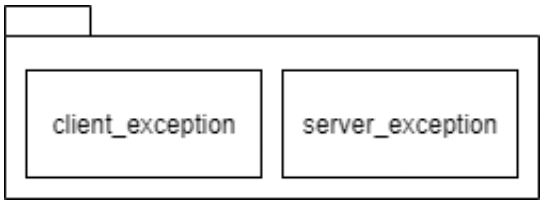


Catalogo de Elementos.

socket_server	Clase encargada de crear el servidor y los hilos de clientes.
connection_logic	Clase encargada de leer los mensajes del cliente y asignarlos a la clase de comando encargada de la lógica correspondiente a cada uno. También filtra los comandos no conocidos.

exceptions

Representación Primaria.

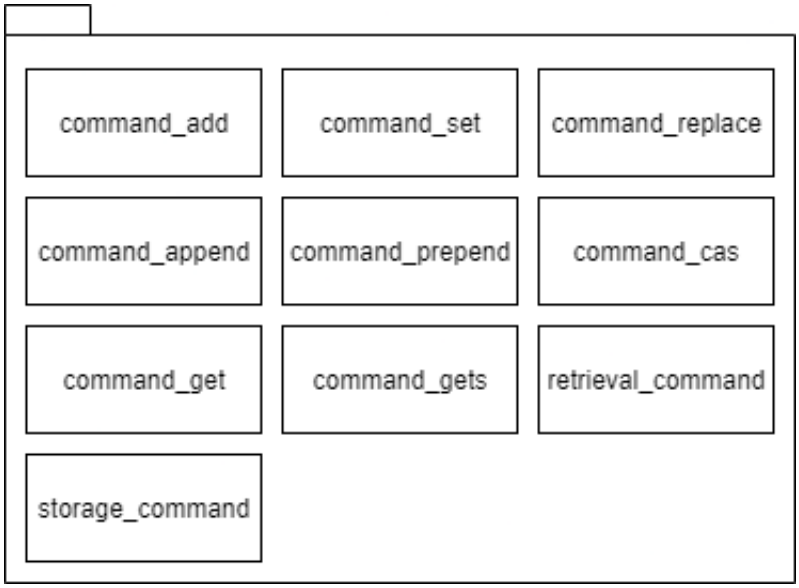


Catalogo de Elementos.

server_exception	Exception creada para errores de servidor.
client_exception	Exception creada para errores de clientes.

commands

Representación Primaria.

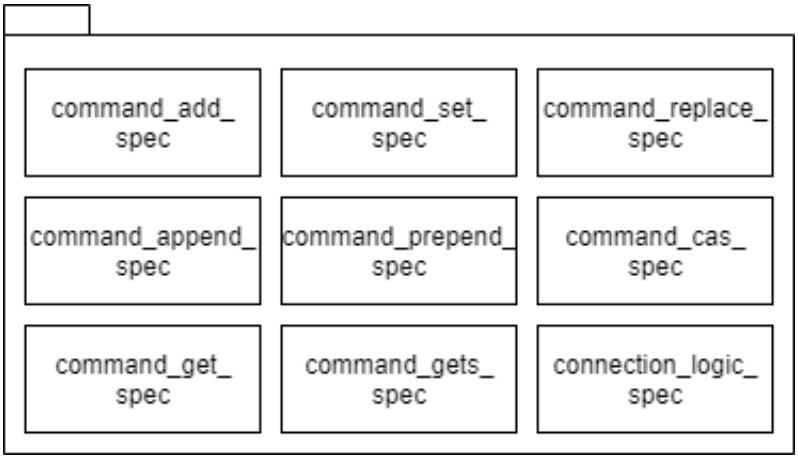


Catalogo de Elementos.

command_add	Clase con la lógica correspondiente a los comandos add.
command_set	Clase con la lógica correspondiente a los comandos set.
command_replace	Clase con la lógica correspondiente a los comandos replace.
command_append	Clase con la lógica correspondiente a los comandos append.
command_prepend	Clase con la lógica correspondiente a los comandos prepend.
command_cas	Clase con la lógica correspondiente a los comandos cas.
command_get	Clase con la lógica correspondiente a los comandos get.
command_gets	Clase con la lógica correspondiente a los comandos gets.
retrieval_command	Clase con lógica utilizada por todos los comandos de tipo retrieval.
storage_command	Clase con lógica utilizada por todos los comandos de tipo storage.

test

Representación Primaria.

