BlinkDB

1.0

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 APIGateway Class Reference

Acts as an intermediary between clients and the BlinkDB storage system.

`#include <APIGateway.h>`

Public Member Functions

- APIGateway (BlinkDB &blinkDB)

  Constructs the APIGateway and initializes services.
- string executeCommand (Command command)

  Executes a given command by routing it to the appropriate service.

3.1.1 Detailed Description

Acts as an intermediary between clients and the BlinkDB storage system.

The APIGateway processes incoming commands, interacts with the cache, and routes requests to the appropriate services for handling `set`, `get`, and `del` operations.

Definition at line 19 of file APIGateway.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 APIGateway()

```
APIGateway::APIGateway (
            BlinkDB & blinkDB)  [inline]
```

Constructs the APIGateway and initializes services.

This constructor initializes the cache and services required for handling database interactions.

Parameters

blinkDB	Reference to the BlinkDB database instance.

Definition at line 66 of file APIGateway.h.

```
00066                                 : blinkDB(blinkDB), setService(blinkDB), getService(blinkDB),
      delService(blinkDB)
00067     {
00068         L1Cache = Cache();
00069     }
```

3.1.3 Member Function Documentation

3.1.3.1 executeCommand()

```
string APIGateway::executeCommand (
            Command command)  [inline]
```

Executes a given command by routing it to the appropriate service.

This function determines the command type and delegates processing to the corresponding service (`set`, `get`, `del`).

- `set`: Stores the key-value pair in both cache and database.
- `get`: Retrieves the value from the cache or falls back to the database.
- `del`: Removes the key from both cache and database.

Parameters

command	The command object containing operation type, key, and optional value.

Returns

std::string Response message indicating success or failure.

Definition at line 84 of file APIGateway.h.

```
00085     {
00086         if (command.getCommand() == "SET")
00087         {
00088             // Clear cache if size exceeds limit
00089             if (L1Cache.getSize() >= 100000000)
00090             {
00091                 L1Cache.clear();
00092             }
00093             // Store in cache and database
00094             L1Cache.set(command.getKey(), command.getValue());
00095             setService.set(command.getKey(), command.getValue());
00096             return "Set Success";
00097         }
00098         else if (command.getCommand() == "GET")
00099         {
00100             // Check cache first
00101             string cacheCheckResult = L1Cache.get(command.getKey());
00102             if (cacheCheckResult == "-1")
00103             {
00104                 return getService.get(command.getKey());
00105             }
00106             else
00107             {
```

```
00108                 return cacheCheckResult;
00109             }
00110         }
00111         else if (command.getCommand() == "DEL")
00112         {
00113             // Remove from cache and database
00114             L1Cache.del(command.getKey());
00115             delService.del(command.getKey());
00116             return "Deletion Success";
00117         }
00118         return "Invalid Command";
00119     }
```

The documentation for this class was generated from the following file:

- APIGateway/APIGateway.h

3.2 BlinkDB Class Reference

Implements an in-memory key-value database with periodic disk backups.

`#include <BlinkDB.h>`

Public Member Functions

- BlinkDB ()

  Constructs a BlinkDB instance.
- ~BlinkDB ()

  Destructor that ensures cleanup of resources.
- void set (string key, string value)

  Stores a key-value pair in the database.
- string get (string key)

  Retrieves the value associated with a key.
- void del (string key)

  Deletes a key-value pair from the database.

3.2.1 Detailed Description

Implements an in-memory key-value database with periodic disk backups.

This class provides functionality to store, retrieve, and delete key-value pairs efficiently. It also manages background backups to disk for data persistence.

Definition at line 14 of file BlinkDB.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 BlinkDB()

`BlinkDB::BlinkDB () [inline]`

Constructs a BlinkDB instance.

Initializes the in-memory database and starts the background backup thread.

Definition at line 97 of file BlinkDB.h.

```
00098     {
00099         DiscBackupHandler discBackupHandler = DiscBackupHandler();
00100         database = unordered_map<string, string>();
00101         backupThread = thread(&BlinkDB::backupWorker, this);
00102         buffer = unordered_map<string, string>();
00103     }
```

3.2.2.2 ~BlinkDB()

```
BlinkDB::~BlinkDB ()  [inline]
```

Destructor that ensures cleanup of resources.

Stops the background backup worker thread before shutting down.

Definition at line 110 of file BlinkDB.h.

```
00111     {
00112         stopBackup = true;
00113         if (backupThread.joinable())
00114             backupThread.join();
00115     }
```

3.2.3 Member Function Documentation

3.2.3.1 del()

```
void BlinkDB::del (
             string key)  [inline]
```

Deletes a key-value pair from the database.

If the key exists, it is removed from the in-memory store.

Parameters

key	The key to be deleted.

Definition at line 155 of file BlinkDB.h.

```
00156     {
00157         database.erase(key);
00158     }
```

3.2.3.2 get()

```
string BlinkDB::get (
             string key)  [inline]
```

Retrieves the value associated with a key.

First, it checks the in-memory database. If not found, it attempts to retrieve the value from the disk backup.

Parameters

key	The key to look up.

Returns

string The corresponding value if found, otherwise an empty string.

Definition at line 139 of file BlinkDB.h.

```
00140     {
00141         if (database.find(key) != database.end())
00142         {
00143             return database[key];
00144         }
00145         return discBackupHandler.checkBackupForKey(key);
00146     }
```

3.2.3.3 set()

```
void BlinkDB::set (
            string key,
            string value)  [inline]
```

Stores a key-value pair in the database.

If the key already exists, its value is updated.

Parameters

key	The key to store.
value	The associated value.

Definition at line 125 of file BlinkDB.h.

```
00126     {
00127         database[key] = value;
00128     }
```

The documentation for this class was generated from the following file:

- Database/BlinkDB.h

3.3 Cache Class Reference

Provides an in-memory key-value store.

```
#include <Cache.h>
```

Public Member Functions

- void set (string key, string value)
  Stores a key-value pair in the cache.
- string get (string key)
  Retrieves a value associated with the given key.
- void del (string key)
  Deletes a key-value pair from the cache.
- size_t getSize ()
  Gets the current size of the cache.
- void clear ()
  Clears all key-value pairs from the cache.

3.3.1 Detailed Description

Provides an in-memory key-value store.

The Cache class is designed to store frequently accessed key-value pairs to improve performance by reducing direct database queries.

Definition at line 13 of file Cache.h.

3.3.2 Member Function Documentation

3.3.2.1 clear()

```
void Cache::clear ()  [inline]
```

Clears all key-value pairs from the cache.

This function removes all stored entries, effectively resetting the cache.

Definition at line 85 of file Cache.h.

```
00086     {
00087         cache.clear();
00088     }
```

3.3.2.2 del()

```
void Cache::del (
             string key)  [inline]
```

Deletes a key-value pair from the cache.

If the key exists in the cache, it is removed.

Parameters

key	The key to be deleted.

Definition at line 63 of file Cache.h.

```
00064     {
00065         cache.erase(key);
00066     }
```

3.3.2.3 get()

```
string Cache::get (
             string key)  [inline]
```

Retrieves a value associated with the given key.

This function looks up a key in the cache and returns the associated value. If the key is not found, it returns "-1".

Parameters

key	The key to search for.

Returns

std::string The corresponding value if found, otherwise "-1".

Definition at line 47 of file Cache.h.

```
00048     {
00049         if (cache.find(key) == cache.end())
00050         {
00051             return "-1";
00052         }
00053         return cache[key];
00054     }
```

3.3.2.4 getSize()

```
size_t Cache::getSize ()  [inline]
```

Gets the current size of the cache.

This function returns the number of key-value pairs currently stored in the cache.

Returns

size_t The total number of stored key-value pairs.

Definition at line 75 of file Cache.h.

```
00076     {
00077         return cache.size();
00078     }
```

3.3.2.5 set()

```
void Cache::set (
             string key,
             string value)  [inline]
```

Stores a key-value pair in the cache.

This function inserts a new key-value pair into the cache. If the key already exists, its value is updated.

Parameters

key	The key to be stored.
value	The corresponding value.

Definition at line 33 of file Cache.h.

```
00034     {
00035         cache[key] = value;
00036     }
```

The documentation for this class was generated from the following file:

- Cache/Cache.h

3.4 Command Class Reference

Represents a user command in BlinkDB.

```
#include <Command.h>
```

Public Member Functions

- Command ()

  Default constructor initializing empty command, key, and value.
- Command (string command, string key, string value)

  Constructs a Command with a specified command type, key, and value.
- Command (string command, string key)

  Constructs a Command with a command type and key (for "get" and "del" commands).
- string getCommand ()

  Retrieves the command type.
- string getKey ()

  Retrieves the key associated with the command.
- string getValue ()

  Retrieves the value associated with the command (only relevant for "set").

3.4.1 Detailed Description

Represents a user command in BlinkDB.

This class encapsulates a database command with a command type, key, and optional value.

Definition at line 11 of file Command.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Command() [1/3]

```
Command::Command ()  [inline]
```

Default constructor initializing empty command, key, and value.

Definition at line 33 of file Command.h.

```
00034     {
00035         command = "";
00036         key = "";
00037         value = "";
00038     }
```

3.4.2.2 Command() [2/3]

```
Command::Command (
            string command,
            string key,
            string value)  [inline]
```

Constructs a Command with a specified command type, key, and value.

Parameters

command	The command type (e.g., "set").
key	The key to be used in the operation.
value	The value to be set (only relevant for "set" commands).

Definition at line 47 of file Command.h.

```
00048     {
00049         this->command = command;
00050         this->key = key;
00051         this->value = value;
00052     }
```

3.4.2.3 Command() [3/3]

```
Command::Command (
            string command,
            string key)  [inline]
```

Constructs a Command with a command type and key (for "get" and "del" commands).

Parameters

command	The command type (e.g., "get", "del").
key	The key associated with the operation.

Definition at line 60 of file Command.h.

```
00061     {
00062         this->command = command;
00063         this->key = key;
00064         this->value = "";
00065     }
```

3.4.3 Member Function Documentation

3.4.3.1 getCommand()

```
string Command::getCommand ()  [inline]
```

Retrieves the command type.

Returns

The command type as a string.

Definition at line 72 of file Command.h.

```
00073     {
00074         return command;
00075     }
```

3.4.3.2 getKey()

```
string Command::getKey ()  [inline]
```

Retrieves the key associated with the command.

Returns

The key as a string.

Definition at line 82 of file Command.h.

```
00083     {
00084         return key;
00085     }
```

3.4.3.3 getValue()

```
string Command::getValue ()  [inline]
```

Retrieves the value associated with the command (only relevant for "set").

Returns

The value as a string.

Definition at line 92 of file Command.h.

```
00093     {
00094         return value;
00095     }
```

The documentation for this class was generated from the following file:

- Models/Command.h

3.5 DelService Class Reference

Service class for handling key deletion in BlinkDB.

```
#include <DelService.h>
```

Public Member Functions

- DelService (BlinkDB &blinkDB)
  Constructs a DelService instance.
- void del (const string &key)
  Deletes a key from BlinkDB.

3.5.1 Detailed Description

Service class for handling key deletion in BlinkDB.

This class provides an interface for deleting keys from the BlinkDB database.

