

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ



Phạm Tùng Lâm

XÂY DỰNG ỨNG DỤNG GAME
ĐA NGƯỜI DÙNG SỬ DỤNG SOCKET
BẰNG NGÔN NGỮ PYTHON

KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY
Ngành: Công nghệ kỹ thuật điện tử viễn thông – Chất lượng cao

HÀ NỘI - 2022

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Phạm Tùng Lâm

**XÂY DỰNG ỨNG DỤNG GAME
ĐA NGƯỜI DÙNG SỬ DỤNG SOCKET
BẰNG NGÔN NGỮ PYTHON**

KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY
Ngành: Công nghệ kỹ thuật điện tử viễn thông - Chất lượng cao

Cán bộ hướng dẫn: TS. Bùi Trung Ninh

HÀ NỘI - 2022

LỜI CẢM ƠN

Quá trình thực hiện khóa luận tốt nghiệp là giai đoạn quan trọng nhất trong quãng đời mỗi sinh viên. Khóa luận tốt nghiệp là tiền đề nhằm trang bị cho chúng em những kỹ năng nghiên cứu, những kiến thức quý báu trước khi lập nghiệp.

Trước hết, chúng em xin chân thành cảm ơn quý thầy, cô Khoa Điện tử - Viễn thông. Đặc biệt là các thầy, cô trong bộ môn Hệ thống viễn thông đã tận tình chỉ dạy và trang bị cho em những kiến thức cần thiết trong suốt thời gian ngồi trên ghế giảng đường, làm nền tảng cho em có thể hoàn thành được bài khóa luận này.

Em xin trân trọng cảm ơn thầy Bùi Trung Ninh đã tận tình giúp đỡ, định hướng cách tư duy và cách làm việc khoa học. Đó là những góp ý hết sức quý báu không chỉ trong quá trình thực hiện khóa luận này mà còn là hành trang tiếp bước cho em trong quá trình học tập và lập nghiệp sau này.

Khóa luận của em còn những hạn chế về năng lực và những thiếu sót trong quá trình nghiên cứu. Em xin lắng nghe và tiếp thu những ý kiến của giáo viên phản biện để hoàn thiện, bổ sung kiến thức.

Em xin chân thành cảm ơn!

TÓM TẮT

Tóm tắt: Ngày nay, Game Online hiện nay ngày càng trở nên phổ biến và có nhiều người chơi. Để có thể lập trình được một Game Online, không những lập trình viên phải có những kiến thức về lập trình thuật toán và đồ họa... thì còn phải có kiến thức về mạng máy tính. Vì thế có thể **coi làm** game online là một cách tốt để lập trình viên có thể tìm hiểu và thực hành những kiến thức về mạng máy tính. Cũng vì lí do này nên em đã quyết **định** chọn đề tài “**Xây dựng ứng dụng game đa người dùng sử dụng socket bằng ngôn ngữ python**”. Nội dung của khóa luận sẽ tập trung vào lập trình socket, sự giao tiếp client – server cũng như thuật toán của game mà sẽ không chú trọng vào phần đồ họa, hình ảnh của game.

Từ Khóa: *Lập trình mạng socket, ngôn ngữ Python, game đa người dùng.*

LỜI CAM ĐOAN

Em xin cam đoan kết quả đạt được trong khóa luận là sản phẩm của riêng cá nhân, không sao chép lại của người khác. Trong toàn bộ nội dung của khóa luận, những điều được trình bày hoặc là của cá nhân hoặc là được tổng hợp từ nhiều nguồn tài liệu. Tất cả các tài liệu tham khảo đều có xuất xứ rõ ràng và được trích dẫn hợp pháp. Em xin hoàn toàn chịu trách nhiệm và chịu mọi hình thức kỷ luật theo quy định cho lời cam đoan của mình.

Hà Nội, ngày tháng năm 2022

Sinh viên

MỤC LỤC

MỞ ĐẦU.....	1
CHƯƠNG 1: TỔNG QUAN VỀ MẠNG MÁY TÍNH VÀ NGÔN NGỮ PYTHON.....	2
1.1. Định nghĩa mạng máy tính.....	2
1.2. Phân loại mạng máy tính.....	4
1.3. Mô hình phân tầng.....	10
1.3.1. Mô hình OSI.....	10
1.3.2. Mô hình TCP/IP.....	13
1.4. Các giao thức mạng.....	15
1.4.1. Giao thức TCP.....	15
1.4.2. Giao thức UDP.....	16
1.5. Mô hình ứng dụng mạng.....	17
1.5.1. Mô hình Khách – Máy chủ (Client – Server).....	17
1.5.2. Mô hình ngang hàng (Peer to peer).....	18
1.5.3. Mô hình lai (Hybrid).....	18
1.6. Tổng quan về ngôn ngữ python.....	19
1.6.1. Giới thiệu ngôn ngữ Python.....	19
1.6.2. Các giai đoạn phát triển của Python.....	19
1.6.3. Các tính năng nổi bật của ngôn ngữ python.....	19
1.6.4. Các thành phần và cú pháp cơ bản trong chương trình Python.....	20
CHƯƠNG 2: LẬP TRÌNH SOCKET.....	24
2.1. Tìm hiểu về Socket.....	24
2.2. Phân loại Socket.....	25
2.2.1. Stream Socket.....	25
2.2.2. Datagram socket.....	25
2.2.3. Unix socket.....	25
2.2.4. Web socket.....	26
2.2. Socket trong python.....	26
2.2.1. Một số khái niệm riêng của Socket.....	26
2.2.2. Mô-đun Socket trong Python.....	27
2.2.3. Các hàm socket sử dụng trong Python.....	28
2.3. Demo chương trình lập trình socket bằng python.....	28
2.3.1. Tạo một Máy chủ (Server) trong Python.....	29
2.3.2. Tạo một máy khách (client) trong Python.....	30

2.3.3.	Thực thi Demo chương trình Client – Server trong Python.....	30
2.4.	Khái niệm hệ quản trị cơ sở dữ liệu SQLite.....	31
2.5.	SQLite trong Python.....	32
2.5.1.	Kết nối tới cơ sở dữ liệu.....	33
2.5.2.	SQLite Cursor.....	33
2.5.3.	Tạo cơ sở dữ liệu.....	33
2.5.4.	Tạo bảng trong cơ sở dữ liệu.....	34
CHƯƠNG 3: XÂY DỰNG CHƯƠNG TRÌNH ỨNG DỤNG.....		36
3.1.	Giới thiệu về trò chơi cờ caro.....	36
3.2.	Phân tích và thiết kế.....	37
3.2.1.	Mục tiêu của chương trình.....	37
3.2.2.	Yêu cầu của chương trình.....	37
3.2.3.	Thiết kế chương trình.....	38
3.3.	Cài đặt và kết quả đạt được.....	40
3.4.	Nhận xét.....	51
KẾT LUẬN.....		52
PHỤ LỤC.....		53

DANH MỤC CÁC HÌNH ẢNH

<i>Hình 1.1: Mô hình thể hiện một số thành phần của mạng máy tính.....</i>	<i>3</i>
<i>Hình 1.2: Mô hình mạng LAN</i>	<i>5</i>
<i>Hình 1.3: Mô hình mạng WAN</i>	<i>6</i>
<i>Hình 1.4: Mô hình mạng MAN</i>	<i>6</i>
<i>Hình 1.5: Mô hình mạng hình sao.....</i>	<i>8</i>
<i>Hình 1.6: Mô hình mạng tuyến tính</i>	<i>8</i>
<i>Hình 1.7: Mô hình mạng vòng.....</i>	<i>9</i>
<i>Hình 1.8: Kiến trúc phân tầng tổng quát.....</i>	<i>10</i>
<i>Hình 1.9: Mô hình OSI.....</i>	<i>11</i>
<i>Hình 1.10. Mô hình TCP/IP</i>	<i>13</i>
<i>Hình 1.11: Mô hình OSI và TCP/IP</i>	<i>15</i>
<i>Hình 1.12. Mô hình Khách – Máy chủ (Client – Server).....</i>	<i>18</i>
<i>Hình 1.13. Ví dụ về câu lệnh trong Python</i>	<i>21</i>
<i>Hình 1.14. Ví dụ về câu lệnh nhiều dòng trong Python.....</i>	<i>22</i>
<i>Hình 1.15. Ví dụ về thụt đầu dòng trong python</i>	<i>22</i>
<i>Hình 1.16. Ví dụ về comment trong Python</i>	<i>23</i>
<i>Hình 1.17. Ví dụ về khối lệnh trong Python.....</i>	<i>23</i>
<i>Hình 2.1. Cách hoạt động của Socket</i>	<i>24</i>
<i>Hình 2.2. Demo chương trình Máy chủ (Server)</i>	<i>29</i>
<i>Hình 2.3. Demo chương trình Máy khách (Client).....</i>	<i>30</i>
<i>Hình 2.4. Kết quả Demo chương trình Client – Server trong Python</i>	<i>31</i>
<i>Hình 2.5. Phương thức connect() trong SQLite.....</i>	<i>33</i>
<i>Hình 2.6. SQLite Cursor</i>	<i>33</i>
<i>Hình 2.7. Ví dụ về tạo cơ sở dữ liệu</i>	<i>34</i>
<i>Hình 2.8. Ví dụ tạo bảng trong cơ sở dữ liệu</i>	<i>35</i>
<i>Hình 2.9. Kết quả tạo bảng cơ sở dữ liệu.</i>	<i>35</i>

Hình 3.1. Ví dụ người chơi O thắng	36
Hình 3.2. Ví dụ người chơi X thắng.....	36
Hình 3.3. Ví dụ cờ hòa	37
Hình 3.4. Kết quả cài đặt máy chủ	40
Hình 3.5. Giao diện đăng nhập/ đăng ký	41
Hình 3.6. Giao diện đăng nhập, đăng ký khi không điền tài khoản, mật khẩu.....	42
Hình 3.7. Giao diện màn hình đăng nhập, đăng ký khi nhập mật khẩu ít hơn 6 ký tự	43
Hình 3.8. Giao diện màn hình đăng nhập, đăng ký khi điền sai tài khoản, mật khẩu ..	44
Hình 3.9. Server khi đăng nhập sai tên tài khoản, mật khẩu.....	44
Hình 3.10. Giao diện màn hình khi đăng nhập thành công.....	45
Hình 3.11. Giao diện bảng xếp hạng người chơi	46
Hình 3.12. Giao diện khi tìm kiếm thứ hạng người chơi	47
Hình 3.13. Giao diện đợi trò chơi.....	48
Hình 3.14. Giao diện khi vào trò chơi	48
Hình 3.16. Giao diện khi người chơi 2 chơi	49
Hình 3.17. Giao diện khi người chơi 2 thắng	50
Hình 3.17. Giao diện bảng xếp hạng người chơi sau khi trò chơi kết thúc.....	50

DANH MỤC CÁC BẢNG BIỂU

Bảng 1.1. Các ứng dụng phổ biến và giao thức giao vận tương ứng	17
Bảng 1.2. Bảng các keyword trong Python 3.10.3	20
Bảng 2.1 Giá trị tham số socket_family trong hàm socket	27
Bảng 2.2. Một số lệnh sơ bản của SQLite	31

MỞ ĐẦU

Trong kỷ nguyên hiện đại kỹ thuật số, việc giao tiếp và trao đổi thông tin là vô cùng quan trọng nên chúng ta không thể phủ nhận được vai trò to lớn **mà** mạng máy tính đem đến cho con người, đây được xem như một trong những giải pháp công nghệ công tin quan trọng nhất hiện nay.

Mạng máy tính có vai trò như người cung cấp thông tin cho mọi đối tượng sử dụng. Hiện nay, mạng máy tính có một vai trò vô cùng quan trọng vì nó cung cấp cho chúng ta những phần mềm trò chuyện trực tuyến như thư điện tử (Gmail) hay ngay trên chính nền tảng xã hội (Facebook, Zalo, Twitter,...)

Không những thế, mạng máy tính còn đóng vai trò là đầu gạch giữa kết nối dữ liệu giữa các máy tính với nhau. Người dùng có thể sử dụng một kho thông tin khổng lồ đa dạng và phong phú để đem đến những thuận tiện trong quá trình học tập, làm việc.

Để có thể tìm hiểu và giải đáp được các kiến thức về mạng máy tính, hay là làm cách nào để Server có thể nói chuyện được với các Client nên em đã quyết định chọn đề tài: **“Xây dựng ứng dụng game đa người dùng sử dụng socket bằng ngôn ngữ python”** cho khóa luận của mình.

Nội dung khóa luận bao gồm các vấn đề: giới thiệu lý thuyết tổng quan, các giao thức UDP, TCP, làm rõ về Socket, giới thiệu ngôn ngữ lập **tình** Python để tạo ra một ứng dụng game đa người **dùng** đơn giản là game Cờ caro. Ứng dụng sẽ cho phép người **dùng** chơi game với các người chơi khác. Ngoài ra em còn tích hợp thêm cơ sở dữ liệu để người dùng có thể đăng nhập và biết được thứ hạng của mình so với những người chơi khác.

CHƯƠNG 1: TỔNG QUAN VỀ MẠNG MÁY TÍNH VÀ NGÔN NGỮ PYTHON

Thuật ngữ "Internet" xuất hiện lần đầu vào khoảng năm 1974. Lúc đó mạng vẫn được gọi là ARPANET. Năm 1983, giao thức TCP/IP chính thức được coi như một chuẩn đối với ngành quân sự Mỹ và tất cả các máy tính nối với ARPANET phải sử dụng chuẩn mới này. Năm 1984, ARPANET được chia ra thành hai phần: phần thứ nhất vẫn được gọi là ARPANET, dành cho việc nghiên cứu và phát triển; phần thứ hai được gọi là MILNET, là mạng dùng cho các mục đích quân sự.

Giao thức TCP/IP ngày càng thể hiện rõ các điểm mạnh của nó, quan trọng nhất là khả năng liên kết các mạng khác với nhau một cách dễ dàng. Chính điều này cùng với các chính sách mở cửa đã cho phép các mạng dùng cho nghiên cứu và thương mại kết nối được với ARPANET, thúc đẩy việc tạo ra một siêu mạng (SuperNetwork). Năm 1980, ARPANET được đánh giá là mạng trụ cột của Internet.

Mốc lịch sử quan trọng của Internet được xác lập vào giữa thập niên 1980 khi tổ chức khoa học quốc gia Mỹ NSF thành lập mạng liên kết các trung tâm máy tính lớn với nhau gọi là NSFNET. Nhiều doanh nghiệp đã chuyển từ ARPANET sang NSFNET và do đó sau gần 20 năm hoạt động, ARPANET không còn hiệu quả đã ngừng hoạt động vào khoảng năm 1990.

