File: ./README.md

```
# Simplifile-Server
Final comp-sci project backend code. Our hard work by these words guarded please don't steal.
```

File: ./database/__init__.py

```
[binary]
```

File: ./database/interface.py

```
"""
Author: Johnathan Van-Doninck
Date: May 23rd, 2022

A set of functions for communicating with the database.
"""

import sqlite3

DB = sqlite3.connect("./simplifile.db")
CURSOR = DB.cursor()

def add_to_table(table: str, values: str) -> bool:
    CURSOR.execute(f"INSERT INTO (?) VALUES (?);", table, values)
    DB.commit()

def remove_from_table(table: str, field: str, key: str) -> bool:
        CURSOR.execute(f"DELETE FROM (?) WHERE (?)=(?);", table, field, key)
        DB.commit()

def get_from_table(table: str, columns: list, field: str, key: str) -> bool:
    CURSOR.execute(f"SELECT (?)\nFROM (?)\nWHERE (?)=(?);", table, columnts.join(', '), field, key)

def add_to_table(table: str, columns: list, values: list):
    CURSOR.execute("INSERT INTO (?) ((?))\nVALUES ((?));", table, columns.join(', '), values.join(', '))
    DB.commit()
```

File: ./simplifile_api/commands.py

```
"""
Author: Johnathan Van-Doninck
Date: November 28th, 2021

All commands will be stored here, and accessed through this module.
"""

# Imports
from os import system
# from database import interface
from dataclasses import dataclass
# from exceptions import UsernameInUse, MailInUse
# from server.transfer_port import *


@dataclass
class _Command:
    """
    A skeleton class that provides the base for all other command classes.
    """
    id: int
    command: str = None
    args: list = None

    def validate(self, ext_command: tuple) -> bool:
        """
        Returns true if the id and the command are the same as the ones fed through ext_command.
```

```
        """
        return (self.id, self.command) == ext_command

    def execute(self):
        """
        Passes the command to the server for handling. Is empty here.
        """
        pass

    def validate_arguments(self, num_of_args: int) -> bool:
        """
        Validates the arguents, ensuring they are valid for the specific function. Is empty here.
        """
        return len(self.args) == num_of_args
# ---

@dataclass
class Upload(_Command):
    def validate(self) -> bool:
        """
        Returns true if the id and the command match the valid ones for the upload command.
        """
        return super().validate((0, "upload"))

    def execute(self):
        """
        Passes the command to the server for handling. Currently WIP.

        args: (0) location on server (1) size of file (2) user to which the file belongs
        """


    def validate_arguments(self):
        """
        Validates the arguents, ensuring they are valid for the specific function. Is empty here.
        """
        return super().validate_arguments(3)
# ---

@dataclass
class Download(_Command):
    def validate(self) -> bool:
        """
        Returns true if the id and the command match the valid ones for the Download command.
        """
        return super().validate((1, "download"))

    def execute(self, data):
        """
        Passes the command to the server for handling. Currently WIP.
        """
        return super().execute()

    def validate_arguments(self):
        """
        Validates the arguents, ensuring they are valid for the specific function. Is empty here.
        """
        return super().validate_arguments(3)
# ---

@dataclass
class CreateUser(_Command):
    def validate(self) -> bool:
        """
        Returns true if the id and the command match the valid ones for the CreateUser command.
        """
        return super().validate((2, "ucreate"))

    def execute(self):
        """
        Passes the command to the server for handling. Currently WIP
        All neccessary details (email: str, username: str, password: str)
        are passed through the self.args field, in a yet to be determined order.
        """
        user = interface.findUsers(self.args[0], 'username')
        mail = interface.findUsers(self.args[0], 'email')
```

```
104         print(user)
105         print(mail)
106         userExist = user == self.args[1]
107         mailExist = mail == self.args[0]
108         print(mailExist)
109         print(userExist)
110         if not userExist:
111             return UsernameInUse
112         elif not mailExist:
113             return MailInUse
114         else:
115             interface.addUser(self.args[0], self.args[1], self.args[2])
116             print("a")
117             system(f"mkdir ./{self.args[0]}")   # INCREDIBLY INSECURE. Implement input checks.
118             return Success
119 
120     def validate_arguments(self):
121         """