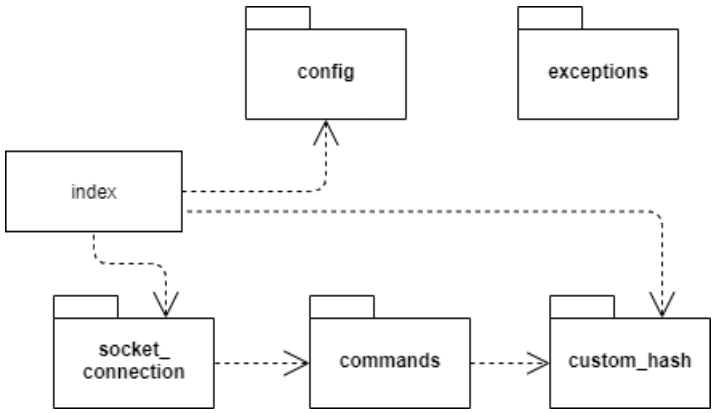


Catalogo de Elementos.

command_add_spec	Clase con las pruebas para los comandos add.
command_set_spec	Clase con las pruebas para los comandos set.
command_replace_spec	Clase con las pruebas para los comandos replace.
command_append_spec	Clase con las pruebas para los comandos append.
command_prepend_spec	Clase con las pruebas para los comandos prepend.
command_cas_spec	Clase con las pruebas para los comandos cas.
command_get_spec	Clase con las pruebas para los comandos get.
command_gets_spec	Clase con las pruebas para los comandos gets.

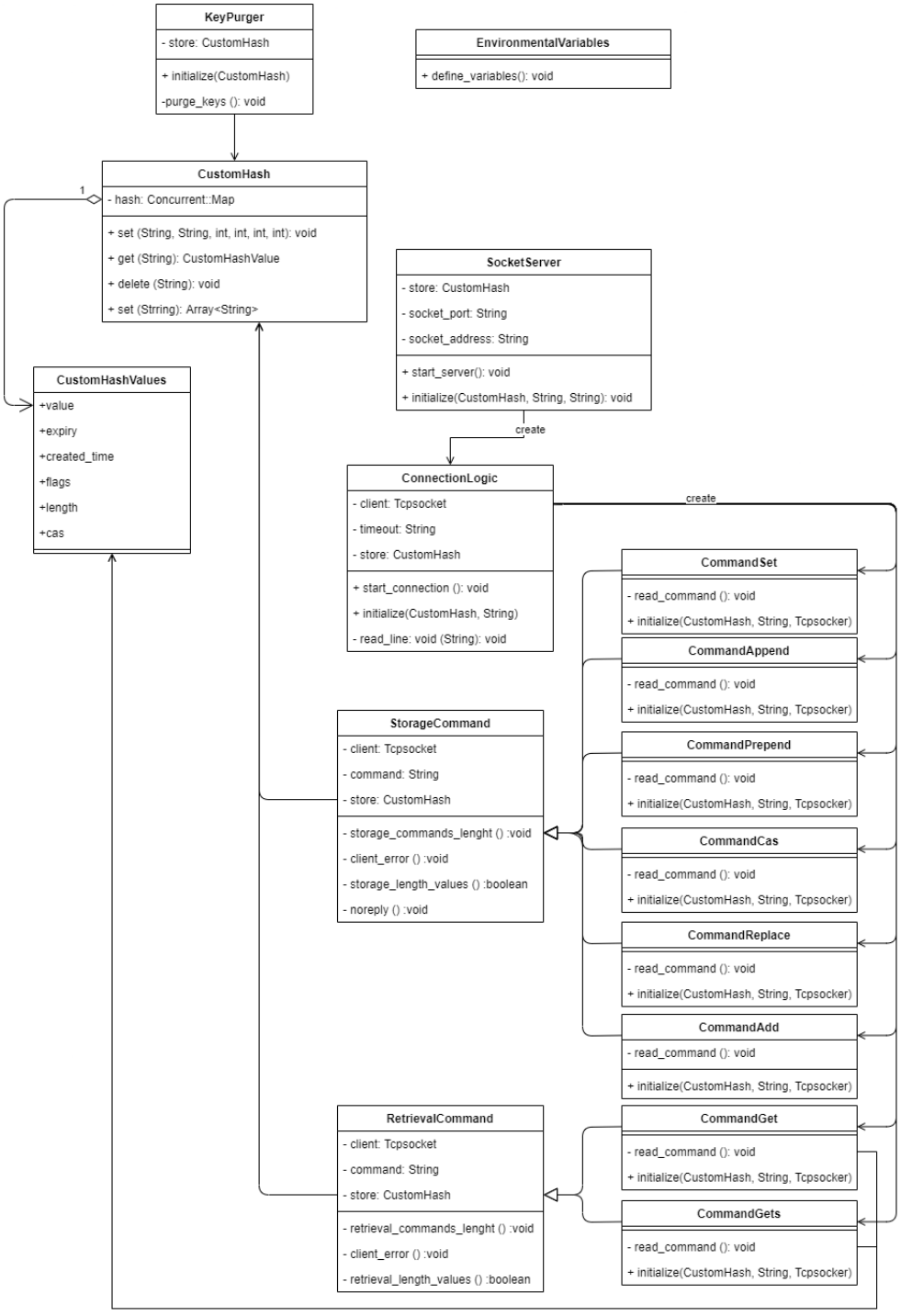
3.2.2. Vista de Uso.