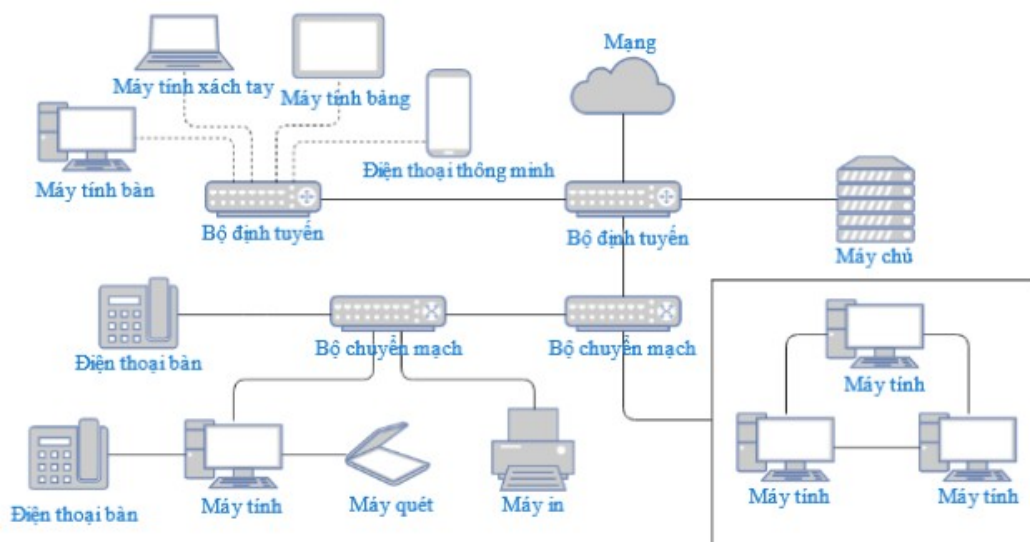
Với khả năng kết nối mở như vậy, Internet đã trở thành một mạng lớn nhất trên thế giới, mạng của các mạng, xuất hiện trong mọi lĩnh vực thương mại, chính trị, quân sự, nghiên cứu, giáo dục, văn hoá, xã hội... Cũng từ đó, các dịch vụ trên Internet không ngừng phát triển tạo ra cho nhân loại một thời kỳ mới: kỷ nguyên thương mại điện tử trên Internet.

1.1. Định nghĩa mạng máy tính

Mạng máy tính là thuật ngữ chung để chỉ một mạng gồm các thiết bị máy tính được kết nối (chẳng hạn như máy tính xách tay, máy tính để bàn, máy chủ, điện thoại thông minh và máy tính bảng) và một loạt các thiết bị IoT ngày càng mở rộng (chẳng hạn như máy ảnh, khóa cửa, chuông cửa, tủ lạnh, hệ thống âm thanh/hình ảnh, bộ điều nhiệt và các cảm biến khác nhau). Khái niệm mạng liên quan đến nhiều vấn đề, bao gồm:

- Giao thức truyền thông (protocol): Mô tả những nguyên tắc mà tất cả các thành phần mạng cần tuân thủ để có thể trao đổi với nhau
- Topo (mô hình ghép nối mạng/ hình trạng mạng): Mô tả cách thức nối các thiết bị với nhau.
- Địa chỉ: Mô tả cách thức định vị một đối tượng trên mạng.
- Định tuyến (routing): Mô tả cách thức dữ liệu truyền từ thiết bị này sang thiết bị khác trên mạng.
- Tính tin cậy (reliability): Giải quyết tính toàn vẹn của dữ liệu. đảm bảo dữ liệu nhận được chính xác như dữ liệu gửi đi.
- Khả năng liên tác (interoperability): Chỉ mức độ các sản phẩm phần mềm và phần cứng của các hãng sản xuất khác nhau có thể làm việc cùng nhau.
- An ninh (security): Đảm bảo an toàn, hoặc bảo vệ tất cả các thành phần của mạng
- Chuẩn (standard): Thiết lập các quy tắc và luật lệ cụ thể cần phải tuân theo.

Mạng viễn thông cũng là mạng máy tính. Các node chuyển mạch là hệ thống máy tính được kết nối với nhau bằng các đường truyền dẫn và hoạt động truyền dẫn tuân theo các chuẩn mô hình tham chiếu OSI.



Hình 1.1: Mô hình thể hiện một số thành phần của mạng máy tính

Một hệ thống mạng máy tính hoàn chỉnh sẽ bao gồm những phần sau: Các thiết bị đầu cuối, môi trường truyền dẫn, thiết bị kết nối vật lý và phần mềm kết nối.

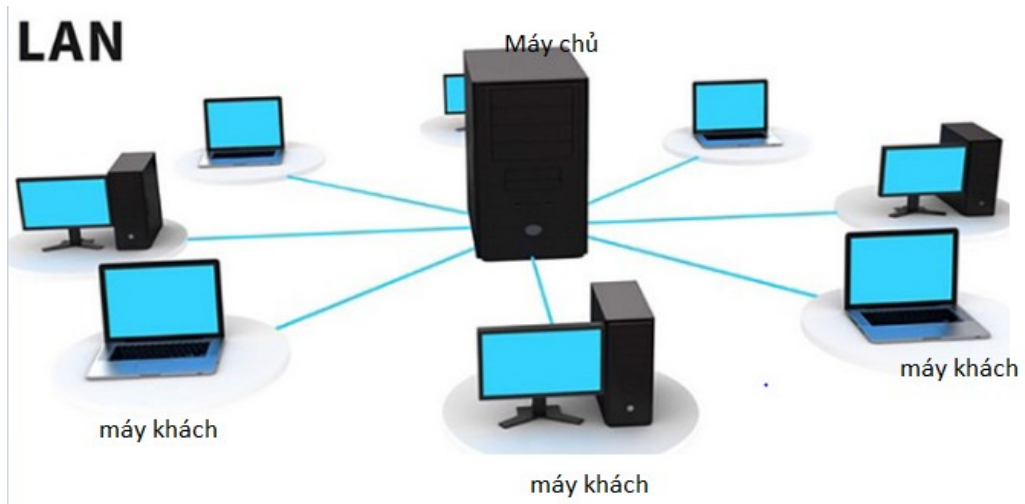
- Các thiết bị đầu cuối: Máy tính, điện thoại, máy in, máy quét, máy ảnh... đều là các thiết bị đầu cuối trong hệ thống mạng máy tính. Những thiết bị này sẽ được kết nối với nhau qua các thiết bị kết nối hoặc môi trường truyền dẫn.
- Môi trường truyền dẫn: Đây là những thiết bị kết nối không dây ví dụ: bộ phát sóng, bộ truyền tín hiệu, sóng điện từ... được dùng để trao đổi dữ liệu.
- Thiết bị kết nối vật lý là những thiết bị như dây nối, modun, switch... được kết nối trực tiếp từ thiết bị đầu cuối này sang thiết bị đầu cuối khác.
- Phần mềm kết nối: Tương tự như môi trường truyền dẫn thì phần mềm kết nối là những chương trình, ứng dụng được cài đặt trên các thiết bị đầu cuối và có chức năng chia sẻ dữ liệu qua các đường truyền không dây.

1.2. Phân loại mạng máy tính

Có rất nhiều kiểu mạng máy tính khác nhau. Việc phân loại chúng dựa trên các đặc điểm chung. Ví dụ. mạng máy tính thường được phân loại vùng địa lý (diện tích hoạt động) (ví dụ: mạng cục bộ, mạng diện rộng ...); theo topo (mô hình ghép nối mạng) (ví dụ: point to point hay broadcast), hoặc **hteo** kiểu đường truyền thông **mà** mạng sử dụng.

a. Phân loại mạng theo khoảng cách địa lý

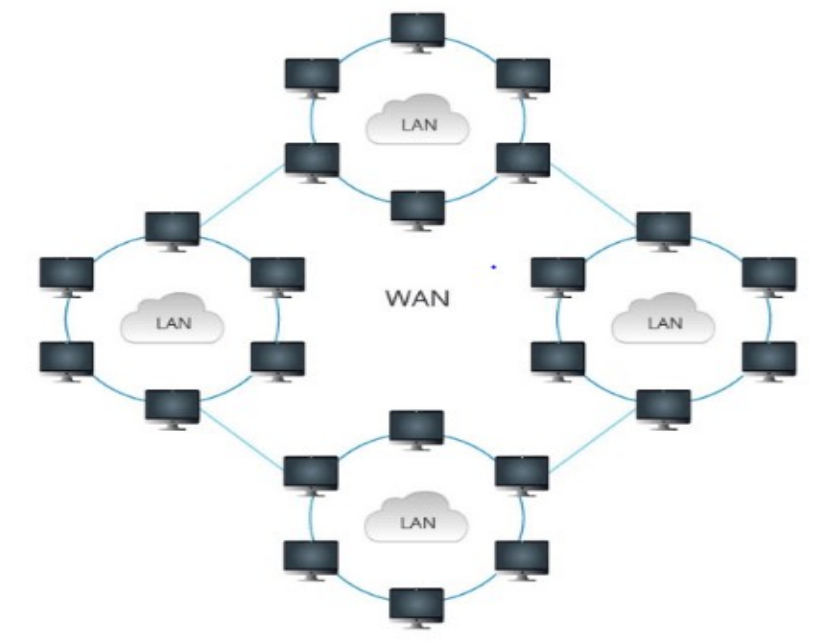
Mạng LAN (Local Area Network – còn gọi là mạng cục bộ) là một nhóm các máy tính và thiết bị truyền thông mạng được kết nối với nhau trong một khu vực nhỏ như tòa nhà cao ốc, trường đại học, khu giải trí... Tốc độ truyền dữ liệu cao, từ 10÷100 Mbps đến hàng trăm Gbps, thời gian trễ nhỏ (cỡ 10 μ s), độ tin cậy cao, tỷ số lỗi bit từ 10⁻⁸ đến 10⁻¹¹. Mạng LAN trong thực tế được kết nối thành mạng ngang hàng hoặc dựa trên máy chủ. Nhưng các máy tính nếu muốn kết nối mạng LAN đều phải có kết nối dựa trên các yêu cầu: phải có **card** giao tiếp mạng (NIC: Network Interface Card) và thiết bị truyền thông (có dây hoặc không dây).



Hình 1.2: Mô hình mạng LAN

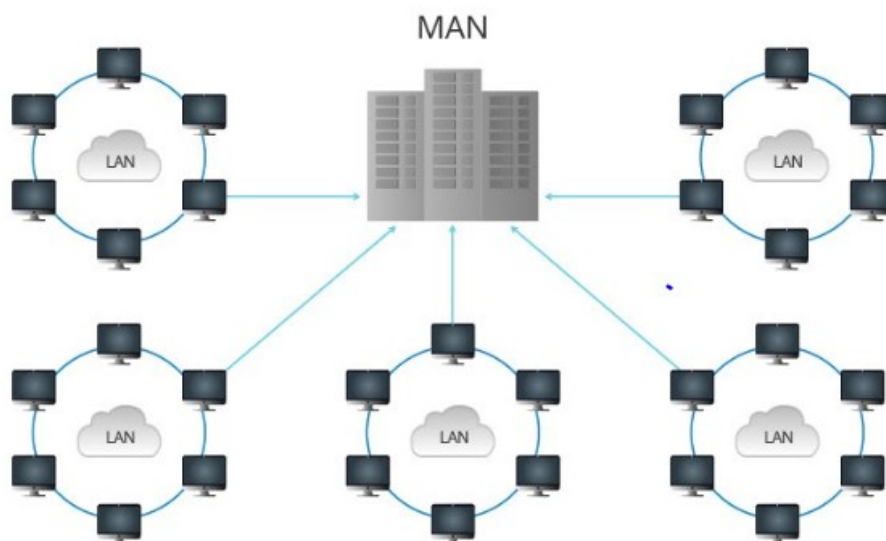
Mạng WAN (Wide Area Network) hay còn gọi là mạng diện rộng với khả năng kết nối các máy tính ở cách nhau những khoảng cách lớn. Mạng diện rộng sẽ bao gồm hai hay nhiều LAN. Mạng WAN có khả năng bao phủ một vùng diện tích rộng (có thể là một thành phố, một vùng lãnh thổ, một quốc gia...). Trên mạng diện rộng này, các LAN được kết nối bằng cách sử dụng các đường dây của nhà cung cấp dịch vụ truyền tải công cộng. . Những đặc trưng cơ bản của một mạng WAN:

- ☐ Hoạt động trên phạm vi quốc gia hoặc toàn cầu
- ☐ Tốc độ truyền dữ liệu thấp so với mạng LAN
- ☐ Lỗi truyền cao



Hình 1.3: Mô hình mạng WAN

MAN (Metropolitan Area Network) hay còn gọi là “mạng đô thị”, là một mạng máy tính kết nối các máy tính trong một khu vực đô thị, có thể là một thành phố lớn, nhiều thành phố và thị trấn hoặc bất kỳ khu vực rộng lớn nhất định có nhiều tòa nhà. Mạng MAN lớn hơn mạng cục bộ LAN, nhưng lại nhỏ hơn WAN.



Hình 1.4: Mô hình mạng MAN

Mạng cá nhân PAN(Person Area Network): là mạng có diện tích nhỏ gồm hai hay nhiều mạng máy tính nối với nhau bằng các thiết bị định tuyến (router) cho phép dữ liệu được gửi qua lại giữa chúng. Các thiết bị định tuyến có nhiệm vụ hướng dẫn giao thông dữ liệu theo đường đúng trong số một số các đường có thể đi qua **liên** mạng để tới đích.

Mạng toàn cầu GAN (Global Area Network): phạm vi của mạng trải rộng toàn Trái đất. Đây là mạng kết nối nhiều máy tính khác nhau của các châu lục. Tương tự như mạng WAN, kết nối của mạng GAN cũng được mạng viễn thông thực hiện. Ngoài ra, kết nối của GAN còn do vệ tinh đảm trách.

Tuy nhiên về sau người ta thường quan niệm chung bằng cách đồng nhất 4 loại thành 2 loại sau:

- ☐ **WAN** là mạng lớn trên diện rộng, hệ mạng này có thể truyền thông và trao đổi dữ liệu với một phạm vi lớn có **khoảng** cách xa như trong một quốc gia hay quốc tế.
- ☐ **LAN** **là** mạng cục bộ được bố trí trong phạm vi hẹp như một cơ quan, một Bộ, Ngành ... một số mạng LAN có thể nối lại với nhau để tạo thành một mạng LAN lớn hơn

b. Phân loại mạng theo topology

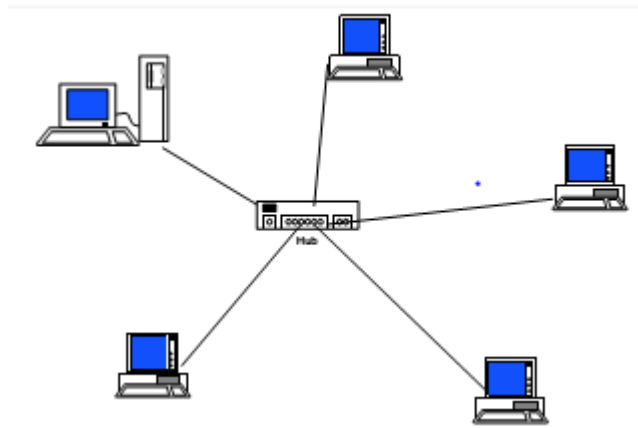
Các thành phần trong mạng LAN cục bộ có thể được sắp xếp, kết nối theo nhiều kiểu khác nhau và chúng được gọi là các Topology. Theo topology, mạng được chia làm các loại như mạng hình sao (Star topology), mạng tuyến tính (Bus topology), mạng vòng (Ring topology) và mạng kết hợp.

☐ **Mạng hình sao** (Start topology)

Mạng hình **sao** là một mô hình mạng bao gồm một thiết bị làm trung tâm và các nút thông tin chịu sự điều khiển của trung tâm đó. Các nút thông tin ở đây có thể là các máy trạm, các thiết bị đầu cuối hay các thiết bị khác trong hệ thống LAN.