Definition at line 12 of file DelService.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 DelService()

```
DelService::DelService (
            BlinkDB & blinkDB)  [inline], [explicit]
```

Constructs a DelService instance.

Parameters

blinkDB	Reference to the BlinkDB database instance.

Definition at line 26 of file DelService.h.

```
00026 : blinkDB(blinkDB) {}
```

3.5.3 Member Function Documentation

3.5.3.1 del()

```
void DelService::del (
            const string & key)  [inline]
```

Deletes a key from BlinkDB.

Parameters

key	The key to be deleted.

Definition at line 33 of file DelService.h.

```
00034     {
00035         try
00036         {
00037             blinkDB.del(key);
00038         }
00039         catch (const exception &e)
00040         {
00041             cerr « "Error deleting key '" « key « "': " « e.what() « endl;
00042         }
00043     }
```

The documentation for this class was generated from the following file:

- Services/DelService.h

3.6 DiscBackupHandler Class Reference

The DiscBackupHandler class handles disk-based backups for BlinkDB.

```
#include <DiscBackupHandler.h>
```

Public Member Functions

- DiscBackupHandler ()
  Constructs a DiscBackupHandler instance.
- bool backup (const unordered_map< string, string > &map)
  Initiates a backup of the given database map to a temporary file.
- bool commitBackup ()
  Commits the buffered backup to permanent disk storage.
- bool terminate ()
  Deletes all backup files, effectively clearing the backup storage.
- string checkBackupForKey (const string &key)
  Checks if a key exists in the disk backup and retrieves its value.

3.6.1 Detailed Description

The DiscBackupHandler class handles disk-based backups for BlinkDB.

This class manages periodic backups of the database by writing data to files, sorting and distributing data across multiple backup files, and retrieving data from disk when necessary.

Definition at line 16 of file DiscBackupHandler.h.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 DiscBackupHandler()

```
DiscBackupHandler::DiscBackupHandler ()  [inline]
```

Constructs a DiscBackupHandler instance.

Ensures the backup directory exists before performing any operations.

Definition at line 111 of file DiscBackupHandler.h.

```
00112     {
00113         std::filesystem::create_directories("./backups");
00114     }
```

3.6.3 Member Function Documentation

3.6.3.1 backup()

```
bool DiscBackupHandler::backup (
            const unordered_map< string, string > & map)  [inline]
```

Initiates a backup of the given database map to a temporary file.

Parameters

map	The database contents to be backed up.

Returns

true if backup was successful, false otherwise.

Definition at line 122 of file DiscBackupHandler.h.

```
00123     {
00124         return backupFromMapToBufferFile(map);
00125     }
```

3.6.3.2 checkBackupForKey()

```
string DiscBackupHandler::checkBackupForKey (
            const string & key)  [inline]
```

Checks if a key exists in the disk backup and retrieves its value.

Searches for the key in the appropriate backup file (`0.txt` to `9.txt`) based on the first character of the key.

Parameters

key	The key to search for.

Returns

The value associated with the key if found, "-1" if not found, "-2" if an error occurs.

Definition at line 161 of file DiscBackupHandler.h.

```
00162     {
00163         lock_guard<mutex> lock(backupMutex);
00164         string filename = key.substr(0, 1) + ".txt";
00165         ifstream backupBuffer("./backups/" + filename);
00166         if (!backupBuffer.is_open())
00167         {
00168             return "-2";
00169         }
00170 
00171         string line, foundValue = "-1";
00172         while (getline(backupBuffer, line))
00173         {
00174             if (utils.startsWith(line, key))
00175             {
00176                 size_t pos = line.find(" ");
00177                 if (pos != string::npos)
00178                 {
00179                     foundValue = line.substr(pos + 1);
00180                 }
00181                 break;
00182             }
00183         }
00184         return foundValue;
00185     }
```

3.6.3.3 commitBackup()

```
bool DiscBackupHandler::commitBackup ()  [inline]
```

Commits the buffered backup to permanent disk storage.

Transfers data from the temporary buffer file (`backup.txt`) to categorized backup files (`0.txt` to `9.txt`).

Returns

true if the commit was successful, false otherwise.

Definition at line 135 of file DiscBackupHandler.h.

```
00136     {
00137         return backupFromBufferFileToDisc();
00138     }
```

3.6.3.4 terminate()

```
bool DiscBackupHandler::terminate ()  [inline]
```

Deletes all backup files, effectively clearing the backup storage.

Returns

true always.

Definition at line 145 of file DiscBackupHandler.h.

```
00146     {
00147         lock_guard<mutex> lock(backupMutex);
00148         std::filesystem::remove_all("./backups");
00149         return true;
00150     }
```

The documentation for this class was generated from the following file:

- Handlers/DiscBackupHandler.h

3.7 GetService Class Reference

Service class for retrieving values from BlinkDB.

```
#include <GetService.h>
```

Public Member Functions

- GetService (BlinkDB &blinkDB)
  Constructs a GetService instance.
- string get (const string &key)
  Retrieves the value associated with a given key.

3.7.1 Detailed Description

Service class for retrieving values from BlinkDB.

This class provides an interface for fetching values associated with a given key.

Definition at line 12 of file GetService.h.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 GetService()

```
GetService::GetService (
            BlinkDB & blinkDB)  [inline], [explicit]
```

Constructs a GetService instance.

Parameters

blinkDB	Reference to the BlinkDB database instance.

Definition at line 26 of file GetService.h.

```
00026 : blinkDB(blinkDB) {}
```

3.7.3 Member Function Documentation

3.7.3.1 get()

```
string GetService::get (
            const string & key)  [inline]
```

Retrieves the value associated with a given key.

Parameters

key	The key whose value is to be retrieved.

Returns

The value corresponding to the key, or "Key not found" if it does not exist.

Definition at line 34 of file GetService.h.

```
00035     {
00036         try
00037         {
00038             string value = blinkDB.get(key);
00039             return value.empty() ? "Key not found" : value;
00040         }
00041         catch (const exception &e)
00042         {
00043             cerr « "Error retrieving key '" « key « "': " « e.what() « endl;
00044             return "Error retrieving value";
00045         }
00046     }
```

The documentation for this class was generated from the following file:

- Services/GetService.h

3.8 Response Class Reference

Represents an API response in BlinkDB.

`#include <Response.h>`

Public Member Functions

- Response ()
  Default constructor initializing an empty response.
- Response (int statusCode, string message, pair< string, string > data)
  Constructs a Response with a status code, message, and data.
- Response (int statusCode, string key, string value, string message)
  Constructs a Response with a status code, key, value, and message.
- string to_string ()
  Converts the response to a string representation.
- string getValue ()

3.8.1 Detailed Description

Represents an API response in BlinkDB.

This class encapsulates the response status, message, and optional key-value data.

Definition at line 11 of file Response.h.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Response() [1/3]

```
Response::Response ()  [inline]
```

Default constructor initializing an empty response.

Definition at line 33 of file Response.h.

```
00034     {
00035         statusCode = 0;
00036         message = "";
00037         data = pair<string, string>("Data", "");
00038     }
```

3.8.2.2 Response() [2/3]

```
Response::Response (
            int statusCode,
            string message,
            pair< string, string > data)  [inline]
```

Constructs a Response with a status code, message, and data.

Parameters

statusCode	The status code of the response.
message	The response message.
data	The key-value pair representing data.

Definition at line 47 of file Response.h.

```
00048     {
00049         this->statusCode = statusCode;
00050         this->message = message;
00051         this->data = data;
00052     }
```

3.8.2.3 Response() [3/3]

```
Response::Response (
            int statusCode,
            string key,
            string value,
            string message)  [inline]
```

Constructs a Response with a status code, key, value, and message.

Parameters

statusCode	The status code of the response.
key	The key associated with the data.
value	The value associated with the key.
message	The response message.

Definition at line 62 of file Response.h.

```
00063     {
00064         this->statusCode = statusCode;
00065         this->message = message;
00066         this->data = pair<string, string>(key, value);
00067     }
```

3.8.3 Member Function Documentation

3.8.3.1 getValue()

```
string Response::getValue ()  [inline]
```

Definition at line 79 of file Response.h.

```
00080     {
00081         if (data.first == "Data")
00082         {
00083             return data.second;
00084         }
00085         return "-1";
00086     }
```

3.8.3.2 to_string()

```
string Response::to_string ()  [inline]
```

Converts the response to a string representation.

Returns

A formatted string containing the status code, message, and data.

Definition at line 74 of file Response.h.

```
00075     {
00076         return "Status Code: " + std::to_string(statusCode) + ", Message: " + message + ", Data: " +
      data.second;
00077     }
```

The documentation for this class was generated from the following file:

- Models/Response.h

3.9 SetService Class Reference

Service class for setting key-value pairs in BlinkDB.

```
#include <SetService.h>
```

Public Member Functions

- SetService (BlinkDB &blinkDB)

  Constructs a SetService instance.
- void set (const string &key, const string &value)

  Stores a key-value pair in BlinkDB.

3.9.1 Detailed Description

Service class for setting key-value pairs in BlinkDB.

This class provides an interface for storing key-value pairs in the BlinkDB database.

Definition at line 11 of file SetService.h.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 SetService()

```
SetService::SetService (
            BlinkDB & blinkDB)  [inline], [explicit]
```

Constructs a SetService instance.

Parameters

blinkDB	Reference to the BlinkDB database instance.

Definition at line 25 of file SetService.h.

```
00025 : blinkDB(blinkDB) {}
```

3.9.3 Member Function Documentation

3.9.3.1 set()

```
void SetService::set (
            const string & key,
            const string & value)  [inline]
```

Stores a key-value pair in BlinkDB.

Parameters

key	The key to store.
value	The value associated with the key.

Definition at line 33 of file SetService.h.

```
00034     {
00035         try
00036         {
00037             blinkDB.set(key, value);
00038         }
00039         catch (const exception &e)
00040         {
00041             cerr « "Error setting key '" « key « "': " « e.what() « endl;
00042         }
00043     }
```

The documentation for this class was generated from the following file:

- Services/SetService.h

3.10 Utils Class Reference

Utility class providing helper functions for hashing, string manipulation, and pattern matching.

```
#include <Utils.h>
```

Public Member Functions

- string hash (const string &key)

  Hashes a string using Boost's hash function.
- vector< string > splitCommand (string command)

  Splits a command string into at most three parts (command, key, and value).
- bool startsWith (const string &str, const string &prefix)

  Checks if a given string starts with a specified prefix.
- string toRESP2 (const string &command)

  Converts a string to Redis Serialization Protocol (RESP2) format.
- vector< string > fromRESP2 (const string &resp)

  Parses a RESP2-formatted string back to a normal string.

3.10.1 Detailed Description

Utility class providing helper functions for hashing, string manipulation, and pattern matching.

Definition at line 10 of file Utils.h.

3.10.2 Member Function Documentation

3.10.2.1 fromRESP2()

```
vector< string > Utils::fromRESP2 (
            const string & resp)  [inline]
```

Parses a RESP2-formatted string back to a normal string.

Parameters

resp	The RESP2-formatted string.

Returns

The extracted string data.