Las excepciones se crearon pero no se vio necesario utilizar estas.



3.2.3. Diagrama de clases.

Cabe destacar que se coloco en los parámetros de los métodos el tipo esperado al llamarlos, debido a que no se restringieron tipos específicos.



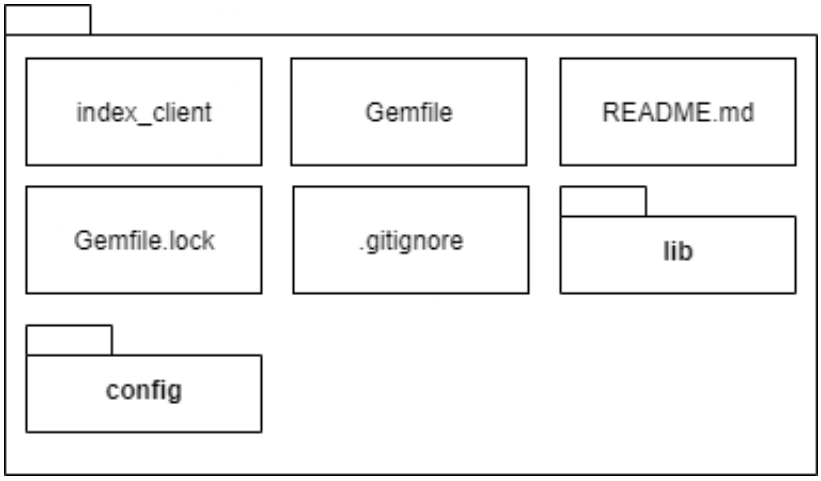
3.3. Cliente.

- Descripción: Es el cliente el cual se conecta al servidor y le envia comandos.
- Lenguaje: Ruby.

3.3.1. Vistas de Descomposición.

main

Representación Primaria.

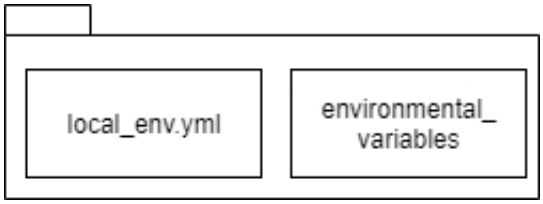


Catalogo de Elementos.