Thiết bị trung tâm của mạng có vai trò quản lý, kiểm soát các hoạt động trong hệ thống, cụ thể với các chức năng như: theo dõi, kiểm duyệt và xử lý sai trong quá trình xử lý thông tin giữa các thiết bị, xác nhận cấp địa chỉ gửi nhận

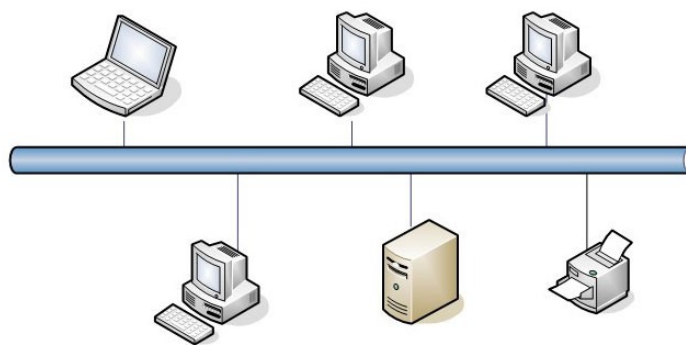
có quyền chiếm tuyến thông tin cũng như liên lạc với nhau và thông báo về các trạng thái của hệ thống mạng.



Hình 1.5: Mô hình **mạng** hình sao

□ **Mạng tuyến tính** (Bus topology)

Đây là một kiểu Topology mà tất cả các thiết bị như máy chủ, máy trạm, các nút thông tin đều được liên kết với nhau trên một đường dây cáp chính để truyền dữ liệu. Phía hai đầu dây cáp được bịt kín bằng hai thiết bị terminator. Các dữ liệu và tín hiệu truyền qua dây cáp đều mang theo địa chỉ cụ thể của điểm đến.



Hình 1.6: Mô hình **mạng** tuyến tính

- Ưu điểm: Dễ thiết kế và chi phí thấp, nếu một nút mạng hỏng thì không ảnh hưởng đến hoạt động của toàn mạng.

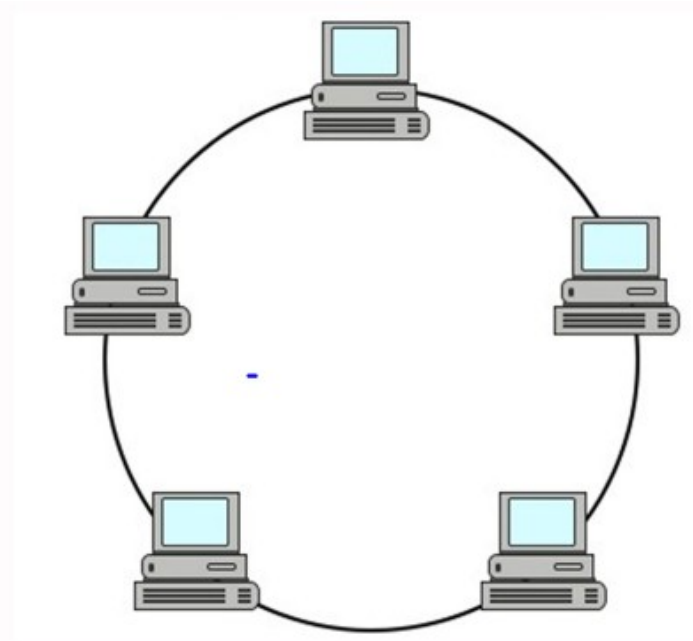
- Nhược điểm: Tính ổn định kém, dễ xảy ra xung đột thông tin trên đường truyền.

- Mạng vòng (Ring topology)

Với mạng vòng (Ring topology) tất cả các node cùng truy nhập chung trên một đường truyền vật lý. Tín hiệu được lưu chuyển trên vòng theo một chiều duy nhất, theo liên kết điểm - điểm. Dữ liệu được chuyển một cách tuần tự từng bit quanh vòng, qua các bộ chuyển tiếp. Bộ chuyển tiếp có ba chức năng: chèn, nhận và hủy bỏ thông tin. Các bộ chuyển tiếp sẽ kiểm tra địa chỉ đích trong các gói dữ liệu khi đi qua nó.

Đây là một kiểu Topology nơi các thiết bị được kết nối thành một vòng tròn khép kín thông qua dây cáp. Tín hiệu truyền sẽ được truyền đi theo một chiều cố định nào đó. Tại một thời điểm, chỉ có một thiết bị (một nút) được truyền tin qua một nút khác. Dữ liệu khi được truyền đi trong hệ thống mạng này phải kèm theo địa chỉ cụ thể của trạm tiếp nhận nó.

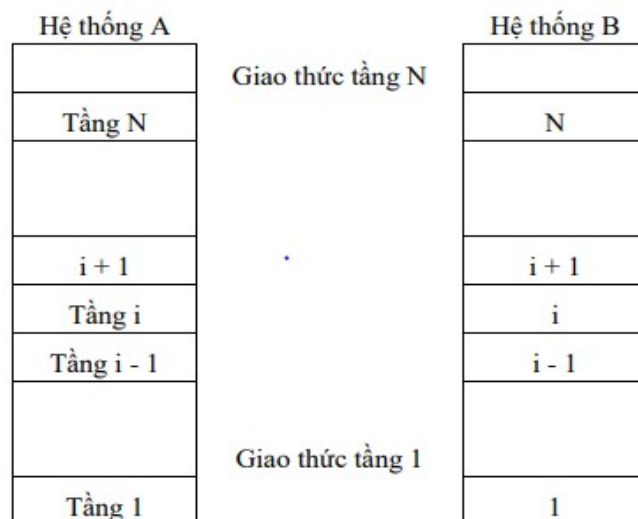
- Ưu điểm: Dễ dàng mở rộng hệ thống LAN ra xa hơn. Tiết kiệm được chiều dài dây cáp (cable) do không yêu cầu nhiều dây dẫn như hai dạng liên kết trên. Tốc độ mạng nhanh hơn mạng dạng tuyến (Bus Topology).
- Nhược điểm: Nhược điểm lớn nhất của Topology này là các thiết bị được nối theo một đường dây khép kín. Khi trên đường dây đó có bất kỳ điểm nào bị trục trặc thì cả hệ thống cũng ngừng hoạt động. Khó kiểm tra để tìm lỗi khi có sự cố.



Hình 1.7: Mô hình **mạng** vòng

1.3. Mô hình phân tầng

Để giảm phức tạp của việc thiết kế và cài đặt mạng, hầu hết các mạng máy tính đều có phân tích, thiết kế theo quan điểm phân tầng (layering). Sự phân tầng giao thức rất quan trọng vì nó cung cấp sự hiểu biết sâu sắc về các thành phần giao thức khác nhau cần thiết cho mạng và thuận tiện cho **vệc** thiết kế và cài đặt các phần mềm truyền thống. Mỗi tầng thực hiện một số chức năng xác định và cung cấp một số dịch vụ nhất định cho tầng cao hơn.



Hình 1.8: Kiến trúc phân tầng tổng quát

Nguyên tắc phân tầng:

- Mỗi hệ thống trong mạng đều có cấu trúc tầng (số lượng tầng và chức năng của mỗi tầng là như nhau).
- Giữa 2 tầng liền kề trong một hệ thống giao tiếp với nhau qua 1 giao diện qua đó xác định các hàm nguyên thủy và các dịch vụ tầng dưới cung cấp.
- Giữa hai tầng đồng mức ở hai hệ thống giao tiếp với nhau thông qua các luật lệ, qui tắc được gọi là giao thức.
- Trong thực tế, dữ liệu không được truyền trực tiếp từ tầng thứ i của hệ thống này sang tầng thứ i của hệ thống khác (trừ tầng thấp nhất). Mà việc kết nối giữa hai hệ thống được thực hiện thông qua hai loại

1.3.1 . Mô hình OSI

OSI – Open Systems Interconnection là một mô hình mạng hỗ trợ các ứng dụng giao tiếp với các thiết bị thông qua nhiều lớp khác nhau. Cụ thể hơn thì OSI cung cấp các kỹ thuật công nghệ và các thông số kỹ thuật hỗ trợ các chương trình phần mềm dễ dàng tương tác với nhau thông qua hệ thống mạng hoặc viễn thông hiện tại.

Mô hình OSI tổ chức các giao thức thành 7 tầng, mỗi tầng tập trung giải quyết một phần của tiến trình truyền thông, chia tiến trình truyền thông thành nhiều tầng và trong mỗi tầng có thể có nhiều giao thức khác nhau thực hiện các nhu cầu truyền thông cụ thể.



Hình 1.9: Mô hình OSI

- **Tầng vật lý:** Tầng vật lý là tầng thấp nhất trong mô hình 7 lớp OSI. Các thực thể tầng giao tiếp với nhau qua một đường truyền vật lý. Tầng vật lý xác định các chức năng, thủ tục về điện, cơ, quang để kích hoạt, duy trì và giải phóng các kết nối vật lý giữa các hệ thống mạng. Cung cấp các cơ chế về điện, hàm, thủ tục, ... nhằm thực hiện việc kết nối các phần tử của mạng thành một hệ thống bằng các phương pháp vật lý. Đảm bảo cho các yêu cầu về chuyển mạch hoạt động nhằm tạo ra các đường truyền thực cho các chuỗi bit thông tin. Các chuẩn trong tầng vật lý là các chuẩn xác định giao diện người sử dụng và môi trường mạng.
- **Tầng liên kết dữ liệu:** Tầng liên kết dữ liệu có chức năng chủ yếu là thực hiện thiết lập các liên kết, duy trì và hủy bỏ các liên kết dữ liệu, kiểm soát lỗi và kiểm soát lưu lượng. Chúng cung cấp các phương tiện có tính chức năng và quy trình để truyền dữ liệu giữa các thực thể mạng, phát hiện và có thể sửa chữa các lỗi trong tầng vật lý.
- **Tầng mạng:** Tầng mạng có nhiệm vụ tạo điều kiện thuận lợi cho việc truyền dữ liệu giữa hai mạng khác nhau. Nếu hai thiết bị giao tiếp trên cùng một mạng, thì tầng mạng là không cần thiết. Tầng mạng chia nhỏ các phân đoạn từ lớp truyền tải thành các đơn vị nhỏ hơn, được gọi là gói, trên thiết bị của người gửi và tập hợp lại các gói này trên thiết bị nhận. Tầng mạng cũng tìm ra con đường vật lý tốt nhất để dữ liệu đến đích của nó; điều này được gọi là định tuyến.
- **Tầng vận chuyển:** Là tầng cao nhất liên quan đến các giao thức trao đổi dữ liệu giữa các hệ thống mở, kiểm soát việc truyền dữ liệu từ nút tới nút (End-to-End). Thủ tục trong 3 tầng dưới (vật lý, liên kết dữ liệu và mạng) chỉ phục vụ việc truyền dữ liệu giữa các tầng kề nhau trong từng hệ thống. Các thực thể đồng tầng hội thoại, thương lượng với nhau trong quá trình truyền dữ liệu. Tầng vận chuyển thực hiện việc chia các gói tin lớn thành các gói tin nhỏ hơn trước khi gửi đi và đánh số các gói tin và đảm bảo chúng chuyển theo đúng thứ tự. Là tầng cuối cùng chịu trách nhiệm về mức độ an toàn trong truyền dữ liệu nên giao thức tầng vận chuyển phụ thuộc nhiều vào bản chất của tầng mạng. Tầng vận chuyển có thể thực hiện việc ghép kênh (multiplex) một vài liên kết vào cùng một liên kết nối để giảm giá thành.

- **Tầng phiên:** Tầng **phiên** cho phép người sử dụng trên các máy khác nhau thiết lập, duy trì và đồng bộ **phiên** truyền thông giữa họ với nhau. Nói cách khác tầng phiên thiết lập “các giao dịch” giữa các thực thể đầu cuối.
- **Tầng trình bày:** Tầng trình bày giải quyết các vấn đề liên quan đến các cú pháp và ngữ nghĩa của thông tin được truyền. Biểu diễn thông tin người sử dụng phù hợp với thông tin làm việc của mạng và ngược lại. Thông thường biểu diễn thông tin các ứng dụng nguồn và ứng dụng đích có thể khác nhau bởi các ứng dụng được chạy trên các hệ thống có thể khác nhau. Tầng trình bày phải chịu trách nhiệm chuyển đổi dữ liệu gửi đi trên mạng từ một loại biểu diễn này sang một loại biểu diễn khác. Để đạt được điều đó nó cung cấp một dạng biểu diễn truyền thông chung cho phép chuyển đổi từ dạng biểu diễn cục bộ sang biểu diễn chung và ngược lại
- **Tầng ứng dụng:** Nhiệm vụ của tầng này là xác định giao diện giữa người sử dụng và môi trường OSI. Bao gồm nhiều giao thức ứng dụng cung cấp các phương diện cho người sử dụng truy cập vào môi trường mạng và cung cấp các dịch vụ phân tán. Khi các thực thể ứng dụng AE (Application Entity) được thiết lập, nó sẽ gọi đến các phần tử dịch vụ ứng dụng ASE (Application Service Element). Mỗi thực thể ứng dụng có thể gồm một hoặc nhiều các phần tử dịch vụ ứng dụng. Các phần tử dịch vụ ứng dụng được phối hợp trong môi trường của thực thể ứng dụng thông qua các liên kết gọi là đối tượng liên kết đơn SAO (Single Association Object). SAO điều khiển việc truyền thông và cho phép tuần tự hóa các sự kiện truyền thông.

1.3.2. Mô hình TCP/IP

TCP/IP viết tắt của Transmission Control Protocol/Internet Protocol – Giao thức điều khiển truyền nhận/ Giao thức liên mạng. Đây là một bộ các giao thức truyền thông được sử dụng để kết nối các thiết bị mạng với nhau trên internet.

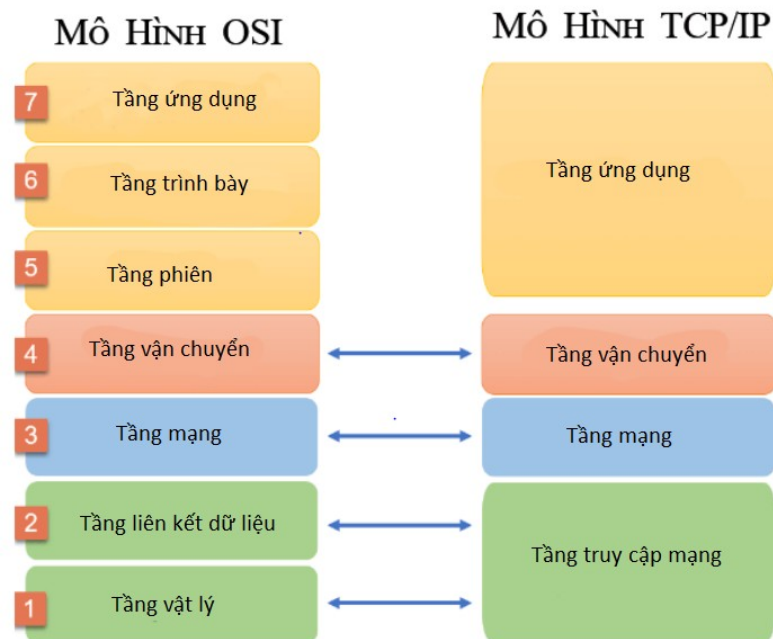
TCP/IP cũng có thể được sử dụng như một giao thức truyền thông trong mạng máy tính riêng (mạng nội bộ). Trong đó, bộ Giao thức internet – một tập hợp các quy tắc và thủ tục – thường gọi là TCP/IP. TCP và IP là hai giao thức chính bên cạnh những giao thức khác trong bộ. Bộ giao thức TCP/IP hoạt động như một lớp trừu tượng giữa các ứng dụng internet và hạ tầng router/switch.