Definition at line 103 of file Utils.h.

```
00104     {
00105         vector<string> result;
00106         istringstream stream(resp);
00107         string line;
00108 
00109         getline(stream, line, '\r');
00110         if (line[0] != '*')
00111             return {}; // Must start with '*'
00112 
00113         int numArgs = stoi(line.substr(1)); // Number of arguments
00114         stream.ignore(1);                   // Ignore '\n'
00115 
00116         for (int i = 0; i < numArgs; i++)
00117         {
00118             getline(stream, line, '\r');
00119             if (line[0] != '$')
00120                 return {}; // Must start with '$'
00121 
00122             int len = stoi(line.substr(1)); // Get length of argument
00123             stream.ignore(1);               // Ignore '\n'
00124 
00125             string arg(len, ' ');
00126             stream.read(&arg[0], len); // Read the argument
00127             result.push_back(arg);
00128 
00129             stream.ignore(2); // Ignore '\r\n'
00130         }
00131 
00132         return result;
00133     }
```

3.10.2.2 hash()

```
string Utils::hash (
            const string & key)  [inline]
```

Hashes a string using Boost's hash function.

Parameters

key	The input string to hash.

Returns

The hashed string value.

Definition at line 19 of file Utils.h.

```
00020     {
00021         boost::hash<string> hash_fn;
00022         size_t hash = hash_fn(key);
00023         return to_string(hash);
00024     }
```

3.10.2.3 splitCommand()

```
vector< string > Utils::splitCommand (
            string command)  [inline]
```

Splits a command string into at most three parts (command, key, and value).

Parameters

command	The input command string.

Returns

A vector containing the split components (command, key, and optionally value).

Definition at line 32 of file Utils.h.

```
00033     {
00034         vector<string> result;
00035         string word = "";
00036         int count = 2;
00037         for (auto x : command)
00038         {
00039             if (x == '\0')
00040             {
00041                 break;
00042             }
00043
00044             if (x == ' ' && count > 0)
00045             {
00046                 result.push_back(word);
00047                 word = "";
00048                 count--;
00049             }
00050             else
00051             {
00052                 word = word + x;
00053             }
00054         }
00055         result.push_back(word);
00056         return result;
00057     }
```

3.10.2.4 startsWith()

```
bool Utils::startsWith (
            const string & str,
            const string & prefix)  [inline]
```

Checks if a given string starts with a specified prefix.

Parameters

str	The main string to check.
prefix	The prefix to compare.

Returns

true if the string starts with the prefix, false otherwise.

Definition at line 66 of file Utils.h.

```
00067     {
00068         return str.rfind(prefix, 0) == 0;
00069     }
```

3.10.2.5 toRESP2()

```
string Utils::toRESP2 (
            const string & command)  [inline]
```

Converts a string to Redis Serialization Protocol (RESP2) format.

Parameters

data	The input string.

Returns

The RESP2 formatted string.

Definition at line 77 of file Utils.h.

```
00078     {
00079         istringstream stream(command);
00080         vector<string> tokens;
00081         string word;
00082
00083         while (stream » word)
00084         {
00085             tokens.push_back(word);
00086         }
00087
00088         string result = "*" + to_string(tokens.size()) + "\r\n";
00089         for (const auto &token : tokens)
00090         {
00091             result += "$" + to_string(token.size()) + "\r\n" + token + "\r\n";
00092         }
00093
00094         return result;
00095     }
```

The documentation for this class was generated from the following file:

- Utils/Utils.h

Chapter 4

File Documentation

4.1 APIGateway/APIGateway.h File Reference

```
#include <bits/stdc++.h>
#include "../Models/Command.h"
#include "../Services/SetService.h"
#include "../Services/GetService.h"
#include "../Services/DelService.h"
#include "../Cache/Cache.h"
```

Classes

- class APIGateway

  Acts as an intermediary between clients and the BlinkDB storage system.

4.2 APIGateway.h

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <bits/stdc++.h>
00004 #include "../Models/Command.h"
00005 #include "../Services/SetService.h"
00006 #include "../Services/GetService.h"
00007 #include "../Services/DelService.h"
00008 #include "../Cache/Cache.h"
00009
00010 using namespace std;
00011
00019 class APIGateway
00020 {
00021 private:
00027     Cache L1Cache;
00028
00034     BlinkDB &blinkDB;
00035
00041     SetService setService;
00042
00048     GetService getService;
00049
00055     DelService delService;
00056
00057 public:
```

```
00066     APIGateway(BlinkDB &blinkDB) : blinkDB(blinkDB), setService(blinkDB), getService(blinkDB),
      delService(blinkDB)
00067     {
00068         L1Cache = Cache();
00069     }
00070
00084     string executeCommand(Command command)
00085     {
00086         if (command.getCommand() == "SET")
00087         {
00088             // Clear cache if size exceeds limit
00089             if (L1Cache.getSize() >= 100000000)
00090             {
00091                 L1Cache.clear();
00092             }
00093             // Store in cache and database
00094             L1Cache.set(command.getKey(), command.getValue());
00095             setService.set(command.getKey(), command.getValue());
00096             return "Set Success";
00097         }
00098         else if (command.getCommand() == "GET")
00099         {
00100             // Check cache first
00101             string cacheCheckResult = L1Cache.get(command.getKey());
00102             if (cacheCheckResult == "-1")
00103             {
00104                 return getService.get(command.getKey());
00105             }
00106             else
00107             {
00108                 return cacheCheckResult;
00109             }
00110         }
00111         else if (command.getCommand() == "DEL")
00112         {
00113             // Remove from cache and database
00114             L1Cache.del(command.getKey());
00115             delService.del(command.getKey());
00116             return "Deletion Success";
00117         }
00118         return "Invalid Command";
00119     }
00120 };
```

4.3 Cache/Cache.h File Reference

```
#include <bits/stdc++.h>
```

Classes

- class Cache

  Provides an in-memory key-value store.

4.4 Cache.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <bits/stdc++.h>
00003
00004 using namespace std;
00005
00013 class Cache
00014 {
00015 private:
00021     unordered_map<string, string> cache;
00022
00023 public:
00033     void set(string key, string value)
```

```
00034     {
00035         cache[key] = value;
00036     }
00037
00047     string get(string key)
00048     {
00049         if (cache.find(key) == cache.end())
00050         {
00051             return "-1";
00052         }
00053         return cache[key];
00054     }
00055
00063     void del(string key)
00064     {
00065         cache.erase(key);
00066     }
00067
00075     size_t getSize()
00076     {
00077         return cache.size();
00078     }
00079
00085     void clear()
00086     {
00087         cache.clear();
00088     }
00089 };
```

4.5 Client.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cstring>
#include <chrono>
#include <sstream>
```

Functions

- string toRESP2 (const string &command)

  Converts a string to Redis Serialization Protocol (RESP2) format.
- vector< string > fromRESP2 (const string &resp)

  Parses a RESP2-formatted string back to a normal string.
- vector< string > splitCommand (string command)

  Splits a command string into at most three parts (command, key, and value).
- int main (int argc, char *argv[])

  Main function to establish a connection with the BlinkDB server and send commands.

4.5.1 Function Documentation

4.5.1.1 fromRESP2()

```
vector< string > fromRESP2 (
            const string & resp)
```

Parses a RESP2-formatted string back to a normal string.

Parameters

resp	The RESP2-formatted string.

Returns

The extracted string data.

Definition at line 51 of file Client.cpp.

```
00052 {
00053     vector<string> result;
00054     istringstream stream(resp);
00055     string line;
00056
00057     getline(stream, line, '\r');
00058     if (line[0] != '*')
00059         return {}; // Must start with '*'
00060
00061     int numArgs = stoi(line.substr(1)); // Number of arguments
00062     stream.ignore(1);                   // Ignore '\n'
00063
00064     for (int i = 0; i < numArgs; i++)
00065     {
00066         getline(stream, line, '\r');
00067         if (line[0] != '$')
00068             return {}; // Must start with '$'
00069
00070         int len = stoi(line.substr(1)); // Get length of argument
00071         stream.ignore(1);               // Ignore '\n'
00072
00073         string arg(len, ' ');
00074         stream.read(&arg[0], len); // Read the argument
00075         result.push_back(arg);
00076
00077         stream.ignore(2); // Ignore '\r\n'
00078     }
00079
00080     return result;
00081 }
```

4.5.1.2 main()

```
int main (
              int argc,
              char * argv[])
```

Main function to establish a connection with the BlinkDB server and send commands.

Returns

int Returns 0 on successful execution, -1 if socket creation fails, and -2 if the connection fails.

Definition at line 120 of file Client.cpp.

```
00121 {
00122     cout « "Connecting to BlinkDB server..." « endl;
00123
00124     // Check if sufficient arguments are provided
00125     if (argc < 2)
00126     {
00127         cout « "Enter 0 for interactive mode and 1 for file mode in command line and a filename for
      file mode." « endl;
00128         cout « "Exiting BlinkDB: Closing server..." « endl;
00129         cout « "Exited" « endl;
00130         return 0;
00131     }
00132
00133     // Create a TCP socket
00134     int clientSocket = socket(AF_INET, SOCK_STREAM, 0);
00135     if (clientSocket == -1)
```

```
00136     {
00137         cerr « "Socket creation failed" « endl;
00138         return -1;
00139     }
00140 
00141     // Define server address
00142     sockaddr_in serverAddress;
00143     serverAddress.sin_family = AF_INET;
00144     serverAddress.sin_port = htons(5000);                    // Port number
00145     serverAddress.sin_addr.s_addr = inet_addr("127.0.0.1"); // Localhost
00146 
00147     // Attempt to connect to the BlinkDB server
00148     int connectionStatus = connect(clientSocket, (sockaddr *)&serverAddress, sizeof(serverAddress));
00149     if (connectionStatus == -1)
00150     {
00151         cerr « "Connection to BlinkDB failed." « endl;
00152         return -2;
00153     }
00154     cout « "Connected to BlinkDB server." « endl;
00155 
00156     string mode = string(argv[1]);
00157     string filename = string(argv[2] != NULL ? argv[2] : "");
00158 
00159     // Validate mode input
00160     if (mode != "0" && mode != "1")
00161     {
00162         cout « "Enter 0 for interactive mode and 1 for file mode" « endl;
00163         cout « "Exiting BlinkDB: Closing server..." « endl;
00164         cout « "Exited" « endl;
00165         return 0;
00166     }
00167 
00168     // Validate filename in file mode
00169     if (mode == "1" && filename == "")
00170     {
00171         cout « "Please provide a filename for the test file" « endl;
00172         cout « "Exiting BlinkDB: Closing server..." « endl;
00173         cout « "Exited" « endl;
00174         return 0;
00175     }
00176 
00177     // File mode execution
00178     if (mode == "1")
00179     {
00180         chrono::high_resolution_clock::time_point start = chrono::high_resolution_clock::now();
00181         ifstream testFile(filename);
00182         string line;
00183         cout « "Executing commands from " « filename « "... Please wait..." « endl;
00184         while (getline(testFile, line))
00185         {
00186             // Convert command to RESP2 format and send it
00187             struct timeval timeout;
00188             timeout.tv_sec = 5; // Set timeout to 5 seconds
00189             timeout.tv_usec = 0;
00190 
00191             setsockopt(clientSocket, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));
00192             string resp = toRESP2(line);
00193             send(clientSocket, resp.substr(0, resp.size() - 1).c_str(), resp.size(), 0);
00194 
00195             // Receive response from server
00196             char response[512];
00197             memset(response, 0, sizeof(response));
00198             recv(clientSocket, response, sizeof(response), 0);
00199         }
00200         chrono::high_resolution_clock::time_point end = chrono::high_resolution_clock::now();
00201         chrono::duration<double> elapsed = end - start;
00202         cout « "Time taken to execute all commands: " « elapsed.count() « "s" « endl;
00203     }
00204     // Interactive mode execution
00205     else if (mode == "0")
00206     {
00207         while (true)
00208         {
00209             cout « "User > ";
00210             string input;
00211             getline(cin, input);
00212 
00213             // Validate user input
00214             if (input.empty())
00215             {
00216                 cout « "Invalid Command\n";
00217                 continue;
00218             }
00219 
00220             // Send command to server
00221             struct timeval timeout;
00222             timeout.tv_sec = 5; // Set timeout to 5 seconds
```