122         Validates the arguents, ensuring they are valid for the specific function. Is empty here.
123         """
124         return super().validate_arguments(3)
125 # ---
126 
127 @dataclass
128 class DeleteUser(_Command):
129     def validate(self) -> bool:
130         """
131         Returns true if the id and the command match the valid ones for the DeleteUser command.
132         """
133         return super().validate((3, "udelete"))
134 
135     def execute(self):
136         """
137         Passes the command to the server for handling. Currently WIP.
138         All neccessary details (email:str, username: str, password: str)
139         are passed through the self.args field
140         """
141         password = interface.findUsers(self.args[0], 'password')
142         if password == self.args[2]:
143             interface.delUser(self.args[0], self.args[1])
144             system(f"rm -r ./{self.args[1]}")   # INCREDIBLY INSECURE. Implement input checks.
145             return Success
146         else:
147             return Abort
148 
149     def validate_arguments(self):
150         """
151         Validates the arguents, ensuring they are valid for the specific function.
152         """
153         return super().validate_arguments(3)
154 # ---
155 
156 @dataclass
157 class ChangePassword(_Command):
158     def validate(self) -> bool:
159         """
160         All necessary details (email: str, password: str, new_pass: str)
161         """
162         return super().validate((4, "uchange"))
163 
164     def execute(self):
165         """
166         Passes the command to the server for handling. Currently WIP
167         """
168         if interface.findUsers(f'{self.args[0]}', 'password') == self.args[2]:
169             interface.changeUsers(f'{self.args[0]}', 'password', value)
170             return Success
171         return Abort
172 
173     def validate_arguments(self):
174         """
175         Validates the arguents, ensuring they are valid for the specific function.
176         """
177         return super().validate_arguments(3)
178 # ---
179 
```

```
@dataclass
class Abort(_Command):
    """
    This class is unique - it tells the server to abort whatever operation it was meant to do.
    Only used in cases where the received command is invalid in one way or another (Invalid user ID,
    Invalid syntax, etc.)
    """
    def validate(self) -> bool:
        """
        Returns true if the id and the command match the valid ones for the Abort command.
        """
        return super().validate((-1, "abort"))

    def execute(self):
        """
        Passes the command to the server for handling. Currently WIP
        """
        return super().execute()

    def validate_arguments(self):
        """
        Validates the arguents, ensuring they are valid for the specific function. Is empty here.
        """
        return super().validate_arguments()
# ---

@dataclass
class Success(_Command):
    """
    Tells the client programme the operation was performed successfully, so that the user can be moved to the next screen.
    """
    def validate(self) -> bool:
        return super().validate((999, "success"))

    def execute(self):
        """
        Passes the command to the server for handling. Currently WIP
        """
        return super().execute()

    def validate_arguments(self):
        """
        Validates the arguents, ensuring they are valid for the specific function. Is empty here.
        """
        return super().validate_arguments()
```

File: ./simplifile_api/parser.py

```
"""
Author: Johnathan Walter Van-Doninck
Date: April 25th, 2022

A service class that provides utilities to parse commands received by the server.
"""

from simplifile_api import commands, exceptions


class_map = {
    "upload":   commands.Upload,        # 0
    "download": commands.Download,      # 1
    "addusr":   commands.CreateUser,    # 2
    "delusr":   commands.DeleteUser,    # 3
    "chgpass":  commands.ChangePassword # 4
}


def _parser(cmd_list: list()):          # Used to parse into an object and validate it.
    try:
        command = class_map[ cmd_list[ 0 ] ](int(cmd_list[ 1 ]), cmd_list[ 0 ], cmd_list[2:])
        if command.validate() and command.validate_arguments():
            return command
        else:
            return exceptions.InvalidRequestError()

    except KeyError:
        return exceptions.InvalidRequestError()
    except TypeError:
        return exceptions.InvalidRequestError()


def parser(command: str):               # Used to parse from raw text into manageable bits.
    if command[-1] != '\x04':
        return exceptions.InvalidRequestError()
    command_list = [ string.split(": ")[1] for string in command[:-1].split("\n") ]
    return _parser(command_list)
```