Hình 1.10. Mô hình TCP/IP

- **Tầng ứng dụng** (Application Layer): Đây là lớp giao tiếp trên cùng của mô hình. Đúng với tên gọi, tầng Ứng dụng đảm nhận vai trò giao tiếp dữ liệu giữa 2 máy khác nhau thông qua các dịch vụ mạng khác nhau (duyệt web, chat, gửi email, một số giao thức trao đổi dữ liệu: SMTP, SSH, FTP,...). Dữ liệu khi đến đây sẽ được định dạng theo kiểu Byte nối Byte, cùng với đó là các thông tin định tuyến giúp xác định đường đi đúng của một gói tin.
- **Tầng vận chuyển** (Transport Layer): Chức năng chính của tầng 3 là xử lý vấn đề giao tiếp giữa các máy chủ trong cùng một mạng hoặc khác mạng được kết nối với nhau thông qua bộ định tuyến. Tại đây dữ liệu sẽ được phân đoạn, mỗi đoạn sẽ không bằng nhau nhưng kích thước phải nhỏ hơn 64KB. Cấu trúc đầy đủ của một Segment lúc này là Header chứa thông tin điều khiển và sau đó là dữ liệu. Trong tầng này còn bao gồm 2 giao thức cốt lõi là TCP và UDP. Trong đó, TCP đảm bảo chất lượng gói tin nhưng tiêu tốn thời gian khá lâu để kiểm tra đầy đủ thông tin từ thứ tự dữ liệu cho đến việc kiểm soát vấn đề tắc nghẽn lưu lượng dữ liệu. Trái với điều đó, UDP cho thấy tốc độ truyền tải nhanh hơn nhưng lại không đảm bảo được chất lượng dữ liệu được gửi đi.
- **Tầng mạng** (Internet Layer): Gần giống như tầng mạng của mô hình OSI. Tại đây, nó cũng được định nghĩa là một giao thức chịu trách nhiệm truyền tải dữ liệu một cách logic trong mạng. Các phân đoạn dữ liệu sẽ được đóng gói (Packets) với kích thước mỗi gói phù hợp với mạng chuyển mạch mà nó dùng để truyền dữ liệu. Lúc này, các gói tin được chèn thêm phần Header chứa thông tin của tầng mạng và tiếp tục được chuyển đến tầng tiếp theo. Các giao thức chính trong tầng là IP, ICMP và ARP.

- **Tầng truy nhập mạng** (Network Access Layer): là sự kết hợp của tầng Data Link và Physical trong mô hình OSI. Là tầng thấp nhất trong mô hình TCP/IP. Chịu trách nhiệm truyền dữ liệu giữa các thiết bị trong cùng một mạng. Tại đây, các gói dữ liệu được đóng vào khung (Frame) và được định tuyến đi đến đích được chỉ định ban đầu.



Hình 1.11: Mô hình OSI và TCP/IP

1.4. Các giao thức mạng

Internet cung cấp hai giao thức giao vận cho tầng ứng dụng là UDP và TCP. Khi xây dựng ứng dụng cho Internet, một trong những quyết định đầu mà nhà **thieeys** kế phải đưa ra là sử dụng UDP hay TCP. Mỗi giao thức cung cấp một kiểu phục vụ khác nhau cho ứng dụng.

1.4.1. Giao thức TCP

Đặc trưng của giao thức TCP là hướng kết nối và cung cấp dịch vụ truyền dữ liệu tin cậy.

- **Hướng kết nối** (Connection oriented): TCP client và TCP server trao đổi các thông tin điều khiển với nhau trước khi truyền dữ liệu ứng dụng. Quá trình “bắt tay” giữa client và server như vậy cho phép cả hai bên sẵn sàng xử lý các gói dữ liệu. Sau quá trình này, xuất hiện một **đường** kết nối TCP (TCP connection) giữa socket của hai tiến trình. Đây là kết nối hai chiều (song công – full duplex) vì cho phép hai **tiền trình** có thể **đồng** thời gửi và nhận dữ liệu. Khi ứng dụng kết thúc việc gửi thông điệp, nó đóng kết nối lại. Dịch vụ này chỉ là hướng kết nối chứ không phải **mạch** ảo (virtual circuit), bởi vì hai tiến trình được kết nối một cách lỏng lẻo.
- Dịch vụ giao vận tin cậy: Tiến trình gửi có thể sử dụng TCP để truyền dữ liệu chính xác và đúng thứ tự. Gửi đi một luồng byte qua socket, tiến trình ứng dụng có thể tin tưởng TCP sẽ chuyển luồng byte này đến socket nhận, không bị lỗi hay trùng lặp byte.

TCP cũng có cơ chế kiểm soát tắc nghẽn, cơ chế này đáp ứng cho cả Internet chứ không phải cho hai tiến trình truyền thông với nhau. Kỹ thuật kiểm soát tắc nghẽn của TCP là giảm tốc độ gửi dữ liệu của mỗi tiến trình (client hay server) khi mạng bị tắc nghẽn.

Giới hạn tốc độ truyền có thể không **thoả** mãn với các ứng dụng audio và video theo thời gian thực, những ứng dụng đòi hỏi phải có một băng thông tối thiểu. Hơn nữa, ứng dụng thời gian thực chấp nhận **mất** mát dữ liệu và không thực sự cần đến một dịch vụ giao vận tin cậy hoàn toàn. Vì các lý do đó, các ứng dụng thời gian thực thường chạy trên nền UDP.

Một số dịch vụ mà TCP không cung cấp. Thứ nhất, TCP không bảo đảm một tốc độ truyền tối thiểu. Tiến trình gửi không được phép truyền với bất kỳ tốc độ **nào** **nó** đề nghị, tốc độ này được kiểm soát bởi cơ chế kiểm soát tắc **nghiên** của TCP. Đôi khi cơ chế này khiến tiến trình **gửi** phải gửi với các độ trung bình trong đôi thấp. Thứ hai, TCP không dành ra bất kỳ sự bảo đảm nào về độ trễ. Khi tiến trình gửi chuyển dữ liệu cho socket TCP, dữ liệu cuối cùng sẽ đến được socket nhận, nhưng TCP không bảo đảm dữ liệu sau bao lâu mới tới được đích. Với những quan sát trên môi trường Internet thực, có thể phải chờ vài giây, thậm chí đến vài phút để TCP gửi thành công một thông điệp (ví dụ, một trang Web HTML từ Web server đến Web client). Nói tóm lại, TCP bảo đảm việc truyền tất cả dữ liệu một cách chính xác, nhưng không bảo đảm về tốc độ **truyền** và độ trễ.

1.4.2. Giao thức UDP

UDP là giao thức giao vận khá đơn giản, với mô hình phục vụ tối thiểu. UDP không hướng kết nối, nghĩa là không có giai đoạn "bắt tay" trước khi hai tiến trình bắt đầu trao đổi dữ liệu. UDP không cung cấp dịch vụ truyền tin cậy. Khi tiến trình gửi **chuyển** thông điệp qua cổng UDP, UDP không đảm **bảo** thông điệp sẽ đến được cổng tiến trình nhận. Hơn nữa, các thông điệp đến đích có thể không đúng thứ tự.

Mặt khác, UDP không có cơ chế kiểm soát tắc nghẽn, vì vậy tiến trình gửi có thể **đẩy** dữ liệu ra cổng UDP với tốc độ bất kỳ. Mặc dù, không phải Tất cả dữ liệu đều **tới** được đích, nhưng phần lớn dữ liệu có thể tới được. Ứng dụng thời gian thực thường lựa chọn UDP ở tầng giao vận. Giống TCP. UDP không bảo đảm về độ trễ.

Bảng 1.1 trình bày các giao thức giao vận của các ứng dụng mạng phổ biến. Thư điện tử, truy cập từ xa. Web và truyền file sử dụng TCP do TCP cung cấp dịch vụ truyền dữ liệu tin cậy, bảo đảm rằng mọi dữ liệu sẽ tới được đích

Người ta thấy rằng, điện thoại qua Internet chạy trên nền UDP, mỗi phía của ứng dụng này cần gửi dữ liệu qua mạng với tốc độ tối thiểu nào đó (Bảng 1,1). Hơn nữa, ứng dụng điện thoại qua Internet chấp nhận mất mát dữ liệu. Vì thế chúng không cần dịch vụ truyền tin cậy của TCP.

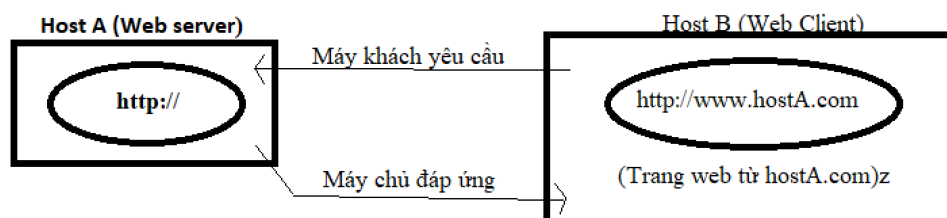
Ứng dụng	Giao thức ứng dụng	Giao thức giao vận
Thư điện tử	SMTP [RFC 821]	TCP
Truy cập từ xa	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2068]	TCP
Truyền File	FTP [RFC 959]	TCP
Remote File Server	NFS	UDP hoặc TCP
Streaming Multimedia	Giao thức riêng, không công bố (ví dụ Real NetWork)	UDP hoặc TCP
Điện thoại Internet	Giao thức riêng, không công bố (ví dụ Vocaltec)	Thường là UDP

Bảng 1.1. Các ứng dụng phổ biến và giao thức giao vận tương ứng

1.5. Mô hình ứng dụng mạng

1.5.1. Mô hình Khách – Máy chủ (Client – Server)

Mô hình Client/Server mô hình mạng máy tính bao gồm 2 thành phần chính là máy khách (client) và máy chủ (server). Trong mô hình này, server là nơi lưu trữ tài nguyên, cài đặt các chương trình dịch vụ và thực hiện các yêu cầu của client. Client đón vai trò gửi yêu cầu đến server. Client gồm máy tính và thiết bị điện tử nói chung. Mô hình Client server cho phép mạng tập trung các ứng dụng và chức năng tại một hoặc nhiều máy chủ dịch vụ file chuyên dụng. Các máy này trở thành trung tâm của hệ thống. Hệ điều hành của Client server cho phép người dùng chia sẻ đồng thời cùng một tài nguyên, không quan trọng vị trí địa lý.



Hình 1.12. Mô hình Khách – Máy chủ (Client – Server)

Client hay chính là máy khách, máy trạm – là nơi gửi yêu cầu đến server. Nó tổ chức giao tiếp với người dùng, server và môi trường bên ngoài tại trạm làm việc. Client tiếp nhận yêu cầu của người dùng sau đó thành lập các query string để gửi cho server. Khi nhận được kết quả từ server, client sẽ tổ chức và trình diễn những kết quả đó.

Server xử lý yêu cầu gửi đến từ client. Sau khi xử lý xong, server sẽ gửi trả lại kết quả, client có thể tiếp tục xử lý các kết quả này để phục vụ người dùng. Server giao tiếp với môi trường bên ngoài và client tại server, tiếp nhận yêu cầu dưới dạng query string (xâu ký tự). Khi phân tích xong các xâu ký tự, server sẽ xử lý dữ liệu và gửi kết quả về cho client.

1.5.2. Mô hình ngang hàng (Peer to peer)

Mô hình ngang hàng là một mạng máy tính trong đó hoạt động của mạng chủ yếu dựa vào khả năng tính toán và băng thông của các máy tham gia chứ không tập trung vào một số nhỏ các máy chủ trung tâm như các mạng thông thường. Không giống như Máy khách-Máy chủ, mô hình ngang hàng không phân biệt giữa máy khách và máy chủ thay vào đó mỗi nút có thể là máy khách hoặc máy chủ tùy thuộc vào việc nút đó có yêu cầu hoặc cung cấp dịch vụ hay không.

1.5.3. Mô hình lai (Hybrid)

Đây là mô hình kết hợp giữa Client-Server và Peer-to-Peer. Phần lớn các mạng máy tính trên thực tế thuộc mô hình này.

1.6. Tổng quan về ngôn ngữ python

1.6.1. Giới thiệu ngôn ngữ Python

Là ngôn ngữ lập trình bậc cao, phục vụ cho các mục đích lập trình đa năng. Ưu điểm nổi bật nhất đó chính là dễ đọc, dễ nhớ, dễ học. Python là ngôn ngữ có cấu trúc tương đối rõ ràng, thuận tiện cho người mới học lập trình. Cấu trúc của python còn cho phép người dùng sử dụng để viết mã lệnh với số lần gõ phím tối thiểu.

Thậm chí trong năm 2021, Python được đánh giá là Top 2 ngôn ngữ lập trình nên học và top 1 ngôn ngữ lập trình được yêu thích nhất. Chính vì thế, Python trở thành lựa chọn hàng đầu cho người mới, hay thậm chí là các lập trình viên chuyên nghiệp

1.6.2. Các giai đoạn phát triển của Python

Để có được một ngôn ngữ lập trình hữu ích như hiện nay, Python đã phải trải qua nhiều giai đoạn phát triển của mình. Quá trình phát triển của Python gồm có 3 giai đoạn:

- Giai đoạn Python 1 – phiên bản 1.x:

Vào đầu năm 1991, lập trình viên vĩ đại **tại** người Ba Lan – Guido Van Rossum đã khai sinh ra ngôn ngữ lập trình Python cơ bản. Vào thời điểm đó, Python được tạo ra dùng để chạy trên hệ điều hành Unix. Phiên bản tối ưu nhất trong giai đoạn này là 1.6.1

- Giai đoạn 2 – Phiên bản 2.x:

Năm 2000, Rossum và đội ngũ nghiên cứu Python chuyển đi và thành lập nên BeOpen Python Labs Team và cho ra mắt phiên bản Python 2.0. Tuy nhiên, đến phiên bản 2.1 thì ngôn ngữ này thuộc về quyền sở hữu của Python Software Foundation. Đến năm 201 thì Python chính thức ngừng cập nhật tại phiên bản 2.7.

- Giai đoạn 3 – Phiên bản 3.x:

Python 3.0 ra đời năm 2008 với sứ mệnh cải thiện và loại bỏ những điểm **trùng lặp** trong cấu trúc và câu lệnh của phiên bản 2.x. Khi sử dụng sẽ có những tiện ích giúp chuyển đổi 2.x sang 3.x một cách thuận tiện.

1.6.3. Các tính năng nổi bật của ngôn ngữ python

- **Miễn phí, mã nguồn mở:** Có thể thỏa mái sử dụng và phân phối python thậm chí là có thể sử dụng chúng để phục vụ cho mục đích thương mại. Bởi chúng là mã nguồn mở, ta không chỉ sử dụng các phần mềm, chương trình được viết trong python mà còn có thể thay đổi mã nguồn. Python có một cộng đồng lớn, thường xuyên cập nhật, không ngừng cải tiến.
- **Ngôn ngữ lập trình đơn giản, dễ đọc:** Python có cấu trúc ngữ pháp đơn giản, rõ ràng. Nó dễ đọc và viết đơn giản hơn nhiều khi so sánh với ngôn ngữ lập trình khác như C ++, Java, C#. Python làm cho việc lập trình trở nên thú vị, giúp các lập trình viên tập trung vào những giải pháp chứ không phải là cú pháp.
- **Khả năng di chuyển:** Các chương trình trên python có thể di chuyển từ nền tảng này đến nền tảng khác mà không gặp phải bất kỳ thay đổi nào khi chạy. Nó chạy liền mạch trên các nền tảng như Mac, Windows, Linux.
- **Khả năng mở rộng và có thể nhúng:** Nếu một ứng dụng đòi hỏi sự phức tạp lớn, ta có thể dễ dàng kết hợp với các phần code bằng C, C ++ và những ngôn ngữ khác vào code python. Bởi vậy, sẽ giúp ứng dụng của ta có những tính năng tốt hơn, khả năng scripting mà các ngôn ngữ lập trình khác khó có thể làm được.