```
00223             timeout.tv_usec = 0;
00224
00225             setsockopt(clientSocket, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));
00226             string resp = toRESP2(input);
00227             send(clientSocket, resp.substr(0, resp.size() - 1).c_str(), resp.size(), 0);
00228
00229             // Handle exit command
00230             if (input == "exit")
00231             {
00232                 close(clientSocket);
00233                 cout « "Exiting BlinkDB..." « endl;
00234                 break;
00235             }
00236
00237             // Receive and display server response
00238             char response[512];
00239             memset(response, 0, sizeof(response));
00240             recv(clientSocket, response, sizeof(response), 0);
00241             vector<string> responseStr = fromRESP2(response);
00242             cout « "Server > ";
00243             for (auto word : responseStr)
00244             {
00245                 cout « word « " ";
00246             }
00247             cout « endl;
00248         }
00249     }
00250
00251     return 0;
00252 }
```

4.5.1.3 splitCommand()

```
vector< string > splitCommand (
            string command)
```

Splits a command string into at most three parts (command, key, and value).

Parameters

command	The input command string.

Returns

A vector containing the split components (command, key, and optionally value).

Definition at line 89 of file Client.cpp.

```
00090 {
00091     vector<string> result;
00092     string word = "";
00093     int count = 2;
00094     for (auto x : command)
00095     {
00096         if (x == '\0')
00097         {
00098             break;
00099         }
00100
00101         if (x == ' ' && count > 0)
00102         {
00103             result.push_back(word);
00104             word = "";
00105             count--;
00106         }
00107         else
00108         {
00109             word = word + x;
00110         }
00111     }
00112     result.push_back(word);
00113     return result;
00114 }
```

4.5.1.4 toRESP2()

```
string toRESP2 (
            const string & command)
```

Converts a string to Redis Serialization Protocol (RESP2) format.

Parameters

data	The input string.

Returns

The RESP2 formatted string.

Definition at line 25 of file Client.cpp.

```
00026 {
00027     istringstream stream(command);
00028     vector<string> tokens;
00029     string word;
00030
00031     while (stream » word)
00032     {
00033         tokens.push_back(word);
00034     }
00035
00036     string result = "*" + to_string(tokens.size()) + "\r\n";
00037     for (const auto &token : tokens)
00038     {
00039         result += "$" + to_string(token.size()) + "\r\n" + token + "\r\n";
00040     }
00041
00042     return result;
00043 }
```

4.6 Client.cpp

Go to the documentation of this file.

```
00001
00005
00006 #include <iostream>
00007 #include <fstream>
00008 #include <string>
00009 #include <sys/socket.h>
00010 #include <netinet/in.h>
00011 #include <arpa/inet.h>
00012 #include <unistd.h>
00013 #include <cstring>
00014 #include <chrono>
00015 #include <sstream>
00016
00017 using namespace std;
00018
00025 string toRESP2(const string &command)
00026 {
00027     istringstream stream(command);
00028     vector<string> tokens;
00029     string word;
00030
00031     while (stream » word)
00032     {
00033         tokens.push_back(word);
00034     }
00035
00036     string result = "*" + to_string(tokens.size()) + "\r\n";
00037     for (const auto &token : tokens)
00038     {
00039         result += "$" + to_string(token.size()) + "\r\n" + token + "\r\n";
00040     }
```

```
00041
00042     return result;
00043 }
00044
00051 vector<string> fromRESP2(const string &resp)
00052 {
00053     vector<string> result;
00054     istringstream stream(resp);
00055     string line;
00056
00057     getline(stream, line, '\r');
00058     if (line[0] != '*')
00059         return {}; // Must start with '*'
00060
00061     int numArgs = stoi(line.substr(1)); // Number of arguments
00062     stream.ignore(1);                   // Ignore '\n'
00063
00064     for (int i = 0; i < numArgs; i++)
00065     {
00066         getline(stream, line, '\r');
00067         if (line[0] != '$')
00068             return {}; // Must start with '$'
00069
00070         int len = stoi(line.substr(1)); // Get length of argument
00071         stream.ignore(1);               // Ignore '\n'
00072
00073         string arg(len, ' ');
00074         stream.read(&arg[0], len); // Read the argument
00075         result.push_back(arg);
00076
00077         stream.ignore(2); // Ignore '\r\n'
00078     }
00079
00080     return result;
00081 }
00082
00089 vector<string> splitCommand(string command)
00090 {
00091     vector<string> result;
00092     string word = "";
00093     int count = 2;
00094     for (auto x : command)
00095     {
00096         if (x == '\0')
00097         {
00098             break;
00099         }
00100
00101         if (x == ' ' && count > 0)
00102         {
00103             result.push_back(word);
00104             word = "";
00105             count--;
00106         }
00107         else
00108         {
00109             word = word + x;
00110         }
00111     }
00112     result.push_back(word);
00113     return result;
00114 }
00115
00120 int main(int argc, char *argv[])
00121 {
00122     cout « "Connecting to BlinkDB server..." « endl;
00123
00124     // Check if sufficient arguments are provided
00125     if (argc < 2)
00126     {
00127         cout « "Enter 0 for interactive mode and 1 for file mode in command line and a filename for
      file mode." « endl;
00128         cout « "Exiting BlinkDB: Closing server..." « endl;
00129         cout « "Exited" « endl;
00130         return 0;
00131     }
00132
00133     // Create a TCP socket
00134     int clientSocket = socket(AF_INET, SOCK_STREAM, 0);
00135     if (clientSocket == -1)
00136     {
00137         cerr « "Socket creation failed" « endl;
00138         return -1;
00139     }
00140
00141     // Define server address
00142     sockaddr_in serverAddress;
```

```
00143     serverAddress.sin_family = AF_INET;
00144     serverAddress.sin_port = htons(5000);                    // Port number
00145     serverAddress.sin_addr.s_addr = inet_addr("127.0.0.1"); // Localhost
00146
00147     // Attempt to connect to the BlinkDB server
00148     int connectionStatus = connect(clientSocket, (sockaddr *)&serverAddress, sizeof(serverAddress));
00149     if (connectionStatus == -1)
00150     {
00151         cerr « "Connection to BlinkDB failed." « endl;
00152         return -2;
00153     }
00154     cout « "Connected to BlinkDB server." « endl;
00155
00156     string mode = string(argv[1]);
00157     string filename = string(argv[2] != NULL ? argv[2] : "");
00158
00159     // Validate mode input
00160     if (mode != "0" && mode != "1")
00161     {
00162         cout « "Enter 0 for interactive mode and 1 for file mode" « endl;
00163         cout « "Exiting BlinkDB: Closing server..." « endl;
00164         cout « "Exited" « endl;
00165         return 0;
00166     }
00167
00168     // Validate filename in file mode
00169     if (mode == "1" && filename == "")
00170     {
00171         cout « "Please provide a filename for the test file" « endl;
00172         cout « "Exiting BlinkDB: Closing server..." « endl;
00173         cout « "Exited" « endl;
00174         return 0;
00175     }
00176
00177     // File mode execution
00178     if (mode == "1")
00179     {
00180         chrono::high_resolution_clock::time_point start = chrono::high_resolution_clock::now();
00181         ifstream testFile(filename);
00182         string line;
00183         cout « "Executing commands from " « filename « "... Please wait..." « endl;
00184         while (getline(testFile, line))
00185         {
00186             // Convert command to RESP2 format and send it
00187             struct timeval timeout;
00188             timeout.tv_sec = 5; // Set timeout to 5 seconds
00189             timeout.tv_usec = 0;
00190
00191             setsockopt(clientSocket, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));
00192             string resp = toRESP2(line);
00193             send(clientSocket, resp.substr(0, resp.size() - 1).c_str(), resp.size(), 0);
00194
00195             // Receive response from server
00196             char response[512];
00197             memset(response, 0, sizeof(response));
00198             recv(clientSocket, response, sizeof(response), 0);
00199         }
00200         chrono::high_resolution_clock::time_point end = chrono::high_resolution_clock::now();
00201         chrono::duration<double> elapsed = end - start;
00202         cout « "Time taken to execute all commands: " « elapsed.count() « "s" « endl;
00203     }
00204     // Interactive mode execution
00205     else if (mode == "0")
00206     {
00207         while (true)
00208         {
00209             cout « "User > ";
00210             string input;
00211             getline(cin, input);
00212
00213             // Validate user input
00214             if (input.empty())
00215             {
00216                 cout « "Invalid Command\n";
00217                 continue;
00218             }
00219
00220             // Send command to server
00221             struct timeval timeout;
00222             timeout.tv_sec = 5; // Set timeout to 5 seconds
00223             timeout.tv_usec = 0;
00224
00225             setsockopt(clientSocket, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));
00226             string resp = toRESP2(input);
00227             send(clientSocket, resp.substr(0, resp.size() - 1).c_str(), resp.size(), 0);
00228
00229             // Handle exit command
```

```
00230             if (input == "exit")
00231             {
00232                 close(clientSocket);
00233                 cout « "Exiting BlinkDB..." « endl;
00234                 break;
00235             }
00236 
00237             // Receive and display server response
00238             char response[512];
00239             memset(response, 0, sizeof(response));
00240             recv(clientSocket, response, sizeof(response), 0);
00241             vector<string> responseStr = fromRESP2(response);
00242             cout « "Server > ";
00243             for (auto word : responseStr)
00244             {
00245                 cout « word « " ";
00246             }
00247             cout « endl;
00248         }
00249     }
00250 
00251     return 0;
00252 }
```

4.7 Database/BlinkDB.h File Reference

```
#include <bits/stdc++.h>
#include "../Handlers/DiscBackupHandler.h"
```

Classes

- class BlinkDB

  Implements an in-memory key-value database with periodic disk backups.

4.8 BlinkDB.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <bits/stdc++.h>
00003 #include "../Handlers/DiscBackupHandler.h"
00004 
00005 using namespace std;
00006 
00014 class BlinkDB
00015 {
00016 private:
00020     unordered_map<string, string> database;
00021 
00025     unordered_map<string, string> buffer;
00026 
00030     DiscBackupHandler discBackupHandler;
00031 
00035     mutex dbMutex;
00036 
00040     mutex bufferMutex;
00041 
00045     thread backupThread;
00046 
00050     bool stopBackup = false;
00051 
00058     void backupWorker()
00059     {
00060         while (!stopBackup)
00061         {
00062             this_thread::sleep_for(chrono::seconds(5));
00063             if (database.size() > 100000000)
00064             {
00065                 dbMutex.lock();
```

```
00066                 buffer = database;
00067                 dbMutex.unlock();
00068
00069                 bufferMutex.lock();
00070                 performBackup();
00071                 buffer.clear();
00072                 bufferMutex.unlock();
00073             }
00074         }
00075     }
00076
00084     bool performBackup()
00085     {
00086         discBackupHandler.backup(database);
00087         discBackupHandler.commitBackup();
00088         return true;
00089     }
00090
00091 public:
00097     BlinkDB()
00098     {
00099         DiscBackupHandler discBackupHandler = DiscBackupHandler();
00100         database = unordered_map<string, string>();
00101         backupThread = thread(&BlinkDB::backupWorker, this);
00102         buffer = unordered_map<string, string>();
00103     }
00104
00110     ~BlinkDB()
00111     {
00112         stopBackup = true;
00113         if (backupThread.joinable())
00114             backupThread.join();
00115     }
00116
00125     void set(string key, string value)
00126     {
00127         database[key] = value;
00128     }
00129
00139     string get(string key)
00140     {
00141         if (database.find(key) != database.end())
00142         {
00143             return database[key];
00144         }
00145         return discBackupHandler.checkBackupForKey(key);
00146     }
00147
00155     void del(string key)
00156     {
00157         database.erase(key);
00158     }
00159 };
```

4.9 Handlers/DiscBackupHandler.h File Reference

```
#include <bits/stdc++.h>
#include <mutex>
#include <filesystem>
#include "../Utils/Utils.h"
```

Classes

- class DiscBackupHandler

  The DiscBackupHandler class handles disk-based backups for BlinkDB.