Elemento	Responsabilidades
index_client	Archivo de ejecución de el cliente. Este archivo crea la clase que leerá las variables de entorno y crea la clase encargada de ejecutar el clinte TCP y toda la lógica relacionada .
config	Contiene el archivo con variables de entorno y la clase encargada de cargarlas.
lib	Contiene la lógica del cliente.
Gemfile	Archivo con todas las gemas que usamos en la aplicación.
Gemfile.lock	Archivo de specs y versiones de gemas.
.gitignore	Evita que archivos se suban al repositorio.
README.md	Archivo con información valiosa sobre el repositorio.

config

Representación Primaria.

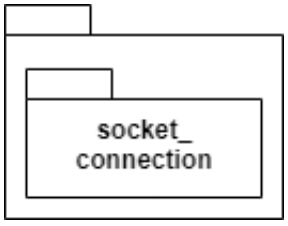


Catalogo de Elementos.

local_env.yml	Contiene las variables de entorno aplicadas al sistema.
environmental_variables	Clase utilizada para leer las variables de entorno.

lib

Representación Primaria.

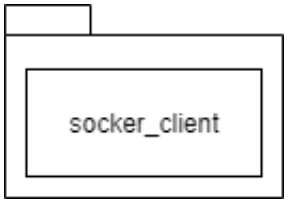


Catalogo de Elementos.

socket_connection	Contiene la lógica del cliente.

socket_connection

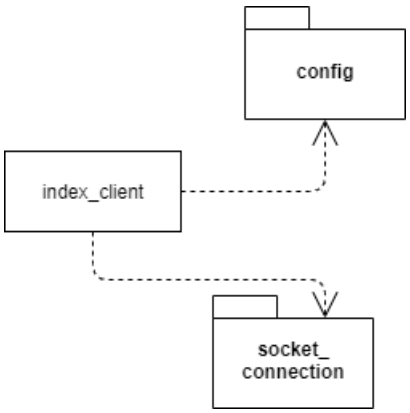
Representación Primaria.



Catalogo de Elementos.

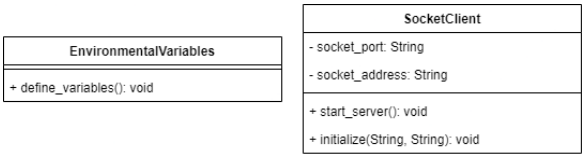
socket_client	Clase encargada de crear el cliente y un hilos de ejecución que lee los mensajes del servidor. Tras la generación de estos, se queda en loop permitiendo enviar mensajes al servidor.

3.3.2. Vista de Uso.



3.3.3. Diagrama de clases.

Cabe destacar que se coloco en los parámetros de los métodos el tipo esperado al llamarlos, debido a que no se restringieron tipos específicos.



4. Justificaciones de Diseño.

A continuación se pasara a detallar algunas de las acciones tomadas para la realización de determinados requerimientos.

RF 12. Borrado de keys.

Dado que se tuvo que crear una estructura personalizada para el almacenamiento, se establecieron dos formas de borrar datos asignados a keys.

Lo primero fue almacenar la hora en que se crean los valores asignados a las keys para podre siempre comparar la hora actual con la hora almacenada.

Un métodos esta presentes en el mismo método de get() del CustomHash, en el que se restan la hora actual con la almacenada y en caso que esto sea mayor al tiempo de expiración, se almacena un valor nulo en la key correspondiente.

Además de la primera acción, se opto por crear una clase la cual, cada x cantidad de tiempo definida en las variables de entorno, recorrerá el CustomHash y eliminara las claves.

RNF 1. Conexión de múltiples usuarios.

Para permitir la conexión de múltiples usuarios, cada vez que se acepte la conexión de un cliente se creara un nuevo hilo, el cual se encargara de ejecutar todos los comandos enviados por el usuario.

RNF 4. Testing de comandos.

Tal y como especificaba la letra, se realizo pruebas de los comandos para cada uno de los casos. También por esto se opto por no realizar testing para el CustomHash o clases extra implementados en el sistema.

RNF 5. Almacenamiento en memoria.

Debido a que no se podian utilizar gemas como Redis o similares, se opto por crear un Hash personalizado el cual pueda cumplir con todos los requisitos necesarios para el sistema.

Este Hash contiene una estructura de valores creado para almacenar los datos relevantes que se deberán almacenar en el servidor.