- **Hướng đối tượng:** Mọi thứ của python đều hướng đối tượng. Lập trình đối tượng sẽ giúp giải quyết các vấn đề một cách trực quan nhất. Với lập trình đối tượng, ta có thể phân chia nhiều phức tạp thành các tập nhỏ hơn bằng cách tạo ra các đối tượng.
- **Thư viện tiêu chuẩn lớn:** Ngôn ngữ python có thư viện lớn giúp cho việc lập trình trở nên dễ dàng hơn vì không phải tự viết tất cả các code. Những thư viện này được kiểm tra kỹ lưỡng, nên chắc chắn nó không làm hỏng code hay ứng dụng nào của ta.

1.6.4. Các thành phần và cú pháp cơ bản trong chương trình Python

a. Từ khóa (keyword) trong python

Từ khóa (keyword) là những từ (word) được dành riêng trong Python. Ta không thể sử dụng từ khóa để đặt tên biến, tên hàm hoặc bất kỳ định danh (identifier) nào khác. Chúng được sử dụng để xác định cú pháp và cấu trúc của ngôn ngữ Python. Trong Python, các từ khóa có sự phân biệt chữ hoa và chữ thường.

Với phiên bản Python 3.10.3 hiện tại em đang sử dụng có tất cả 35 từ khóa. Tất cả các từ khóa ngoại trừ True, False và None đều ở dạng chữ thường. Chúng ta có thể sử dụng lệnh help("keywords") trong **trình** thông dịch Python để xem danh sách tất cả các từ khóa trong Python.

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
And	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Bảng 1.2. Bảng các keyword trong Python 3.10.3

b. Định danh (identifier) trong Python

Định danh (identifier) là tên được đặt cho các thực thể như lớp, hàm, biến,... Định danh giúp phân biệt thực thể này với thực thể khác.

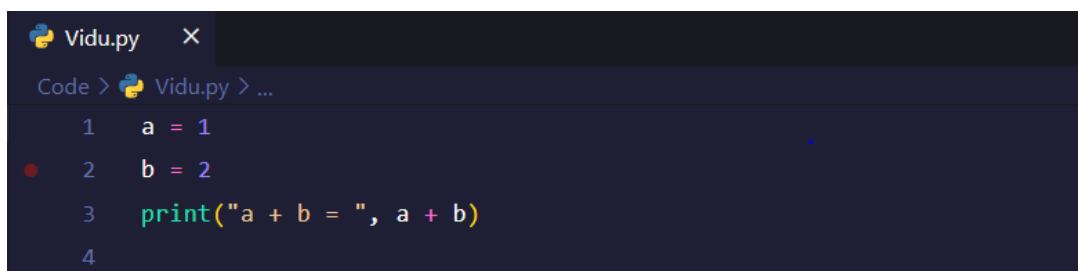
Những quy tắc khi đặt tên định danh:

- Tên định danh có thể bao gồm các chữ thường (a đến z), chữ hoa (A đến Z), chữ số (0 đến 9), dấu gạch dưới `_`. Ví dụ, `myClass`, `var_1` và `print_this_to_screen` là các tên định danh hợp lệ.

- ☐ Tên định danh không được bắt đầu bằng một chữ số. Ví dụ, tên định danh *1variable* không hợp lệ nhưng *variable1* thì hợp lệ.
- ☐ Không được đặt tên định danh giống với từ khóa (keyword).
- ☐ Không được sử dụng các ký hiệu đặc biệt như *!, @, #, \$, %, ...* trong tên định danh.
- ☐ Tên định danh có thể có độ dài bất kỳ.

c. Câu lệnh (statement) trong Python

Python sẽ thông dịch từng câu lệnh (statement) để thực thi. Một statement trong Python thường được viết trong 1 dòng. Ta không cần thiết phải thêm dấu chấm phẩy ; vào cuối mỗi câu lệnh. Ví dụ:



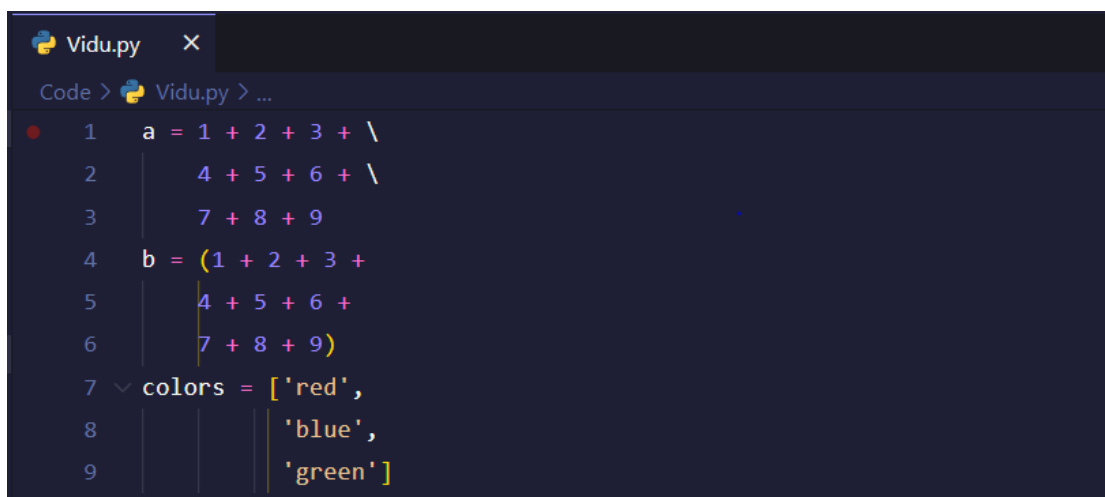
```

Vidu.py
Code > Vidu.py > ...
1  a = 1
2  b = 2
3  print("a + b = ", a + b)
4

```

Hình 1.13. Ví dụ về câu lệnh trong Python

Ta có thể viết một câu lệnh trên nhiều dòng bằng cách sử dụng thích hợp các ký tự như ký tự tiếp tục (**), dấu ngoặc đơn (*()*), ngoặc vuông [*]*, ngoặc nhọn *{}*.



```

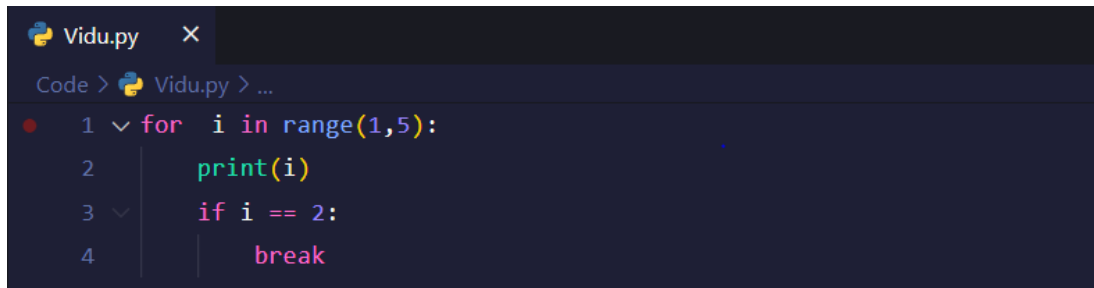
Vidu.py
Code > Vidu.py > ...
1  a = 1 + 2 + 3 + \
2      4 + 5 + 6 + \
3      7 + 8 + 9
4  b = (1 + 2 + 3 +
5      4 + 5 + 6 +
6      7 + 8 + 9)
7  colors = ['red',
8            'blue',
9            'green']

```

Hình 1.14. Ví dụ về câu lệnh nhiều dòng trong Python

d. Thụt đầu dòng (indentation) trong python

Python sử dụng thụt đầu dòng (indentation) để định nghĩa một khối lệnh (code block) như thân hàm, thân vòng lặp, ... Lưu ý: Python không sử dụng dấu ngoặc nhọn *{}* cho code block như các ngôn ngữ C/C++, Java, ... Ví dụ:



```

1  for i in range(1,5):
2      print(i)
3      if i == 2:
4          break

```

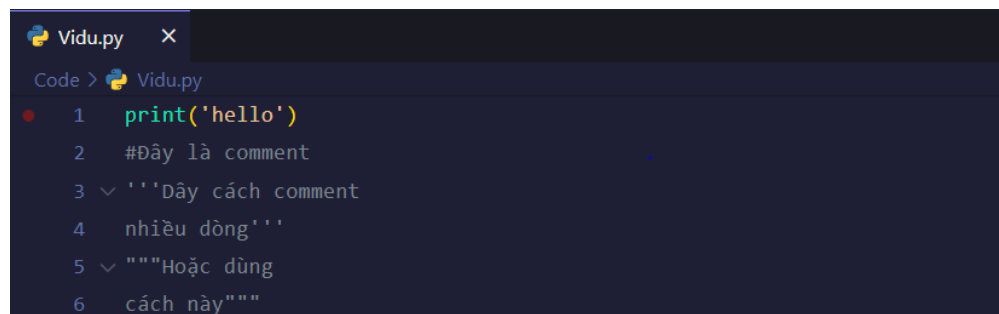
Hình 1.15. Ví dụ về thụt đầu dòng trong python

Trong python, 4 khoảng **trắng** được sử dụng cho thụt đầu dòng và được ưu tiên hơn các tab. Nếu sử dụng indentation không chính xác thì chương trình sẽ báo lỗi **IndentationError**.

e. Ghi chú (comment) trong python

Ghi chú (comment) được sử dụng để giải thích code đang thực hiện những gì. Việc này rất quan trọng khi đọc lại source code, bảo trì chương trình sau này. Sử dụng ký hiệu hash (#) để bắt đầu viết comment trong Python. Trình thông dịch Python sẽ bỏ qua comment bắt đầu từ ký hiệu hash (#) cho đến khi gặp ký tự bắt đầu dòng mới.

Ta có thể viết comment trên nhiều dòng, mỗi dòng bắt đầu bằng ký tự hash(#), Hoặc một cách khác để comment trên nhiều dòng bằng cách sử dụng dấu ''' hoặc """". Ví dụ:



```

1  print('hello')
2  #Đây là comment
3  '''Đây cách comment
4  nhiều dòng'''
5  """Hoặc dùng
6  cách này"""

```

Hình 1.16. Ví dụ về comment trong Python

f. Khối lệnh (code block) trong Python

Một hoặc nhiều câu lệnh (statement) có thể tạo thành một khối lệnh (code block). Các khối lệnh thường gặp trong Python như các lệnh trong lớp (class), hàm (function), vòng lặp (loop),... Python sử dụng thụt đầu dòng (indentation) để bắt đầu định nghĩa, phân tách một code block với các code block khác. Ví dụ:

```
1  import sys                                khối lệnh chính
2
3  file = "text.txt"
4  try:
5      myFile = open(file, "r")                khối lệnh thân try
6      myLine = myFile.readline()
7      while myline:
8          print(myline)                        khối lệnh thân while
9          myline = myFile.readline()
10     myFile.close()
11 except IOError as e:
12     print("I/O error({0}): {1}".format(e.errno, e.strerror))
13 except:
14     print("Unexoected error: ", sys.exc_info()[0])
15 print("done")
```

Hình 1.17. Ví dụ về khối lệnh trong Python

CHƯƠNG 2: LẬP TRÌNH SOCKET

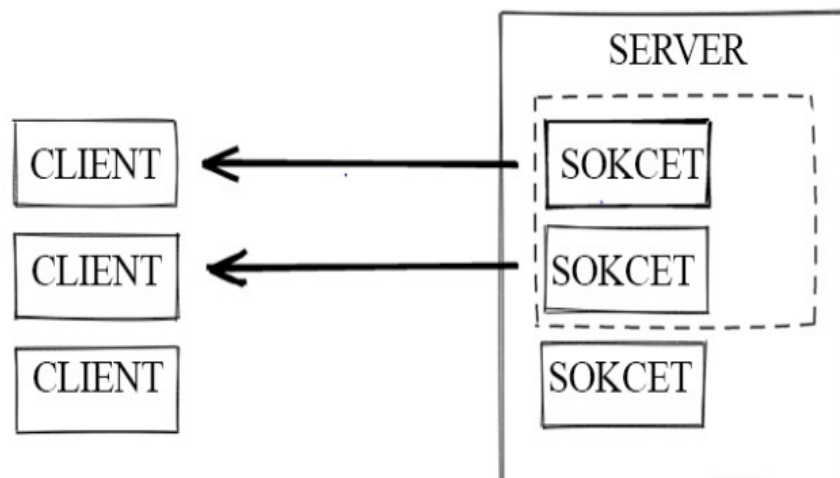
2.1. Tìm hiểu về Socket

Socket là một giao diện lập trình (API – Application Program Interface) ứng dụng mạng thông qua giao diện này có thể lập trình điều khiển việc truyền thông giữa 2 máy sử dụng các giao thức mức thấp như TCP,UDP... Giữa hai chương trình chạy trên mạng cần có một liên kết giao tiếp hai chiều, hay còn gọi là two-way communication để kết nối 2 process trò chuyện với nhau. Điểm cuối (endpoint) của liên kết này được gọi là Socket.

Ưu điểm lớn nhất khiến socket trở nên cần thiết là nó có thể tương thích với hầu hết các hệ điều hành, từ Windows, Linux cho đến Mac OS X... Ngoài ra, socket còn có thể kết hợp được với rất nhiều ngôn ngữ lập trình như: C, C++, Java, Visual Basic, Visual C++... Có thể thấy rằng socket phù hợp để sử dụng ở mọi điều kiện cũng như hoàn cảnh khác nhau.

Đặc biệt là lập trình viên có thể chạy song song nhiều socket trong cùng một lúc. Điều này giúp rút ngắn thời gian và tăng hiệu suất hoạt động.

Cách thức hoạt động của Socket: Thông qua TCP/IP và UDP, socket sẽ tiến hành truyền và nhận dữ liệu Internet. Từ đó tạo nên một cầu nối giữa client và server. Điều kiện để hoạt động này diễn ra là có đủ thông tin về thông số IP và **giữ** liệu cổng của 2 process muốn kết nối với nhau. Hai ứng dụng muốn liên kết có thể nằm cùng trên 1 máy hoặc 2 máy khác nhau đều được. Tuy nhiên, nếu 2 process cùng nằm trên một máy, các số liệu bắt buộc phải khác nhau hoàn toàn. Đây là điều kiện cần thiết để socket io có thể tiến hành hoạt động.



Hình 2.1. Cách hoạt động của Socket

2.2. Phân loại Socket

2.2.1. Stream Socket

Stream Socket còn được gọi là socket TCP. Nó hoạt động dựa trên giao thức hướng kết nối. Tức là chỉ sử dụng được khi máy chủ và máy khách có sự liên kết.