4.10 DiscBackupHandler.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <bits/stdc++.h>
00003 #include <mutex>
00004 #include <filesystem>
00005 #include "../Utils/Utils.h"
00006
00007 using namespace std;
00008
00016 class DiscBackupHandler
00017 {
00018 private:
00024     string backupFiles[10] = {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"};
00025
00029     Utils utils;
00030
00034     mutex backupMutex;
00035
00042     bool backupFromMapToBufferFile(const unordered_map<string, string> &map)
00043     {
00044         lock_guard<mutex> lock(backupMutex);
00045         ofstream file("./backups/backup.txt", ios::app);
00046         if (!file.is_open())
00047         {
00048             return false;
00049         }
00050         for (const auto &entry : map)
00051         {
00052             file « entry.first « " " « entry.second « endl;
00053         }
00054         return true;
00055     }
00056
00065     bool backupFromBufferFileToDisc()
00066     {
00067         lock_guard<mutex> lock(backupMutex);
00068         ifstream backupBuffer("./backups/backup.txt");
00069         if (!backupBuffer.is_open())
00070         {
00071             return false;
00072         }
00073
00074         vector<string> lines;
00075         string line;
00076         while (getline(backupBuffer, line))
00077         {
00078             lines.push_back(line);
00079         }
00080         backupBuffer.close();
00081
00082         sort(lines.begin(), lines.end());
00083
00084         for (int i = 0; i < 10; i++)
00085         {
00086             ofstream file("./backups/" + backupFiles[i] + ".txt", ios::out);
00087             if (!file.is_open())
00088             {
00089                 return false;
00090             }
00091             for (const string &entry : lines)
00092             {
00093                 if (utils.startsWith(entry, backupFiles[i]))
00094                 {
00095                     file « entry « endl;
00096                 }
00097             }
00098         }
00099
00100         // Clear buffer file
00101         ofstream clearBuffer("./backups/backup.txt", ios::out);
00102         return true;
00103     }
00104
00105 public:
00111     DiscBackupHandler()
00112     {
00113         std::filesystem::create_directories("./backups");
00114     }
00115
00122     bool backup(const unordered_map<string, string> &map)
00123     {
00124         return backupFromMapToBufferFile(map);
00125     }
```

```
00126
00135     bool commitBackup()
00136     {
00137         return backupFromBufferFileToDisc();
00138     }
00139
00145     bool terminate()
00146     {
00147         lock_guard<mutex> lock(backupMutex);
00148         std::filesystem::remove_all("./backups");
00149         return true;
00150     }
00151
00161     string checkBackupForKey(const string &key)
00162     {
00163         lock_guard<mutex> lock(backupMutex);
00164         string filename = key.substr(0, 1) + ".txt";
00165         ifstream backupBuffer("./backups/" + filename);
00166         if (!backupBuffer.is_open())
00167         {
00168             return "-2";
00169         }
00170
00171         string line, foundValue = "-1";
00172         while (getline(backupBuffer, line))
00173         {
00174             if (utils.startsWith(line, key))
00175             {
00176                 size_t pos = line.find(" ");
00177                 if (pos != string::npos)
00178                 {
00179                     foundValue = line.substr(pos + 1);
00180                 }
00181                 break;
00182             }
00183         }
00184         return foundValue;
00185     }
00186 };
```

4.11 Models/Command.h File Reference

```
#include <string>
```

Classes

- class Command

  Represents a user command in BlinkDB.

4.12 Command.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <string>
00003
00004 using namespace std;
00005
00011 class Command
00012 {
00013 private:
00017     string command;
00018
00022     string key;
00023
00027     string value;
00028
00029 public:
00033     Command()
```

```
00034     {
00035         command = "";
00036         key = "";
00037         value = "";
00038     }
00039
00047     Command(string command, string key, string value)
00048     {
00049         this->command = command;
00050         this->key = key;
00051         this->value = value;
00052     }
00053
00060     Command(string command, string key)
00061     {
00062         this->command = command;
00063         this->key = key;
00064         this->value = "";
00065     }
00066
00072     string getCommand()
00073     {
00074         return command;
00075     }
00076
00082     string getKey()
00083     {
00084         return key;
00085     }
00086
00092     string getValue()
00093     {
00094         return value;
00095     }
00096 };
```

4.13 Models/Response.h File Reference

```
#include <bits/stdc++.h>
```

Classes

- class Response

  Represents an API response in BlinkDB.

4.14 Response.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <bits/stdc++.h>
00003
00004 using namespace std;
00005
00011 class Response
00012 {
00013 private:
00017     int statusCode;
00018
00022     string message;
00023
00027     pair<string, string> data;
00028
00029 public:
00033     Response()
00034     {
00035         statusCode = 0;
00036         message = "";
00037         data = pair<string, string>("Data", "");
```

```
00038     }
00039
00047     Response(int statusCode, string message, pair<string, string> data)
00048     {
00049         this->statusCode = statusCode;
00050         this->message = message;
00051         this->data = data;
00052     }
00053
00062     Response(int statusCode, string key, string value, string message)
00063     {
00064         this->statusCode = statusCode;
00065         this->message = message;
00066         this->data = pair<string, string>(key, value);
00067     }
00068
00074     string to_string()
00075     {
00076         return "Status Code: " + std::to_string(statusCode) + ", Message: " + message + ", Data: " +
      data.second;
00077     }
00078
00079     string getValue()
00080     {
00081         if (data.first == "Data")
00082         {
00083             return data.second;
00084         }
00085         return "-1";
00086     }
00087 };
```

4.15 REPL.cpp File Reference

```
#include <bits/stdc++.h>
#include <iostream>
#include <fstream>
#include <atomic>
#include "./APIGateway/APIGateway.h"
#include "./Models/Response.h"
```

Functions

- void signalHandler (int signal)

  Handles termination signals (e.g., Ctrl+C).
- int executeCommand (string input, string mode)

  Parses and executes a given command.
- void REPL (string mode, string filename="")

  Read-Eval-Print Loop (REPL) for processing user commands.
- int main (int argc, char ∗argv[ ])

  Entry point of the BlinkDB server.

Variables

- BlinkDB blinkDB

  Represents the main database instance.
- Command command

  Stores the command to be executed.
- APIGateway apiGateway (blinkDB)

  Handles API requests for executing database commands.

- DiscBackupHandler discBackupHandler

  Manages disk backup operations.
- Utils utils

  Utility class for helper functions.
- mutex dbMutex

  Mutex to synchronize access to the database.

4.15.1 Function Documentation

4.15.1.1 executeCommand()

```
int executeCommand (
            string input,
            string mode)
```

Parses and executes a given command.

This function tokenizes the input command, determines its validity, executes the appropriate API request, and returns the response.

Parameters

input	The command entered by the user.
mode	Execution mode (interactive or batch file execution).

Returns

int Returns -1 for exit, 0 for invalid command, otherwise continues execution.

Definition at line 68 of file REPL.cpp.

```
00069 {
00070     vector<string> result = utils.splitCommand(input);
00071
00072     // Process the command based on its type
00073     if (result.size() == 3 && result[0] == "SET")
00074     {
00075         command = Command(result[0], result[1], result[2]);
00076     }
00077     else if (result.size() == 2 && result[0] == "GET")
00078     {
00079         command = Command(result[0], result[1]);
00080     }
00081     else if (result.size() == 2 && result[0] == "DEL")
00082     {
00083         command = Command(result[0], result[1]);
00084     }
00085     else if (result[0] == "EXIT")
00086     {
00087         return -1;
00088     }
00089     else
00090     {
00091         cout « "Invalid command" « endl;
00092         return 0;
00093     }
00094
00095     // Execute the command and retrieve the response
00096     string apiResponse;
00097     {
00098         lock_guard<mutex> lock(dbMutex); // Ensures thread safety while accessing the database
00099         apiResponse = apiGateway.executeCommand(command);
00100     }
00101
00102     // Construct and print the response
```

```
00103     Response response;
00104     if (apiResponse == "-1" || apiResponse == "-2")
00105     {
00106         response = Response(404, "Not Found", {"Data", "Key not found"});
00107     }
00108     else
00109     {
00110         response = Response(200, "Success", {"Data", apiResponse});
00111     }
00112
00113     // Print response based on mode
00114     if (mode == "1" && command.getCommand() == "GET")
00115     {
00116         cout « response.getValue() « endl;
00117     }
00118     else if (mode == "0")
00119     {
00120         cout « "Response: " « response.to_string() « endl;
00121     }
00122     return 0;
00123 }
```

4.15.1.2 main()

```
int main (
              int argc,
              char * argv[])
```

Entry point of the BlinkDB server.

This function initializes the server, sets up a signal handler for termination, starts the REPL loop, and gracefully shuts down the system.

Returns

int Exit status code.

Definition at line 174 of file REPL.cpp.