File: ./simplifile_api/__init__.py

```
[binary]
```

File: ./simplifile_api/exceptions.py

```
"""
Author: Johnathan Van-Doninck
Date: April 25th, 2022

A file containing all self made exceptions for parsing, permission,
and validation errors.
"""

class InvalidRequestError(Exception):
    def __init__(self, message = "Error: Invalid request. See documentation."):
        self.message = message
        super().__init__(message)
```

File: ./simplifile_api/__pycache__/parser.cpython-310.pyc

[binary]

File: ./simplifile_api/__pycache__/__init__.cpython-310.pyc

[binary]

File: ./simplifile_api/__pycache__/commands.cpython-310.pyc

[binary]

File: ./simplifile_api/__pycache__/exceptions.cpython-310.pyc

[binary]

File: ./server/file_recieve_port.py

```python
"""
Author: Johnathan Van-Doninck
Date: May 2nd, 2022

Receives files from client
"""

from socketserver import BaseRequestHandler
from simplifile_api import commands, exceptions

class FileRecieveHandler(BaseRequestHandler):
    def __init__(self, request, client_address, server, file_size, file_name, client_name):
        self.file_size = file_size
        self.file_name = file_name

    @classmethod
    def creator(cls, *args, **kwargs):
        """
        Used to create a handler class with the required parameters to facilitate file transfer.
        """
        def _handler_creator(request, client_address, server):
            cls(request, client_address, server, *args, **kwargs)
        return _handler_creator

    def handle(self):
        timer = time.time()
        try:
            with open(self.file_name, 'bw') as file:
                data = b''
                while True:
                    data = self.request.recv(self.file_size)
                    print(data)
                    if not data: break
                    file.write(data)
                self.request.send(bytes(f"{commands.Success()}", 'ascii'))
                print(time.time() - timer)
                print("Done.")
                print("# ")
        except Exception as e:
            print(e)
            system(f"rm -r {self.file_path}")
            print("# ")
```

File: ./server/server.py

```
"""
Author: Johnathan Van-Doninck
Date: May 12th, 2022

Full server, implementing the request port and the upload/download port.
"""

from socketserver import ThreadingTCPServer
from threading import Thread
from request_port import RequestHandler
from os import system
from sys import exit

VERSION = "Simplifile Server Management Shell, Version b0.0.1"

def help():
    print("help - print this help message")
    print("clear - clear the screen")
    print("version - print the programme version")
    print("exit - shutdown the server and exit")


def clear():
    system("clear")

def version():
    print(VERSION)

def main():
    with ThreadingTCPServer((HOST, PORT), RequestHandler) as server:
        server_thread = Thread(target=server.serve_forever)
        print("Server thread created...")
        server_thread.daemon = True
        print("Thread daemonized...")
        server_thread.start()
        print(f"Server started on thread {server_thread.name}")
        print("Enter 'help' for more information.")
        while True:
            man_com = input("# ")
            if man_com == "help": help()
            elif man_com == "version": version()
            elif man_com == "clear": clear()
            elif man_com == "exit": exit()
            else: print("Unknown command.")
```