Stream Socket giúp quá trình truyền gửi diễn ra nhanh chóng và đúng hạn. Ngoài ra, với yếu tố đảm bảo, người dùng cũng có thể yên tâm rằng dữ liệu sẽ được chuyển đến đúng người nhận với độ tin cậy tuyệt đối. Mỗi hành động diễn ra trong quá trình dịch chuyển thông tin đều sẽ được ghi lại kết quả và truyền về cho người dùng dù có thành công hay không. Các bản ghi dữ liệu cũng không hề có giới hạn nào, bạn có thể thoải mái truyền bao nhiêu thông tin tùy thích. Song song với đó, Stream Socket còn sở hữu 2 cơ chế bao gồm quản lý luồng lưu thông trên mạng và chống tắc nghẽn nhằm tối ưu hóa thời gian truyền dữ liệu.

Điều kiện để sử dụng Stream Socket là phải có địa chỉ IP rõ ràng giữa 2 đầu kết nối. Các thông tin được gửi đi tuân tự theo kế hoạch lên trước. Mỗi thông điệp được thực hiện phải có thông báo trả về **mới** tính là hoàn thành. Ngoài ra, Stream Socket hoạt động dựa trên mô hình lắng nghe và chấp nhận. Có nghĩa rằng giữa 2 process phải có 1 bên yêu cầu kết nối trước.

2.2.2. Datagram socket

Datagram Socket hoạt động dựa trên giao thức UDP về việc truyền thông tin không yêu cầu sự kết nối. Để hoạt động này diễn ra, nó cung cấp connection-less point cho việc gửi và nhận thông tin. Chính vì thế mà Datagram Socket còn được gọi là socket không hướng kết nối.

Hai tiến trình có thể liên lạc với nhau **thông** qua Datagram Socket mà không cần IP chung. Thông điệp muốn gửi đi phải kèm theo thông điệp người nhận. Có thể gửi một thông điệp nhiều lần, tuy nhiên không thể gửi cùng một lúc. Ngoài ra, thứ tự hoàn thành dịch chuyển cũng không cố định, thông điệp gửi sau có thể đến trước và ngược lại.

Datagram Socket không đảm bảo tuyệt đối kết quả của tiến trình. Một số trường hợp ghi nhận thông điệp không thể đến tay của bên nhận. Cùng với đó, điều kiện để thực hiện các cuộc trao đổi 2 đầu là 1 trong 2 tiến trình phải công bố port của socket mà mình đang sử dụng.

Tuy nhiên vì không yêu cầu kết nối của 2 tiến trình nên quá trình truyền dữ liệu diễn ra vô cùng nhanh chóng, phù hợp để ứng dụng trong cách hoạt động như nhắn tin, chat game online...

2.2.3. Unix socket

Unix socket được biết đến như một điểm chuyển giao giữa các ứng dụng ở trong một máy tính. Vì không phải qua bước kiểm tra và routing nên quá trình truyền tin diễn ra vô cùng nhẹ nhàng và nhanh chóng. Đường chuyển khép kín đảm bảo không bị rò rỉ thông tin khi thực hiện.

Unix socket mang đến những ưu điểm tuyệt vời như: tăng tốc độ truy cập MySQL lên đến 30-50%, tăng PostgreSQL lên hơn 30%, tăng Redis lên 50%. Cùng với đó còn giảm thời gian latency xuống từ 60ms còn 5ms.

Bên cạnh đó, Unix socket vẫn còn một số nhược điểm tồn đọng như: không thể dịch chuyển giữa 2 máy khác nhau, đôi khi xảy ra delay do vấn đề phân quyền giữa các tệp tin.

2.2.4. Web socket

Không giống như 3 loại socket trên, Websockets được sử dụng nhiều nhất nhờ những ứng dụng to lớn mà nó mang lại.

Vậy websocket là gì? Websocket là một module hỗ trợ kết nối giữa hai đầu máy nhờ giao thức TCP mà không cần quan HTTP. Websocket được thiết kế chuyên dụng dành cho web nhưng vẫn có thể được dùng để ứng dụng cho các phần mềm.

Websocket sở hữu gần như hầu hết những ưu điểm của các loại socket khác như: tỷ lệ xảy ra delay thấp, dễ xử lý lỗi, khả năng dịch chuyển thông tin nhanh chóng và mạnh mẽ, phù hợp cho những hoạt động cần đến tính tức thời như chat realtime, chat online, biểu đồ chứng khoán...

2.2. Socket trong python

2.2.1. Một số khái niệm riêng của Socket

Các Socket có thể được triển khai thông qua các kênh khác nhau: domain, TCP, UDP, ... Thư viện socket cung cấp các lớp riêng để xử lý các trình truyền tải cũng như một Interface chung để xử lý phần còn lại.

Một số khái niệm riêng của Socket:

- domain: Là family của các giao thức protocol được sử dụng như là kỹ thuật truyền tải. Các giá trị này là các hằng như AF_INET, PF_INET, PF_UNIX, PF_X25, ...
- type: Kiểu giao tiếp giữa hai endpoint, đặc trưng là SOCK_STREAM cho các giao thức hướng **kết** nối (connection-oriented) và SOCK_DGRAM cho các giao thức không hướng kết nối.
- protocol: Đặc trưng là 0, mà có thể được sử dụng để nhận diện một biến thể của một giao thức bên trong một domain hoặc type.
- hostname: Định danh của một network interface:

- Một chuỗi, có thể là tên một host, địa **chỉ** IPV6,...
- Một chuỗi "<broadcast>", xác **định** một địa chỉ INADDR_BROADCAST
- Một **chuỗi** có độ dài là 0, xác định INADDR_ANY, hoặc một số nguyên được thông dịch dưới dạng một địa chỉ **nhị** phân trong thứ tự host byte.
- port: Mỗi Server nghe các lời gọi từ Client trên một hoặc nhiều cổng (port). Một port có thể là một chuỗi chứa số hiệu của port, một tên của một dịch vụ, ...

2.2.2. Mô-đun Socket trong Python

Mô-đun socket là một phần được tích hợp sẵn trong Python. Để sử dụng được mô-đun này ta cần nhập chúng vào chương trình Python như sau: *import socket*

Để tạo một Socket, ta phải sử dụng hàm `socket.socket()` có sẵn trong socket Module, có cú pháp như sau: *s = socket.socket (socket_family, socket_type, protocol=0)*

Trong đó:

- Tham số *socket_family* chỉ định **vùng** hay họ địa chỉ áp đặt cho socket. *socket_family* có thể nhận một trong các giá trị sau:

AF_UNIX	Mở Socket kết nối theo giao thức tập tin (xuất nhập socket dựa trên xuất nhaoj tập tin) của UNIX/Linux
AF_INET	Mở socket theo giao thức internet (sử dụng đại chỉ IP để kết nối)
AF_IPX	Vùng giao thức IPX (mạng Novell)
AF_ISO	Chuẩn giao thức ISO
AF_NS	Giao thức Xerox Network System

Bảng 2.1 Giá trị tham số socket_family trong hàm socket

Hầu như chỉ sử dụng AF_UNIX và AF_INET là chính. Các vùng giao tiếp **khác** đã lỗi thời và hiện này ít được sử dụng

- Tham số *type* trong hàm *socket()* dùng chỉ định kiểu giao tiếp hay truyền dữ liệu của socket. Có thể chỉ định hằng *SOCK_STREAM* dùng cho truyền dữ liệu bảo đảm hoặc *SOCK_DGRAM* dùng cho kiểu truyền **không** bảo đảm.
- Tham số *protocol* dùng để chọn giao thức áp dụng cho kiểu socket (trong trường hợp có nhiều giao thức áp dụng cho một kiểu truyền). Tuy nhiên chỉ cần đặt giá trị 0 (lấy giao thức mặc định). *AF_INET* chỉ cài đặt một

giao thức duy nhất cho các kiểu truyền *SOCK_STREAM* và *SOCK_DGRAM*, đó là *TCP* và *UDP*.

2.2.3. Các hàm socket sử dụng trong Python

Khi đã có đối tượng Socket, ta có thể sử dụng các hàm cần thiết để tạo Máy chủ. Một số hàm cần thiết sử dụng trong chương trình tạo Máy chủ:

- *s.bind()*: Phương thức này liên kết địa chỉ (*hostname*, *port*) với *socket*.
- *s.listen()*: Phương thức này thực hiện lắng nghe trên cổng (*port*) và *hostname* đã được liên kết ở hàm *s.bind()*. Việc lắng nghe được thực hiện trên giao thức *TCP*.
- *s.accept()*: Phương thức này sẽ thiết lập kết nối và chấp nhận thụ động kết nối máy khách *TCP*, đợi cho đến khi kết nối đến.

Khi Máy chủ được tạo ra, nó sẽ luôn lắng nghe và chấp nhận thụ **động** các kết nối của Khách tới Máy chủ. Để một Khách có thể kết nối đến một Server thì cần sử dụng hàm: *s.connect()*. Hàm này sẽ thực hiện kết nối từ Client tới một Server thông qua *hostname* và *port*.

Một số hàm thực hiện truyền dữ liệu từ Server về Client hoặc từ Client gửi lên Server:

- *s.recv()*: Phương thức này trả về **nhận** bản tin *TCP*
- *s.send()*: Phương thức này thực **hiện** truyền đi dữ liệu *TCP*
- *s.recvfrom()*: Phương thức nhận về bản tin *UDP*
- *s.sendto()*: Phương thức này thực hiện **truyền** đi dữ liệu *UDP*
- *s.close()*: Phương thức này đóng lại các Socket
- *socket.gethostname()*: Phương thức này trả về **tên** máy chủ

2.3. Demo chương trình **lập** trình socket bằng python.

Các bước khởi tạo, kết nối và trao đổi thông tin giữa Client và Server trong mô hình **Stream Socket**:

1. Khởi tạo một máy chủ (server) thông qua hàm *socket()*
2. Liên kết máy chủ với host và port thông qua hàm *bind()*
3. Cho phép máy chủ (server) lắng nghe trên port đó thông qua hàm *listen()*

4. Tạo yêu cầu kết nối từ máy khách (client) tới máy chủ thông qua hàm *connect()*
5. Từ phía máy chủ (server) sẽ chấp nhận yêu cầu kết nối của máy khách (client) thông qua hàm *accept()*
6. Sau khi máy khách đã kết nối được tới máy chủ, khi đó có thể thực hiện trao đổi thông tin giữa client – server thông qua hàm *read()/write()*
7. Việc trao đổi tin hoàn thành có thể đóng kết nối giữa client – server thông qua hàm *close()*

2.3.1. Tạo một Máy chủ (Server) trong Python

Sau khi một đối tượng socket được khởi tạo, ta có thể sử dụng hàm *s.bind()* để liên kết giữa host và port vào trong server. Sau khi liên kết thành công, ta gọi hàm *s.listen()* để lắng nghe các kết nối từ phía client vào server.

Tiếp theo, gọi phương thức *s.accept()* để chấp thuận kết nối giữa client tới server và thực hiện trao đổi thông tin giữa client – server bằng các hàm *s.read()/s.write()* (việc trao đổi thông tin sẽ chỉ thực hiện sau khi việc chấp thuận kết nối giữa client tới server hoàn tất).

Bước cuối cùng, khi việc trao đổi thông tin hoàn thành ta sẽ đóng kết nối lại bằng hàm *s.close()*.


```
Server.py X
Server.py > ...
1
2 import socket
3 #Tao doi tuong socket cho server
4 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5 #Lay ra ten host tren may (localhost)
6 host = socket.gethostname() #host = "localhost"
7 #Chi dinh cong cho ket noi
8 port = 8080
9 # Lien ket host va port voi may chu
10 s.bind((host, port))
11 # Lang nghe cac ket noi tu clien toi server (chi chap nhan 5 ket noi)
12 s.listen(5)
13 # Tao mot thong diep de truyen ve client
14 message = "Day la chương trình demo Socket Server"
15 while True:
16     # Chap thuan ket noi giua client toi server
17     c, addr = s.accept()
18     print ("Mot may khách đang thực hiện kết nối từ:", addr)
19     # Truyen thong diep tu server ve client
20     c.send(message.encode())
21     # Dong ket noi
22     c.close()
```

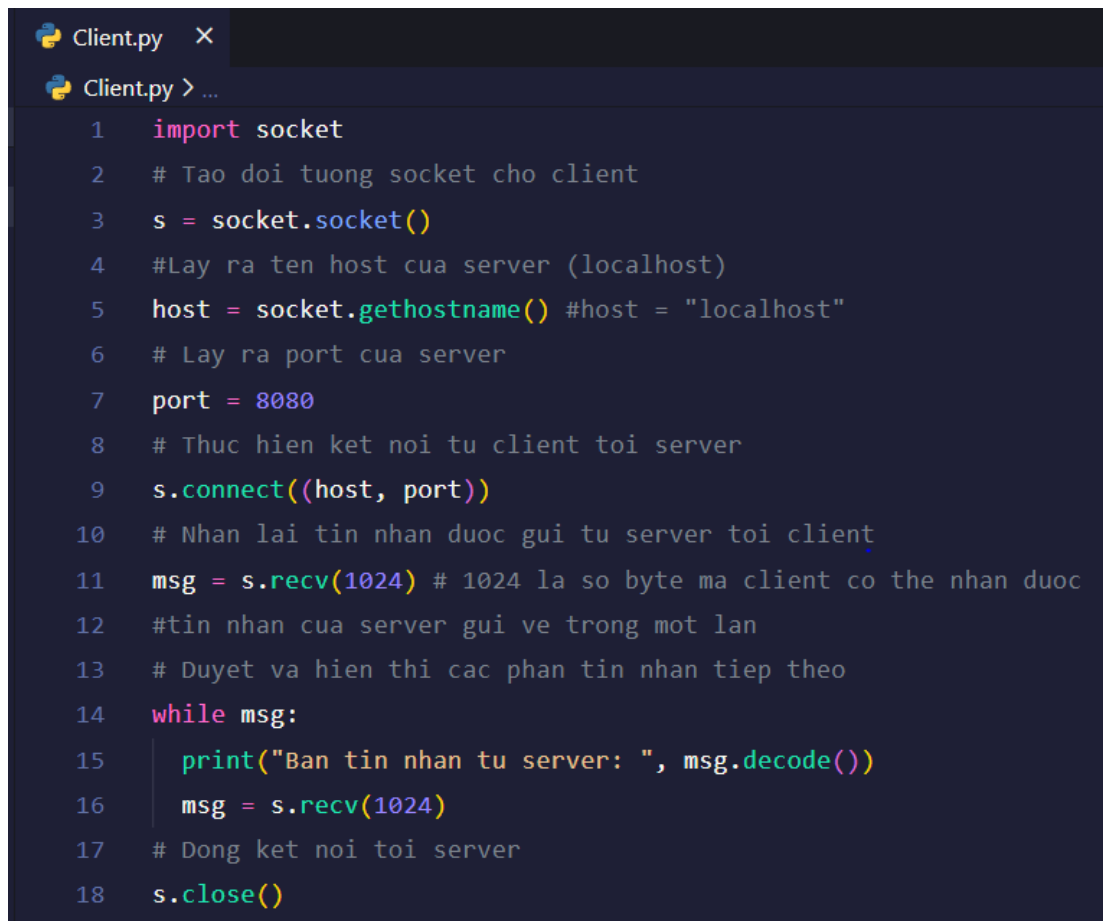
Hình 2.2. Demo chương trình Máy chủ (Server)

2.3.2. Tạo một máy khách (client) trong Python

Điều kiện để client kết nối tới server chính là cần có được thông tin của host và port của server cần kết nối tới. Sau đó sẽ gọi kết nối tới server thông qua hàm `s.connect()` – truyền vào `host` và port.