```
00175 {
00176     cout « "Initializing BlinkDB server..." « endl;
00177
00178     // Register signal handler for graceful termination
00179     signal(SIGINT, signalHandler);
00180     if (argc < 2)
00181     {
00182         cout « "Enter 0 for interactive mode and 1 for file mode in command line and a filename for
      file mode." « endl;
00183         discBackupHandler.terminate();
00184         cout « "Exiting BlinkDB: Closing server..." « endl;
00185         cout « "Exited" « endl;
00186         return 0;
00187     }
00188     string mode = string(argv[1]);
00189     string filename = string(argv[2] != NULL ? argv[2] : "");
00190     if (mode != "0" && mode != "1")
00191     {
00192         cout « "Enter 0 for interactive mode and 1 for file mode" « endl;
00193         discBackupHandler.terminate();
00194         cout « "Exiting BlinkDB: Closing server..." « endl;
00195         cout « "Exited" « endl;
00196         return 0;
00197     }
00198
00199     if (mode == "1" && filename == "")
00200     {
00201         cout « "Please provide a filename for the test file" « endl;
00202         discBackupHandler.terminate();
00203         cout « "Exiting BlinkDB: Closing server..." « endl;
00204         cout « "Exited" « endl;
00205         return 0;
00206     }
00207     // Start the Read-Eval-Print Loop
00208     REPL(mode, filename);
00209
00210     // Cleanup before exiting
00211     discBackupHandler.terminate();
00212     cout « "Exiting BlinkDB: Closing server..." « endl;
00213     cout « "Exited" « endl;
00214
00215     return 0;
00216 }
```

4.15.1.3 REPL()

```
void REPL (
            string mode,
            string filename = "")
```

Read-Eval-Print Loop (REPL) for processing user commands.

This function continuously prompts the user for input, parses the command, executes it via the API Gateway, and prints the response. It supports both interactive and batch (file-based) execution modes.

Parameters

mode	Execution mode ("0" for interactive, "1" for batch processing).
filename	Optional filename for batch execution.

Definition at line 135 of file REPL.cpp.

```
00136 {
00137     if (mode == "1")
00138     {
00139         chrono::high_resolution_clock::time_point start = chrono::high_resolution_clock::now();
00140         ifstream testFile(filename);
00141         string line;
00142         while (getline(testFile, line))
00143         {
00144             executeCommand(line, mode);
00145         }
00146         chrono::high_resolution_clock::time_point end = chrono::high_resolution_clock::now();
00147         chrono::duration<double> elapsed = end - start;
00148         cout « "Time taken to execute all commands: " « elapsed.count() « "s" « endl;
00149     }
00150     else if (mode == "0")
00151     {
00152         while (true)
00153         {
00154             cout « "User > ";
00155             string input;
00156             getline(cin, input);
00157             int result = executeCommand(input, mode);
00158             if (result == -1)
00159             {
00160                 break;
00161             }
00162         }
00163     }
00164 }
```

4.15.1.4 signalHandler()

```
void signalHandler (
            int signal)
```

Handles termination signals (e.g., Ctrl+C).

This function ensures a graceful shutdown of BlinkDB by cleaning up backups and closing the server safely before exiting the program.

Parameters

signal	The received signal code.

Definition at line 48 of file REPL.cpp.

```
00049 {
00050     cout « "Exiting BlinkDB: Deleting Backups..." « endl;
00051     discBackupHandler.terminate();
00052     cout « "Exiting BlinkDB: Deleting Backups... Done" « endl;
00053     cout « "Exiting BlinkDB: Closing server..." « endl;
00054     cout « "Exited" « endl;
00055     exit(0);
00056 }
```

4.15.2 Variable Documentation

4.15.2.1 apiGateway

```
APIGateway apiGateway(blinkDB) (
            blinkDB )
```

Handles API requests for executing database commands.

4.15.2.2 blinkDB

```
BlinkDB blinkDB
```

Represents the main database instance.

Definition at line 13 of file REPL.cpp.

4.15.2.3 command

```
Command command
```

Stores the command to be executed.

Definition at line 18 of file REPL.cpp.

4.15.2.4 dbMutex

```
mutex dbMutex
```

Mutex to synchronize access to the database.

Definition at line 38 of file REPL.cpp.

4.15.2.5 discBackupHandler

```
DiscBackupHandler discBackupHandler
```

Manages disk backup operations.

Definition at line 28 of file REPL.cpp.

4.15.2.6 utils

```
Utils utils
```

Utility class for helper functions.

Definition at line 33 of file REPL.cpp.

4.16 REPL.cpp

Go to the documentation of this file.

```
00001 #include <bits/stdc++.h>
00002 #include <iostream>
00003 #include <fstream>
00004 #include <atomic>
00005 #include "./APIGateway/APIGateway.h"
00006 #include "./Models/Response.h"
00007
00008 using namespace std;
00009
00013 BlinkDB blinkDB;
00014
00018 Command command;
00019
00023 APIGateway apiGateway(blinkDB);
00024
00028 DiscBackupHandler discBackupHandler;
00029
00033 Utils utils;
00034
00038 mutex dbMutex;
00039
00048 void signalHandler(int signal)
00049 {
00050     cout « "Exiting BlinkDB: Deleting Backups..." « endl;
00051     discBackupHandler.terminate();
00052     cout « "Exiting BlinkDB: Deleting Backups... Done" « endl;
00053     cout « "Exiting BlinkDB: Closing server..." « endl;
00054     cout « "Exited" « endl;
00055     exit(0);
00056 }
00057
00068 int executeCommand(string input, string mode)
00069 {
00070     vector<string> result = utils.splitCommand(input);
00071
00072     // Process the command based on its type
00073     if (result.size() == 3 && result[0] == "SET")
00074     {
00075         command = Command(result[0], result[1], result[2]);
00076     }
00077     else if (result.size() == 2 && result[0] == "GET")
00078     {
00079         command = Command(result[0], result[1]);
00080     }
00081     else if (result.size() == 2 && result[0] == "DEL")
00082     {
00083         command = Command(result[0], result[1]);
00084     }
00085     else if (result[0] == "EXIT")
00086     {
00087         return -1;
00088     }
00089     else
00090     {
00091         cout « "Invalid command" « endl;
00092         return 0;
00093     }
00094
00095     // Execute the command and retrieve the response
00096     string apiResponse;
00097     {
00098         lock_guard<mutex> lock(dbMutex); // Ensures thread safety while accessing the database
00099         apiResponse = apiGateway.executeCommand(command);
00100     }
00101
00102     // Construct and print the response
00103     Response response;
00104     if (apiResponse == "-1" || apiResponse == "-2")
00105     {
00106         response = Response(404, "Not Found", {"Data", "Key not found"});
00107     }
00108     else
00109     {
00110         response = Response(200, "Success", {"Data", apiResponse});
00111     }
00112
00113     // Print response based on mode
00114     if (mode == "1" && command.getCommand() == "GET")
00115     {
00116         cout « response.getValue() « endl;
00117     }
00118     else if (mode == "0")
```

```
00119     {
00120         cout « "Response: " « response.to_string() « endl;
00121     }
00122     return 0;
00123 }
00124
00135 void REPL(string mode, string filename = "")
00136 {
00137     if (mode == "1")
00138     {
00139         chrono::high_resolution_clock::time_point start = chrono::high_resolution_clock::now();
00140         ifstream testFile(filename);
00141         string line;
00142         while (getline(testFile, line))
00143         {
00144             executeCommand(line, mode);
00145         }
00146         chrono::high_resolution_clock::time_point end = chrono::high_resolution_clock::now();
00147         chrono::duration<double> elapsed = end - start;
00148         cout « "Time taken to execute all commands: " « elapsed.count() « "s" « endl;
00149     }
00150     else if (mode == "0")
00151     {
00152         while (true)
00153         {
00154             cout « "User > ";
00155             string input;
00156             getline(cin, input);
00157             int result = executeCommand(input, mode);
00158             if (result == -1)
00159             {
00160                 break;
00161             }
00162         }
00163     }
00164 }
00165
00174 int main(int argc, char *argv[])
00175 {
00176     cout « "Initializing BlinkDB server..." « endl;
00177
00178     // Register signal handler for graceful termination
00179     signal(SIGINT, signalHandler);
00180     if (argc < 2)
00181     {
00182         cout « "Enter 0 for interactive mode and 1 for file mode in command line and a filename for
      file mode." « endl;
00183         discBackupHandler.terminate();
00184         cout « "Exiting BlinkDB: Closing server..." « endl;
00185         cout « "Exited" « endl;
00186         return 0;
00187     }
00188     string mode = string(argv[1]);
00189     string filename = string(argv[2] != NULL ? argv[2] : "");
00190     if (mode != "0" && mode != "1")
00191     {
00192         cout « "Enter 0 for interactive mode and 1 for file mode" « endl;
00193         discBackupHandler.terminate();
00194         cout « "Exiting BlinkDB: Closing server..." « endl;
00195         cout « "Exited" « endl;
00196         return 0;
00197     }
00198
00199     if (mode == "1" && filename == "")
00200     {
00201         cout « "Please provide a filename for the test file" « endl;
00202         discBackupHandler.terminate();
00203         cout « "Exiting BlinkDB: Closing server..." « endl;
00204         cout « "Exited" « endl;
00205         return 0;
00206     }
00207     // Start the Read-Eval-Print Loop
00208     REPL(mode, filename);
00209
00210     // Cleanup before exiting
00211     discBackupHandler.terminate();
00212     cout « "Exiting BlinkDB: Closing server..." « endl;
00213     cout « "Exited" « endl;
00214
00215     return 0;
00216 }
```

4.17 Server.cpp File Reference

```
#include <bits/stdc++.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cstring>
#include <thread>
#include <atomic>
#include <csignal>
#include "./APIGateway/APIGateway.h"
#include "./Models/Response.h"
```

Functions

- atomic< int > activeConnections (0)
- void closeServer ()
- void signalHandler (int signal)
- void handleClient (int clientSocket, string mode)
- int main (int argc, char *argv[])

Variables

- int serverSocket
- BlinkDB blinkDB
- Command command
- APIGateway apiGateway (blinkDB)
- DiscBackupHandler discBackupHandler
- Utils utils
- mutex dbMutex
- mutex sendMutex

4.17.1 Function Documentation

4.17.1.1 activeConnections()

```
atomic< int > activeConnections (
            0 )
```

4.17.1.2 closeServer()

```
void closeServer ()
```

Definition at line 25 of file Server.cpp.

```
00026 {
00027     close(serverSocket);
00028 }
```

4.17.1.3 handleClient()

```
void handleClient (
            int clientSocket,
            string mode)
```

Definition at line 41 of file Server.cpp.