File: ./server/__init__.py

```
[binary]
```

File: ./server/file_send_port.py

```
"""
Author: Johnathan Van-Doninck
Date: May 2nd, 2022

Send files to client
"""

from socketserver import BaseRequestHandler

class FileSendHandler(BaseRequestHandler):
    def __init__(self, request, client_address, server, file_name, file_size, client_name):
        self.file_name = file_name
        self.file_size = file_size
        self.client_name = client_name

    @classmethod
    def creator(cls, *args, **kwargs):
        def _handler_creator(request, client_address, server):
            cls(request, client_address, server, *args, **kwargs)
        return _handler_creator

    def handle(self):
        with open(self.file_name, 'br') as file:
            data = b''
            while data != '':
                data = file.read(self.file_size)
                self.request.send(data)
                print(data)
            self.request.sendall(bytes(f"{commands.Success()}", 'ascii'))
            print("Done")
            print("# ")

```

File: ./server/request_port.py

```
"""
Author: Johnathan Van-Doninck
Date: April 26th, 2022

The port that receives all requests from clients and handles them accordingly.
"""

from sys import path
path.append("/home/luciferin/Documents/Fuck my life/Simplifile-Server")
from socketserver import BaseRequestHandler, TCPServer
from simplifile_api import exceptions, commands, parser

BUFFER = 1024

class RequestHandler(BaseRequestHandler):
    def handle(self):
        command_raw = ""
        print(f"Receiving command from {self.client_address}")
        while len(command_raw) < BUFFER:
            data = self.request.recv(1).decode('ascii')
            print(data, end='')
            command_raw += data
            if data == '\x04':
                break
        print(command_raw)
        command = parser.parser(command_raw)
        print(command)
        if command is commands._Command:
            command.execute()
            self.request.send("Operation Successful.")
            print("# ")
        else:
            self.request.send(bytes(f'{command}', 'ascii'))
            print("# ")


if __name__ == '__main__':
    # For testing purposes only, remove once server is fully implemented.
    host = "127.0.0.1"
    port = 55445
    with TCPServer((host, port), RequestHandler) as server:
        print("Server Started...")
        server.serve_forever()
```

File: ./make.sh

```bash
#!/bin/bash

# Author - Johnathan Van-Doninck
# Date - May 7th, 2022
# Make script for setting up project dependencies, such as
# the API, the database, the root of the server, etc.
#
# Positional arguments:
# $1 - server root, defaults to /home/{user}/simplifile/
# $2 - database file name, defaults to /home/{user}/simplifile/Simplifile

# Function definitions
verify_exists() {       # Verifies a given program exists on the computer.
    whereis_out=$(whereis $1)
    if [[ "$1:" = "$whereis_out" ]]
    then
        return 0
    else
        return 1
    fi
}
verify_requirements() {
    verify_exists $1
    if [[ $? -eq 0 ]]
    then
        echo "error: dependency $1 could not be located. Please resolve issue and try again."
        reqs_met=0
        let GLOBAL error_code+=1
    fi
}
test_fun() {
    echo $@
}
# ---
# Positional argument parsing
error_code=0
if [[ $1 -eq 0 ]]       # Checks if positional argument 1 is used, defaults if not.
then
    user = whoami
    root_path="/home/$user/simplifile/"
else
    root_path=$1
fi
if [[ $2 -eq 0 ]]       # Checks if positional argument 2 is used, defaults if not.
then
    db_path="~/simplifile/Simplifile"
else
    db_path=$1
fi
# ---
# Requirement verification
reqs_met=1
verify_requirements "python3"
verify_requirements "sqlite3"
# ---
# Environment setup
if [[ reqs_met -eq 1 ]]
then
    echo "Working..."
    mkdir root_path
    mv ./ $root_path
    cd $root_path
    echo "Created server root..."
    echo "export PATH=$PATH:\"$root_path/simplifile_api\"" > /home/$user/.bashrc
    echo "Added API to PATH variable..."
    # TODO: Create database
    exec
    # TODO: Add server files to server root
    source "~/.bashrc"

else
    echo "Exiting with error code $error_code"
fi
```