Hàm `s.connect(host, port)` sẽ mở một kết nối TCP với tên máy chủ trên cổng . Khi bạn đã thực sự kết nối tới server – khi đó ta sẽ nhận được các bản tin mà server sẽ gửi về cho ta bằng cách sử dụng hàm `s.recv(1024)` để nhận tin nhắn với độ dài 1024 byte mà mỗi lần server gửi tin về cho client. Để hiển thị ra các thông điệp của server gửi cho client – khi đó ta cần gọi hàm `s.recv(1024).decode()` để giải mã và thực hiện hiển thị các thông điệp mà server gửi về client.

Bước cuối cùng, khi việc trao đổi thông tin hoàn thành ta sẽ đóng kết nối lại bằng hàm `s.close()`.



```
Client.py X
Client.py > ...
1 import socket
2 # Tao doi tuong socket cho client
3 s = socket.socket()
4 #Lay ra ten host cua server (localhost)
5 host = socket.gethostname() #host = "localhost"
6 # Lay ra port cua server
7 port = 8080
8 # Thuc hien ket noi tu client toi server
9 s.connect((host, port))
10 # Nhan lai tin nhan duoc gui tu server toi client
11 msg = s.recv(1024) # 1024 la so byte ma client co the nhan duoc
12 #tin nhan cua server gui ve trong mot lan
13 # Duyet va hien thi cac phan tin nhan tiep theo
14 while msg:
15     print("Ban tin nhan tu server: ", msg.decode())
16     msg = s.recv(1024)
17 # Dong ket noi toi server
18 s.close()
```

Hình 2.3. Demo chương trình Máy khách (Client)

2.3.3. Thực thi Demo chương trình Client – Server trong Python

Đầu tiên chạy File Server.py bằng câu lệnh trong command line (CMD): `python Server.py`

Tiếp theo, sẽ thực thi file client.py bằng câu lệnh như sau trong command line (CMD) và nếu như nhận được bản tin được gửi từ server về client thì đồng nghĩa với việc kết nối client – server đã thành công.



```
PS D:\project\kltn\Code\Demo Socket> python .\Client.py
Ban tin nhan tu server: Day la chuong trinh demo Socket Server
```

Hình 2.4. Kết quả Demo chương trình Client – Server trong Python

2.4. Khái niệm hệ quản trị cơ sở dữ liệu SQLite

Đầu tiên ta phải hiểu **cơ sở dữ liệu (Database)** là gì? **Cơ sở dữ liệu** là một bộ sưu tập dữ liệu được tổ chức **bày bản** và thường được truy cập từ hệ thống máy tính hoặc tồn tại dưới dạng tập tin trong hệ quản trị cơ sở dữ liệu. Database còn có thể được lưu trữ trên thiết bị có chức năng ghi nhớ như: thẻ nhớ, đĩa cứng, CD...

Vì thế Hệ quản trị cơ sở dữ liệu (Database Management System) là một phần mềm cho phép tạo lập các CSDL cho các ứng dụng khác nhau và điều khiển mọi truy cập tới các CSDL đó. Nghĩa là, hệ quản trị CSDL cho phép định nghĩa (xác định kiểu, cấu trúc, ràng buộc dữ liệu), tạo lập (lưu trữ dữ liệu trên các thiết bị nhớ) và thao tác (truy vấn, cập nhật, kết xuất, ...) các CSDL cho các ứng dụng khác nhau. Ví dụ: MS. Access, MS. SQL Server, ORACLE, IBM DB2,...

Ở đây ta sẽ sử dụng SQLite để xây dựng chương trình ứng dụng. Đặc điểm nổi bật của SQLite so với các DBMS khác là gọn, nhẹ, đơn giản, đặt biệt không cần mô hình server-client, không cần cài đặt, cấu hình hay khởi động nên không có khái niệm user, password hay quyền hạn trong SQLite Database. Dữ liệu cũng được lưu ở một file duy nhất.

Các lệnh SQLite chuẩn để tương tác với Cơ sở dữ liệu quan hệ là giống như SQL. Chúng là CREATE, SELECT, INSERT, UPDATE, DELETE và DROP.

Lệnh	Miêu tả
CREAT	Tạo một bảng mới, một View của một bảng hawocj đối tượng khác trong Database
ALTER	Sửa đổi một đối tượng cơ sở dữ liệu đang tồn tại, ví dụ một bảng
DROP	Xóa cả một bảng, một View của một bảng hoặc đối tượng khác trong Database
INSERT	Tạo một bản ghi
UPDATE	Sửa đổi các bản ghi
DELETE	Xóa các bản ghi
SELECT	Lấy các bản ghi cụ thể từ một hoặc nhiều bảng

Bảng 2.2. Một số lệnh sơ bản của SQLite

Một số **ưu điểm** của SQLite:

- ☐ SQLite không cần phải cấu hình tức là bạn không cần phải cài đặt.
- ☐ Với SQLite database được lưu trữ trên một tập tin duy nhất.
- ☐ SQLite hỗ trợ hầu hết các tính năng của ngôn ngữ truy vấn SQL theo chuẩn SQL92.
- ☐ SQLite rất nhỏ gọn bản đầy đủ các tính năng nhỏ hơn 500kb, và có thể nhỏ hơn nếu lược bớt một số tính năng.
- ☐ Các thao tác dữ liệu trên SQLite chạy nhanh hơn so với các hệ quản trị cơ sở dữ liệu theo mô hình client – server.
- ☐ SQLite rất đơn giản và dễ dàng sử dụng.
- ☐ SQLite tuân thủ 4 tính chất ACID (là tính nguyên tử (Atomic), tính nhất quán (Consistent), tính cô lập (Isolated), và tính bền vững (Durable)).

- Với đặc tính nhỏ gọn, truy xuất dữ liệu nhanh SQLite thường được sử dụng để nhúng vào các dự án

Một số **nhược điểm** của SQLite:

- Một số tính năng của SQL92 không được hỗ trợ trong SQLite như ALTER DROP COLUMN, ADD CONSTRAINT không được hỗ trợ; RIGHT JOIN; TRIGGER; phân quyền GRANT và REVOKE.
- Vì SQLite không cần cấu hình, cài đặt, không hỗ trợ GRANT và REVOKE nên việc phân quyền truy cập cơ sở dữ liệu chỉ có thể là quyền truy cập file của hệ thống.
- SQLite sử dụng cơ chế coarse-grained locking nên trong cùng một thời điểm có thể hỗ trợ nhiều người đọc dữ liệu, nhưng chỉ có 1 người có thể ghi dữ liệu.
- SQLite không phù hợp với các hệ thống có nhu cầu xử lý trên một khối lượng dữ liệu lớn, phát sinh liên tục.

2.5. SQLite trong Python

Python có một thư viện để truy cập đến các cơ sở dữ liệu SQLite, được gọi là `sqlite3`, cung cấp khả năng làm việc với cơ sở dữ liệu này. Thư viện `sqlite3` đã được tích hợp vào trong gói Python kể từ phiên bản 2.5.

Sau đây ta sẽ tìm hiểu một số cách thao tác cơ bản với SQLite trong Python.

2.5.1. Kết nối tới cơ sở dữ liệu

Để có thể tương tác với cơ sở dữ liệu SQLite trong Python, việc đầu tiên chúng ta cần làm là thực hiện thêm mô-đun `sqlite3` đã được tích hợp sẵn trong Python. Câu lệnh thực hiện như sau: `import sqlite3`

Sau khi thực hiện nhập mô-đun `sqlite3`. Bây giờ, ta sẽ thực hiện kết nối với cơ sở dữ liệu bằng phương thức `connect()` được cung cấp bởi `sqlite3`. Câu lệnh này sẽ trả về một đối tượng `Connection`. Sau đó, ta cần cung cấp đường dẫn của cơ sở dữ liệu đến phương thức `connect()`. Ta sẽ đặt tên của cơ sở dữ liệu là `demo` và thêm vào cuối với phần đuôi mở rộng là `db`. Như vậy, câu lệnh hoàn chỉnh để thực hiện sẽ như sau:

```
D: > project > kltm > code > SQLite > demoSQLite.py > ...
1  import sqlite3
2  con = sqlite3.connect('demo.db')
3
```

Hình 2.5. Phương thức `connect()` trong SQLite

Câu lệnh này sẽ thực hiện tạo một tệp với tên là demo.db.

2.5.2. SQLite Cursor

Để thực thi các câu lệnh SQLite trong Python, ta cần một đối tượng **cursor**. Ta có thể tạo nó bằng phương thức `cursor()`.

Cursor trong SQLite3 là một phương thức của đối tượng Connection. Để thực thi các câu lệnh SQLite3, trước tiên ta cần thiết lập một kết nối và sau đó tạo một đối tượng **Cursor** bằng cách sử dụng đối tượng Connection như sau:

```
D: > project > kltm > code > SQLite > demoSQLite.py > ...
1 import sqlite3
2 con = sqlite3.connect('demo.db')
3 cursorObj = con.cursor()
```

Hình 2.6. SQLite Cursor

Như vậy, bây giờ ta đã có thể sử dụng đối tượng **cursor** và thực hiện gọi phương thức `execute()` để thực thi các câu lệnh truy vấn SQL.

2.5.3. Tạo cơ sở dữ liệu

Khi tạo kết nối với cơ sở dữ liệu SQLite, kết nối đó sẽ tự động tạo tệp cơ sở dữ liệu nếu nó chưa tồn tại. Tệp cơ sở dữ liệu này được tạo trên đĩa. Ta cũng có thể tạo cơ sở dữ liệu trong RAM bằng cách sử dụng `:memory:` với hàm `connect()`.

```
D: > project > kltm > code > SQLite > demoSQLite.py > ...
1 import sqlite3
2 from sqlite3 import Error
3 def sql_connection():
4     try:
5         con = sqlite3.connect(':memory:')
6         print("Connection is established: Database is created in memory")
7     except Error:
8         print(Error)
9     finally:
10        con.close()
11
12 sql_connection()
```

Hình 2.7. Ví dụ về tạo cơ sở dữ liệu

Trong đoạn mã bên trên, đầu tiên, ta thực hiện nhập mô-đun `sqlite3`, sau đó thực hiện định nghĩa một hàm có tên là `sql_connection`. Bên trong **hàm** này, ta có một khối

try, trong đó chứa hàm *connect()* trả về một đối tượng Connection sau khi thiết lập kết nối.

Sau đó, ta tạo khối *except*, trong trường hợp có bất kỳ ngoại lệ nào, khối lệnh bên trong sẽ in ra thông báo lỗi. Nếu không có bất kỳ lỗi nào, kết nối sẽ được thiết lập và sẽ hiển thị thông báo.

Sau đó, ta thực hiện **đóng** kết nối trong khối *finally*. Việc đóng kết nối là tùy chọn, nhưng nên thực hiện **đóng** kết nối với cơ sở dữ liệu sau khi thao tác xong bởi nó giúp giải phóng bộ nhớ khỏi tài nguyên mà không sử dụng tới.

2.5.4. Tạo bảng trong cơ sở dữ liệu

Để tạo bảng trong SQLite3, ta có thể sử dụng câu truy vấn tạo bảng và đưa vào trong phương thức *execute()*. Các bước thực hiện như sau:

- ☐ Tạo một đối tượng Connection.
- ☐ Tạo một đối tượng **Cursor**.
- ☐ Bằng cách sử dụng đối tượng **Cursor**, ta thực hiện gọi phương thức *execute()* với câu truy vấn tạo bảng làm tham số.

Ta sẽ thực hiện tạo một bảng sinh viên (Student) với các thuộc tính như sau: *Student (Id, Name, Address, DateOfBirth)*

Đoạn mã thực thi như sau:

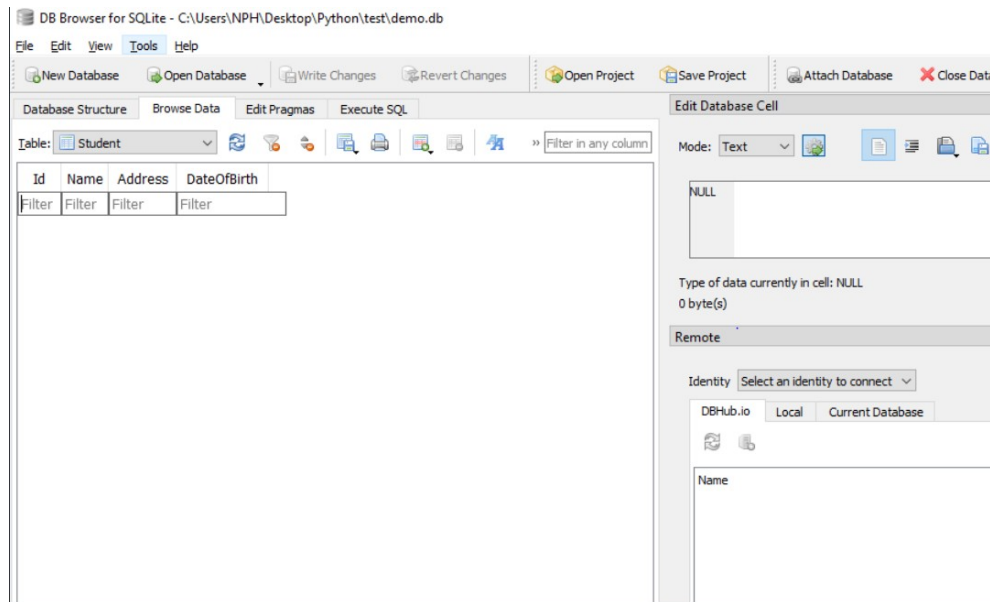
```
D: > project > kltm > code > SQLite > demoSQLite.py > ...
1 import sqlite3
2 from sqlite3 import Error
3
4 def sql_connection():
5     try:
6         con = sqlite3.connect('demo.db')
7         return con
8     except Error:
9         print(Error)
10
11 def sql_table(con):
12     cursorObj = con.cursor()
13     cursorObj.execute("CREATE TABLE Student(Id, Name, Address,
14                                     DateOfBirth)")
15     con.commit()
16
17 con = sql_connection()
18 sql_table(con)
```

Hình 2.8. Ví dụ tạo bảng trong cơ sở dữ liệu

Trong đoạn mã bên trên, ta đã định nghĩa hai phương thức, phương thức thứ nhất có nhiệm vụ thực hiện thiết lập kết nối và phương thức thứ hai có nhiệm vụ thực hiện tạo đối tượng **Cursor** để thực hiện câu lệnh truy vấn tạo bảng **vào** trong cơ sở dữ liệu.

Phương thức *commit()* thực hiện lưu lại tất cả các thay đổi mà ta đã thực hiện. Cuối cùng, cả hai phương thức này đều được gọi để thực hiện tác vụ như mong muốn.

Mở tệp demo.db bằng cách sử dụng trình duyệt DB cho SQLite ta có kết quả như sau:



Hình 2.9. Kết quả tạo bảng cơ sở dữ liệu.