```
00042 {
00043     {
00044         lock_guard<mutex> lock(sendMutex);
00045         activeConnections++;
00046     }
00047
00048     char buffer[512];
00049     while (true)
00050     {
00051         memset(buffer, 0, sizeof(buffer));
00052         int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
00053         if (bytesReceived <= 0)
00054         {
00055             {
00056                 lock_guard<mutex> lock(sendMutex);
00057                 --activeConnections;
00058             }
00059             break;
00060         }
00061
00062         vector<string> result = utils.fromRESP2(buffer);
00063         if (result.empty())
00064             continue;
00065
00066         if (result[0] == "CONFIG")
00067             result.erase(result.begin());
00068
00069         string apiResponse;
00070         Response response;
00071
00072         Command command; // Fix: Use local variable instead of global command
00073
00074         if (result.size() == 3 && result[0] == "SET")
00075         {
00076             command = Command(result[0], result[1], result[2]);
00077         }
00078         else if (result.size() == 2 && result[0] == "GET")
00079         {
00080             command = Command(result[0], result[1]);
00081         }
00082         else if (result.size() == 2 && result[0] == "DEL")
00083         {
00084             command = Command(result[0], result[1]);
00085         }
00086         else if (result[0] == "EXIT")
00087         {
00088             break;
00089         }
00090         else if (result[0] == "PING")
00091         {
00092             response = Response(200, "Success", {"Data", "PONG"});
00093         }
00094         else
00095         {
00096             response = Response(400, "Bad Request", {"Data", "Invalid command"});
00097         }
00098
00099         if (result[0] != "PING")
00100         {
00101             {
00102                 lock_guard<mutex> lock(dbMutex);
00103                 apiResponse = apiGateway.executeCommand(command);
00104             }
00105             if (mode == "0")
00106             {
00107                 apiResponse = "+OK\r\n";
00108                 memset(buffer, 0, sizeof(buffer));
00109
00110                 // Fix: Use strncpy to avoid buffer overflow
00111                 strncpy(buffer, apiResponse.c_str(), sizeof(buffer) - 1);
00112                 buffer[sizeof(buffer) - 1] = '\0';
00113
00114                 {
00115                     lock_guard<mutex> lock(sendMutex);
00116                     send(clientSocket, buffer, strlen(buffer), 0);
00117                 }
```

```
00118             }
00119             else
00120             {
00121                 if (apiResponse == "-1" || apiResponse == "-2")
00122                 {
00123                     response = Response(404, "Not Found", {"Data", "Key not found"});
00124                 }
00125                 else
00126                 {
00127                     response = Response(200, "Success", {"Data", apiResponse});
00128                 }
00129                 memset(buffer, 0, sizeof(buffer));
00130                 string temp = utils.toRESP2(response.to_string());
00131                 temp = temp.substr(0, temp.size() - 1);
00132 
00133                 // Fix: Use strncpy
00134                 strncpy(buffer, temp.c_str(), sizeof(buffer) - 1);
00135                 buffer[sizeof(buffer) - 1] = '\0';
00136 
00137                 {
00138                     lock_guard<mutex> lock(sendMutex);
00139                     send(clientSocket, buffer, strlen(buffer), 0);
00140                 }
00141             }
00142         }
00143     }
00144 
00145     close(clientSocket);
00146 }
```

4.17.1.4 main()

```
int main (
            int argc,
            char * argv[])
```

Definition at line 148 of file Server.cpp.

```
00149 {
00150     cout « "Initializing BlinkDB server..." « endl;
00151     signal(SIGINT, signalHandler);
00152 
00153     if (argc < 2)
00154     {
00155         cout « "Enter 1 for Client-server mode and 0 for redis-benchmark mode" « endl;
00156         cout « "Exiting BlinkDB: Closing server..." « endl;
00157         cout « "Exited" « endl;
00158         discBackupHandler.terminate();
00159         return 0;
00160     }
00161 
00162     serverSocket = socket(AF_INET, SOCK_STREAM, 0);
00163     if (serverSocket == -1)
00164     {
00165         cerr « "Socket creation failed" « endl;
00166         return -1;
00167     }
00168 
00169     sockaddr_in serverAddress;
00170     serverAddress.sin_family = AF_INET;
00171     serverAddress.sin_port = htons(5000);
00172     serverAddress.sin_addr.s_addr = INADDR_ANY;
00173 
00174     int bindStatus = bind(serverSocket, (sockaddr *)&serverAddress, sizeof(serverAddress));
00175     if (bindStatus == -1)
00176     {
00177         cerr « "Binding to port 5000 failed." « endl;
00178         return -2;
00179     }
00180 
00181     int listenStatus = listen(serverSocket, 100000);
00182     if (listenStatus == -1)
00183     {
00184         cerr « "Listening on port 5000 failed." « endl;
00185         return -3;
00186     }
00187 
00188     cout « "Initialized BlinkDB server." « endl;
00189     cout « "Listening on port 5000..." « endl;
00190     cout « "Press Ctrl+C to exit." « endl;
00191 
```

```
00192     while (true)
00193     {
00194         sockaddr_in clientAddress;
00195         socklen_t clientAddressSize = sizeof(clientAddress);
00196         int clientSocket = accept(serverSocket, (sockaddr *)&clientAddress, &clientAddressSize);
00197         if (clientSocket == -1)
00198         {
00199             cerr « "Accepting connection failed." « endl;
00200             continue;
00201         }
00202 
00203         thread(handleClient, clientSocket, argv[1]).detach();
00204     }
00205 
00206     close(serverSocket);
00207     return 0;
00208 }
```

4.17.1.5 signalHandler()

```
void signalHandler (
             int signal)
```

Definition at line 30 of file Server.cpp.

```
00031 {
00032     cout « "Exiting BlinkDB: Deleting Backups..." « endl;
00033     discBackupHandler.terminate();
00034     cout « "Exiting BlinkDB: Deleting Backups... Done" « endl;
00035     cout « "Exiting BlinkDB: Closing server..." « endl;
00036     cout « "Exited" « endl;
00037     closeServer();
00038     exit(signal);
00039 }
```

4.17.2 Variable Documentation

4.17.2.1 apiGateway

```
APIGateway apiGateway(blinkDB) (
             blinkDB )
```

4.17.2.2 blinkDB

```
BlinkDB blinkDB
```

Definition at line 16 of file Server.cpp.

4.17.2.3 command

```
Command command
```

Definition at line 17 of file Server.cpp.

4.17.2.4 dbMutex

```
mutex dbMutex
```

Definition at line 22 of file Server.cpp.

4.17.2.5 discBackupHandler

```
DiscBackupHandler discBackupHandler
```

Definition at line 19 of file Server.cpp.

4.17.2.6 sendMutex

```
mutex sendMutex
```

Definition at line 23 of file Server.cpp.

4.17.2.7 serverSocket

```
int serverSocket
```

Definition at line 15 of file Server.cpp.

4.17.2.8 utils

```
Utils utils
```

Definition at line 20 of file Server.cpp.

4.18 Server.cpp

Go to the documentation of this file.

```
00001 #include <bits/stdc++.h>
00002 #include <sys/socket.h>
00003 #include <netinet/in.h>
00004 #include <arpa/inet.h>
00005 #include <unistd.h>
00006 #include <cstring>
00007 #include <thread>
00008 #include <atomic>
00009 #include <csignal>
00010 #include "./APIGateway/APIGateway.h"
00011 #include "./Models/Response.h"
00012
00013 using namespace std;
00014
00015 int serverSocket;
00016 BlinkDB blinkDB;
00017 Command command;
00018 APIGateway apiGateway(blinkDB);
00019 DiscBackupHandler discBackupHandler;
00020 Utils utils;
00021 atomic<int> activeConnections(0);
00022 mutex dbMutex;
00023 mutex sendMutex;
00024
00025 void closeServer()
00026 {
00027     close(serverSocket);
00028 }
00029
00030 void signalHandler(int signal)
00031 {
00032     cout « "Exiting BlinkDB: Deleting Backups..." « endl;
00033     discBackupHandler.terminate();
00034     cout « "Exiting BlinkDB: Deleting Backups... Done" « endl;
```

```
00035     cout « "Exiting BlinkDB: Closing server..." « endl;
00036     cout « "Exited" « endl;
00037     closeServer();
00038     exit(signal);
00039 }
00040
00041 void handleClient(int clientSocket, string mode)
00042 {
00043     {
00044         lock_guard<mutex> lock(sendMutex);
00045         activeConnections++;
00046     }
00047
00048     char buffer[512];
00049     while (true)
00050     {
00051         memset(buffer, 0, sizeof(buffer));
00052         int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
00053         if (bytesReceived <= 0)
00054         {
00055             {
00056                 lock_guard<mutex> lock(sendMutex);
00057                 --activeConnections;
00058             }
00059             break;
00060         }
00061
00062         vector<string> result = utils.fromRESP2(buffer);
00063         if (result.empty())
00064             continue;
00065
00066         if (result[0] == "CONFIG")
00067             result.erase(result.begin());
00068
00069         string apiResponse;
00070         Response response;
00071
00072         Command command; // Fix: Use local variable instead of global command
00073
00074         if (result.size() == 3 && result[0] == "SET")
00075         {
00076             command = Command(result[0], result[1], result[2]);
00077         }
00078         else if (result.size() == 2 && result[0] == "GET")
00079         {
00080             command = Command(result[0], result[1]);
00081         }
00082         else if (result.size() == 2 && result[0] == "DEL")
00083         {
00084             command = Command(result[0], result[1]);
00085         }
00086         else if (result[0] == "EXIT")
00087         {
00088             break;
00089         }
00090         else if (result[0] == "PING")
00091         {
00092             response = Response(200, "Success", {"Data", "PONG"});
00093         }
00094         else
00095         {
00096             response = Response(400, "Bad Request", {"Data", "Invalid command"});
00097         }
00098
00099         if (result[0] != "PING")
00100         {
00101             {
00102                 lock_guard<mutex> lock(dbMutex);
00103                 apiResponse = apiGateway.executeCommand(command);
00104             }
00105             if (mode == "0")
00106             {
00107                 apiResponse = "+OK\r\n";
00108                 memset(buffer, 0, sizeof(buffer));
00109
00110                 // Fix: Use strncpy to avoid buffer overflow
00111                 strncpy(buffer, apiResponse.c_str(), sizeof(buffer) - 1);
00112                 buffer[sizeof(buffer) - 1] = '\0';
00113
00114                 {
00115                     lock_guard<mutex> lock(sendMutex);
00116                     send(clientSocket, buffer, strlen(buffer), 0);
00117                 }
00118             }
00119             else
00120             {
00121                 if (apiResponse == "-1" || apiResponse == "-2")
```

```
00122                 {
00123                     response = Response(404, "Not Found", {"Data", "Key not found"});
00124                 }
00125                 else
00126                 {
00127                     response = Response(200, "Success", {"Data", apiResponse});
00128                 }
00129                 memset(buffer, 0, sizeof(buffer));
00130                 string temp = utils.toRESP2(response.to_string());
00131                 temp = temp.substr(0, temp.size() - 1);
00132 
00133                 // Fix: Use strncpy
00134                 strncpy(buffer, temp.c_str(), sizeof(buffer) - 1);
00135                 buffer[sizeof(buffer) - 1] = '\0';
00136 
00137                 {
00138                     lock_guard<mutex> lock(sendMutex);
00139                     send(clientSocket, buffer, strlen(buffer), 0);
00140                 }
00141             }
00142         }
00143     }
00144 
00145     close(clientSocket);
00146 }
00147 
00148 int main(int argc, char *argv[])
00149 {
00150     cout << "Initializing BlinkDB server..." << endl;
00151     signal(SIGINT, signalHandler);
00152 
00153     if (argc < 2)
00154     {
00155         cout << "Enter 1 for Client-server mode and 0 for redis-benchmark mode" << endl;
00156         cout << "Exiting BlinkDB: Closing server..." << endl;
00157         cout << "Exited" << endl;
00158         discBackupHandler.terminate();
00159         return 0;
00160     }
00161 
00162     serverSocket = socket(AF_INET, SOCK_STREAM, 0);
00163     if (serverSocket == -1)
00164     {
00165         cerr << "Socket creation failed" << endl;
00166         return -1;
00167     }
00168 
00169     sockaddr_in serverAddress;
00170     serverAddress.sin_family = AF_INET;
00171     serverAddress.sin_port = htons(5000);
00172     serverAddress.sin_addr.s_addr = INADDR_ANY;
00173 
00174     int bindStatus = bind(serverSocket, (sockaddr *)&serverAddress, sizeof(serverAddress));
00175     if (bindStatus == -1)
00176     {
00177         cerr << "Binding to port 5000 failed." << endl;
00178         return -2;
00179     }
00180 
00181     int listenStatus = listen(serverSocket, 100000);
00182     if (listenStatus == -1)
00183     {
00184         cerr << "Listening on port 5000 failed." << endl;
00185         return -3;
00186     }
00187 
00188     cout << "Initialized BlinkDB server." << endl;
00189     cout << "Listening on port 5000..." << endl;
00190     cout << "Press Ctrl+C to exit." << endl;
00191 
00192     while (true)
00193     {
00194         sockaddr_in clientAddress;
00195         socklen_t clientAddressSize = sizeof(clientAddress);
00196         int clientSocket = accept(serverSocket, (sockaddr *)&clientAddress, &clientAddressSize);
00197         if (clientSocket == -1)
00198         {
00199             cerr << "Accepting connection failed." << endl;
00200             continue;
00201         }
00202 
00203         thread(handleClient, clientSocket, argv[1]).detach();
00204     }
00205 
00206     close(serverSocket);
00207     return 0;
00208 }
```

4.19 Services/DelService.h File Reference

```
#include <bits/stdc++.h>
#include "../Database/BlinkDB.h"
```

Classes

- class DelService

  Service class for handling key deletion in BlinkDB.

4.20 DelService.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <bits/stdc++.h>
00003 #include "../Database/BlinkDB.h"
00004
00005 using namespace std;
00006
00012 class DelService
00013 {
00014 private:
00018     BlinkDB &blinkDB;
00019
00020 public:
00026     explicit DelService(BlinkDB &blinkDB) : blinkDB(blinkDB) {}
00027
00033     void del(const string &key)
00034     {
00035         try
00036         {
00037             blinkDB.del(key);
00038         }
00039         catch (const exception &e)
00040         {
00041             cerr « "Error deleting key '" « key « "': " « e.what() « endl;
00042         }
00043     }
00044 };
```

4.21 Services/GetService.h File Reference

```
#include <bits/stdc++.h>
#include "../Database/BlinkDB.h"
```

Classes

- class GetService

  Service class for retrieving values from BlinkDB.

4.22 GetService.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <bits/stdc++.h>
00003 #include "../Database/BlinkDB.h"
00004
00005 using namespace std;
00006
00012 class GetService
00013 {
00014 private:
00018     BlinkDB &blinkDB;
00019
00020 public:
00026     explicit GetService(BlinkDB &blinkDB) : blinkDB(blinkDB) {}
00027
00034     string get(const string &key)
00035     {
00036         try
00037         {
00038             string value = blinkDB.get(key);
00039             return value.empty() ? "Key not found" : value;
00040         }
00041         catch (const exception &e)
00042         {
00043             cerr « "Error retrieving key '" « key « "': " « e.what() « endl;
00044             return "Error retrieving value";
00045         }
00046     }
00047 };
```

4.23 Services/SetService.h File Reference

```
#include "../Database/BlinkDB.h"
```

Classes

- class SetService

  Service class for setting key-value pairs in BlinkDB.

4.24 SetService.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "../Database/BlinkDB.h"
00003
00004 using namespace std;
00005
00011 class SetService
00012 {
00013 private:
00017     BlinkDB &blinkDB;
00018
00019 public:
00025     explicit SetService(BlinkDB &blinkDB) : blinkDB(blinkDB) {}
00026
00033     void set(const string &key, const string &value)
00034     {
00035         try
00036         {
00037             blinkDB.set(key, value);
00038         }
00039         catch (const exception &e)
00040         {
00041             cerr « "Error setting key '" « key « "': " « e.what() « endl;
00042         }
00043     }
00044 };
```

4.25 Tests/TestGenerator.cpp File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
```

Functions

- int main ()

4.25.1 Function Documentation

4.25.1.1 main()

```
int main ()
```

Definition at line 8 of file TestGenerator.cpp.

```
00009 {
00010     vector<string> commandType = {"SET", "GET", "DEL", "SET", "GET", "DEL"};
00011     string key[] = {"key1", "key2", "key3", "key4", "key5", "key6", "key7", "key8", "key9", "key10"};
00012     string value[] = {"value1", "value2", "value3", "value4", "value5", "value6", "value7", "value8",
      "value9", "value10"};
00013     string command;
00014     string keyName;
00015     string keyValue;
00016     ofstream output;
00017     cout « "Enter the type of workload you want to generate" « endl;
00018     cout « "Read Heavy: 1" « endl;
00019     cout « "Write Heavy: 2" « endl;
00020     cout « "Balanced: 3" « endl;
00021
00022     int workloadType;
00023     cin » workloadType;
00024
00025     cout « "Enter the number of commands you want to generate" « endl;
00026     int numCommands;
00027     cin » numCommands;
00028
00029     if (workloadType == 1)
00030     {
00031         commandType[0] = "GET";
00032         commandType[1] = "GET";
00033         commandType[2] = "GET";
00034         commandType[3] = "GET";
00035         commandType[4] = "SET";
00036         commandType[5] = "DEL";
00037         output.open("readHeavy_" + to_string(numCommands) + ".txt", ios::out);
00038     }
00039     else if (workloadType == 2)
00040     {
00041         commandType[0] = "SET";
00042         commandType[1] = "SET";
00043         commandType[2] = "SET";
00044         commandType[3] = "SET";
00045         commandType[4] = "GET";
00046         commandType[5] = "DEL";
00047         output.open("writeHeavy_" + to_string(numCommands) + ".txt", ios::out);
00048     }
00049     else
00050     {
00051         output.open("balanced_" + to_string(numCommands) + ".txt", ios::out);
00052     }
00053
00054     for (int i = 0; i < numCommands; i++)
00055     {
00056         command = commandType[rand() % commandType.size()];
00057         keyName = key[rand() % 10];
00058         keyValue = value[rand() % 10];
00059         if (!output.is_open())
```

```
00060         {
00061             cerr « "Unable to open file" « endl;
00062             break;
00063         }
00064         if (command == "SET")
00065         {
00066             output « command « " " « keyName « " " « keyValue « endl;
00067         }
00068         else
00069         {
00070             output « command « " " « keyName « endl;
00071         }
00072     }
00073     return 0;
00074 }
```

4.26 TestGenerator.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <string>
00003 #include <fstream>
00004 #include <vector>
00005
00006 using namespace std;
00007
00008 int main()
00009 {
00010     vector<string> commandType = {"SET", "GET", "DEL", "SET", "GET", "DEL"};
00011     string key[] = {"key1", "key2", "key3", "key4", "key5", "key6", "key7", "key8", "key9", "key10"};
00012     string value[] = {"value1", "value2", "value3", "value4", "value5", "value6", "value7", "value8",
    "value9", "value10"};
00013     string command;
00014     string keyName;
00015     string keyValue;
00016     ofstream output;
00017     cout « "Enter the type of workload you want to generate" « endl;
00018     cout « "Read Heavy: 1" « endl;
00019     cout « "Write Heavy: 2" « endl;
00020     cout « "Balanced: 3" « endl;
00021
00022     int workloadType;
00023     cin » workloadType;
00024
00025     cout « "Enter the number of commands you want to generate" « endl;
00026     int numCommands;
00027     cin » numCommands;
00028
00029     if (workloadType == 1)
00030     {
00031         commandType[0] = "GET";
00032         commandType[1] = "GET";
00033         commandType[2] = "GET";
00034         commandType[3] = "GET";
00035         commandType[4] = "SET";
00036         commandType[5] = "DEL";
00037         output.open("readHeavy_" + to_string(numCommands) + ".txt", ios::out);
00038     }
00039     else if (workloadType == 2)
00040     {
00041         commandType[0] = "SET";
00042         commandType[1] = "SET";
00043         commandType[2] = "SET";
00044         commandType[3] = "SET";
00045         commandType[4] = "GET";
00046         commandType[5] = "DEL";
00047         output.open("writeHeavy_" + to_string(numCommands) + ".txt", ios::out);
00048     }
00049     else
00050     {
00051         output.open("balanced_" + to_string(numCommands) + ".txt", ios::out);
00052     }
00053
00054     for (int i = 0; i < numCommands; i++)
00055     {
00056         command = commandType[rand() % commandType.size()];
00057         keyName = key[rand() % 10];
00058         keyValue = value[rand() % 10];
00059         if (!output.is_open())
00060         {
00061             cerr « "Unable to open file" « endl;
```

```
00062             break;
00063         }
00064         if (command == "SET")
00065         {
00066             output « command « " " « keyName « " " « keyValue « endl;
00067         }
00068         else
00069         {
00070             output « command « " " « keyName « endl;
00071         }
00072     }
00073     return 0;
00074 }
```

4.27 Utils/Utils.h File Reference

```
#include <boost/container_hash/hash.hpp>
#include <bits/stdc++.h>
```

Classes

- class Utils

  Utility class providing helper functions for hashing, string manipulation, and pattern matching.

4.28 Utils.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <boost/container_hash/hash.hpp>
00003 #include <bits/stdc++.h>
00004
00005 using namespace std;
00006
00010 class Utils
00011 {
00012 public:
00019     string hash(const string &key)
00020     {
00021         boost::hash<string> hash_fn;
00022         size_t hash = hash_fn(key);
00023         return to_string(hash);
00024     }
00025
00032     vector<string> splitCommand(string command)
00033     {
00034         vector<string> result;
00035         string word = "";
00036         int count = 2;
00037         for (auto x : command)
00038         {
00039             if (x == '\0')
00040             {
00041                 break;
00042             }
00043
00044             if (x == ' ' && count > 0)
00045             {
00046                 result.push_back(word);
00047                 word = "";
00048                 count--;
00049             }
00050             else
00051             {
00052                 word = word + x;
00053             }
00054         }
00055         result.push_back(word);
00056         return result;
```

```
00057     }
00058
00066     bool startsWith(const string &str, const string &prefix)
00067     {
00068         return str.rfind(prefix, 0) == 0;
00069     }
00070
00077     string toRESP2(const string &command)
00078     {
00079         istringstream stream(command);
00080         vector<string> tokens;
00081         string word;
00082
00083         while (stream » word)
00084         {
00085             tokens.push_back(word);
00086         }
00087
00088         string result = "*" + to_string(tokens.size()) + "\r\n";
00089         for (const auto &token : tokens)
00090         {
00091             result += "$" + to_string(token.size()) + "\r\n" + token + "\r\n";
00092         }
00093
00094         return result;
00095     }
00096
00103     vector<string> fromRESP2(const string &resp)
00104     {
00105         vector<string> result;
00106         istringstream stream(resp);
00107         string line;
00108
00109         getline(stream, line, '\r');
00110         if (line[0] != '*')
00111             return {}; // Must start with '*'
00112
00113         int numArgs = stoi(line.substr(1)); // Number of arguments
00114         stream.ignore(1);                   // Ignore '\n'
00115
00116         for (int i = 0; i < numArgs; i++)
00117         {
00118             getline(stream, line, '\r');
00119             if (line[0] != '$')
00120                 return {}; // Must start with '$'
00121
00122             int len = stoi(line.substr(1)); // Get length of argument
00123             stream.ignore(1);               // Ignore '\n'
00124
00125             string arg(len, ' ');
00126             stream.read(&arg[0], len); // Read the argument
00127             result.push_back(arg);
00128
00129             stream.ignore(2); // Ignore '\r\n'
00130         }
00131
00132         return result;
00133     }
00134 };
```

Index