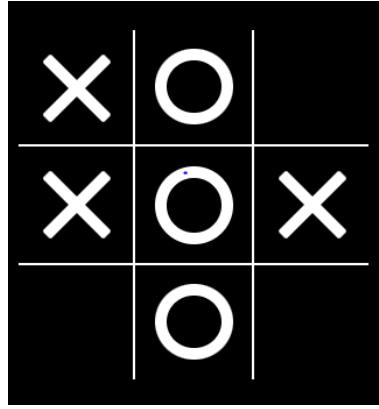
CHƯƠNG 3: XÂY DỰNG CHƯƠNG TRÌNH ỨNG DỤNG

3.1. Giới thiệu về trò chơi cờ caro

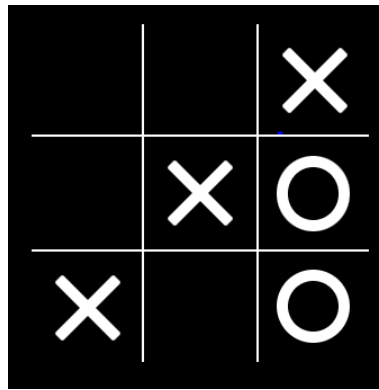
Tic Tac Toe là một trò chơi khá phổ biến dùng viết trên bàn cờ giấy có 9 ô. Hai người chơi, một người dùng ký hiệu O, người kia dùng ký hiệu X, lần lượt điền ký hiệu của mình vào các ô. Người thắng cuộc là người đầu tiên tạo được một dãy 3 ký hiệu của mình theo các chiều ngang, dọc hay chéo đều được. Nếu sau khi đã lấp đầy các ô trống mà vẫn không có ai đạt được một dãy 3 ô thẳng hàng thì sẽ là hòa.

Tính thân thiện của trò chơi khiến trò chơi này đã trở thành một trò chơi dạy học trong các trường học để dạy về tinh thần thể thao và trí tuệ. Người Việt thường chơi trò tương tự, gọi là Cờ ca-rô, bàn cờ không giới hạn trong 9 ô, có thể vẽ thêm ô, để kéo rộng ra cho đến khi người nào đạt được một dãy 5 thì thắng cuộc.

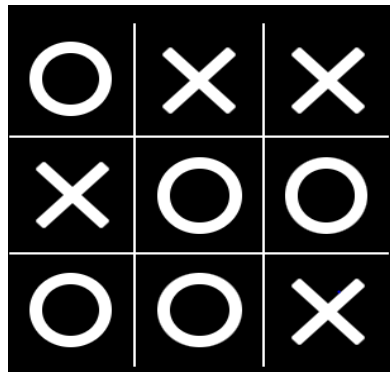
Hình sau mô tả 3 trường hợp ví dụ thắng – thua – hòa trong cờ caro:



Hình 3.1. Ví dụ người chơi O thắng



Hình 3.2. Ví dụ người chơi X thắng



Hình 3.3. Ví dụ cờ hòa

3.2. Phân tích và thiết kế

3.2.1. Mục tiêu của chương trình

Mục tiêu của chương trình là phát triển trò chơi còwd caro sử dụng Socket để **ch** phép người chơi (Client) chơi với người chơi khác trên các máy tính khác nhau (hoặc giống nhau) trên cùng một mạng máy tính.

Máy chủ (Server) sẽ **co** thêm chức năng xác thực, kết nối được với cơ sở dữ liệu (Database) và kết nối hai người chơi với nhau trong trò chơi. Điều này hoàn toàn có thể làm được bằng cách sử dụng Socket trong Python.

Trong cơ sở dữ liệu sẽ xác định một bảng dữ liệu bao gồm thông tin người chơi như số trận thắng, số trận thua, số trận đánh, tên tài khoản, mật khẩu. Vì thế, người chơi cũng có thể xem bảng xếp hạng gồm thứ hạng các người chơi được sắp xếp theo thứ tự **thứ** cao đến thấp dựa vào số trận thắng.

Để có đăng nhập vào trò chơi, người chơi tạo một tài khoản và mật khẩu. Dữ liệu này sẽ được lưu vào Database. Khi đăng nhập, Máy chủ sẽ so sánh tên đăng nhập và mật khẩu mà người chơi nhập vào với dữ liệu đã lưu trong Database. Đặc biệt để có thể kết nối với Máy chủ, người cần nhập đúng thông tin của Máy chủ bao gồm tên máy chủ và **cổng** máy tính.

3.2.2. Yêu cầu của chương trình

Để hoàn thành chương trình này, ta cần thực hiện những việc sau:

- ☐ Thiết kế giao diện cho người dùng với các giao diện khác nhau **bào** gồm: giao diện xác thực máy khách, trang chủ, giao diện game và giao diện bảng điểm. Để tạo giao diện ta sẽ sử dụng Tkinter trong Python.
- ☐ Nghiên cứu và ứng dụng và sử dụng Socket trong Python để kết nối Máy khách(Client) với Máy chủ (Server). Máy chủ phải có chức năng xử lý và gửi về mọi hành động của Máy khách và ngược lại.
- ☐ Phát **triển** hệ thống đăng kí, đăng nhập bằng cách sử dụng hệ quản trị cơ sở dữ liệu SQLite.
- ☐ Lập trình logic game cơ Caro hoàn thiện có 3 **trang** thái kết thúc trò chơi là thắng, thua và hòa
- ☐ Sau khi hoàn **thanh** trò chơi , Máy chủ sẽ gửi kết quả về cơ sở dữ liệu để có thể cập nhật bảng xếp **hạng** người chơi.
- ☐ Đánh giá, nhận xét về chương trình.

3.2.3. Thiết kế chương trình

a. Thiết kế cơ sở dữ liệu

Như đã nói ở trên, cơ sở dữ liệu sẽ sử dụng là SQLite vì tính tiện dụng của nó và sẽ dùng câu lệnh SQL để tương tác.

- ☐ Khởi tạo cơ sở dữ liệu:

```
CREATE TABLE IF NOT EXISTS users (
    username VARCHAR (25) NOT NULL UNIQUE,
    password VARCHAR (500) NOT NULL,
    wins INT UNSIGNED DEFAULT 0,
    loses INT UNSIGNED DEFAULT 0,
    games_played int UNSIGNED DEFAULT 0
);
```

- ☐ Từ khóa “IF NOT EXIT” sẽ không cho phép máy chủ tạo bảng thứ hai khi chương trình được khởi động lại
- ☐ Các trường như “win”, “loses” và ”game played” là các số nguyên dương. Điều này giúp giảm bộ nhớ phải sử dụng đi.

- ☐ Kiểm tra thông tin người dùng trong cơ sở dữ liệu:

“:username” là biến thể hiện cho người dùng

```
SELECT username FROM users WHERE username = :username;
```

- ☐ Tạo tài khoản người **dung** trong cơ sở dữ liệu:

“:username” là biến thể hiện cho người dùng

```
INSERT INTO users (username, password) VALUES (?, ?);
```

- ☐ Cập nhật dữ liệu người **dung** sau khi chơi trò chơi

“:username” là biến thể hiện cho người dùng

If the player won:

```
UPDATE users SET wins = wins+1, games_played=games_played+1
WHERE username = :username
```

Else:

```
UPDATE users SET loses=loses+1, games_played=games_played+1
WHERE username = :username
```

b. Thiết **kết** giao diện chương trình

Giao diện chương trình sẽ được tạo bởi Python Tkinter. Mục đích của giao diện:

- ☐ Nút ấn rõ ràng, dễ phân biệt
- ☐ Thể hiện thông tin người dùng
- ☐ Màn hình Đăng nhập /Đăng ký

Phần đăng nhập phải bao gồm cả tên tài khoản và mật khẩu để người dùng có thể đăng nhập. Về phần thông tin máy chủ phải cung cấp cho người dùng tên máy chủ (dạng chuỗi kí tự) và cổng (dạng số nguyên). Máy khách không thể đăng nhập máy chủ khi nhập sai thông tin máy chủ hoặc khi gặp sự cố xác thực đăng nhập /đăng ký. Sẽ có một thông báo lỗi hiển thị *“Can’t reach the socket server”*.

Nếu máy khách muốn đăng ký một tài khoản, sẽ có *action* [USER LOGIN] được gửi đến máy chủ. Nếu tên tài khoản đã được đăng ký, máy khách sẽ được thông báo rằng tên người dùng đã được sử dụng **trong** cơ sở dữ liệu. Nếu tên tài khoản chưa có trong cơ **sở** dữ liệu, tài khoản sẽ được tạo.

Khi máy khách đăng nhập sẽ có một *action* [USER LOGIN] được gửi đến máy chủ. Nếu tên tài khoản được tìm thấy cơ sở dữ liệu hoặc mật khẩu không tương ứng cũng có một thông báo rằng tài khoản hoặc mật khẩu không chính xác.

- ☐ Màn hình trang chủ

Màn hình này chỉ hiển thị khi máy **khách** đã đăng nhập được. Sẽ có một thông báo chào mừng ở trên cùng với 2 nút *“join game”* và *“view leader-board”*.

- ☐ Màn hình đợi trò chơi

Màn hình này xuất hiện khi máy khách ấn nút *“join game”* với thông báo đang đợi người chơi khác (*“Waiting for another player to join”*) và một nút ẩn để máy **khách** có thể thoát khỏi **hàng** chờ trên máy chủ.

- ☐ Màn hình trò chơi

Sẽ có một bảng trò chơi 3 3 cho phép người **chơi** hiện tại thực hiện lượt chơi bằng cách chọn ô vuông mà mình muốn chọn. Nếu không phải lượt của mình, việc chọn **ô** sẽ không có tác dụng gì. Ở **bên** dưới bảng sẽ có thông tin người chơi và lượt chơi hiện tại. Khi trò chơi kết thúc, sẽ xuất hiện nút ẩn để quay lại trang chủ.

- ☐ Màn hình bảng xếp hạng người chơi

Một cột là danh sách các tài khoản của máy **khách** với số trận thắng được sắp xếp giảm dần. Bên phải màn hình sẽ có một nút để cập nhật dữ liệu và tìm kiếm thứ hạng của người dung. Cuối cùng sẽ có một nút để quay lại trang chủ.

3.3. Cài đặt và kết quả đạt được

□ Cài đặt Máy chủ (Server)

Sử dụng lệnh CMD *python server.py* để chạy file Server.py. Chương **tình** sẽ được chạy với phiên bản Python 3.7 và ta có kết quả:

```
PS D:\project\unity\New folder\kltn> python .\server.py
server started on Host: LAPTOP-S3223PHU , Port: 4201
```

Hình 3.4. Kết quả cài **đặt** máy chủ

Chương trình máy chủ đang được chạy trên một chiếc máy tính xách tay, vì thế **tên** máy chủ là LAPTOP-S3223PHU với cổng 4201.

□ Cài đặt máy khách (Client)

Tương tự như máy chủ ta cũng chạy câu lệnh CMD với câu lệnh: *python client.py*. Ta sẽ được giao diện đăng nhập/ đăng ký.

The screenshot shows a web application interface with two main sections: 'Authentication' and 'Server Info'. The 'Authentication' section contains two input fields: 'Username:' and 'Password:'. The 'Server Info' section contains two input fields: 'Host name:' and 'Port number:'. The 'Port number' field has the value '4201' entered. Below these sections are two buttons: 'Login' and 'Register'.

Hình 3.5. Giao diện đăng nhập/ đăng ký

Giao diện thông báo khi đăng nhập mà không điền tài khoản, mật khẩu: *tài khoản phải có một hoặc nhiều **kí** tự*

The image shows a web form with two main sections: 'Authentication' and 'Server Info'. The 'Authentication' section contains two input fields: 'Username:' and 'Password:'. The 'Server Info' section contains two input fields: 'Host name:' and 'Port number:'. Below these sections are two buttons: 'Login' and 'Register'. At the bottom, there is a small text message: 'Username must be 1 character or more'.

Authentication	
Username:	<input type="text"/>
Password:	<input type="password"/>

Server Info	
Host name:	<input type="text"/>
Port number:	<input type="text" value="4201"/>

Username must be 1 character or more

Hình 3.6. Giao diện đăng nhập, đăng ký khi không điền tài khoản, mật khẩu

Tiếp theo là giao diện màn hình đăng nhập, đăng ký khi đăng ký tài khoản với mật khẩu ít hơn 6 ký tự. Màn hình sẽ hiện dòng thông báo yêu cầu mật khẩu phải có ít nhất 6 ký tự.

Authentication

Username:

Password:

Server Info

Host name:

Port number:

Login **Register**

Password must be 6 characters or more

Hình 3.7. Giao diện màn hình đăng nhập, đăng ký khi nhập mật khẩu ít hơn 6 ký tự

Giao diện màn hình đăng nhập, đăng ký khi điền tên người **dung** và mật khẩu không giống trong cơ sở dữ liệu:

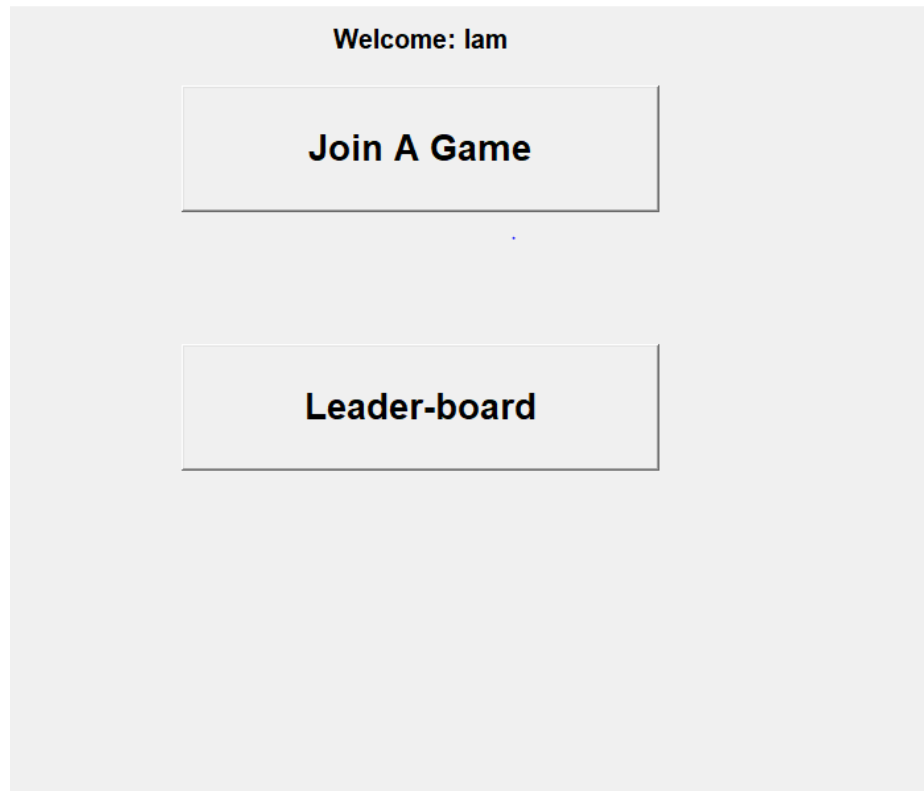
The screenshot shows a web application interface with two main sections: 'Authentication' and 'Server Info'. The 'Authentication' section contains two input fields: 'Username' with the value 'lam' and 'Password' with masked characters '*****'. Below these fields are two buttons: 'Login' and 'Register'. The 'Server Info' section contains two input fields: 'Host name' with the value 'LAPTOP-S3223PHU' and 'Port number' with the value '4201'. At the bottom of the interface, a message states: 'No user found with the username: lam'.

Hình 3.8. Giao diện màn hình đăng nhập, đăng ký khi điền sai tài khoản, mật khẩu
Bên Máy chủ (Server) sẽ so sánh và thông báo kết nối lỗi:

```
Attempt to authenticate client lam
[USER LOGIN - FAIL]
Clnet authenticated: Fail
```

Hình 3.9. Server khi đăng nhập sai tên tài khoản, mật khẩu

Khi máy khách đăng nhập tên người dùng và mật khẩu chính xác, thông báo đăng nhập sẽ được hiện lên và chuyển sang màn hình trang chủ:



Hình 3.10. Giao diện màn hình khi đăng nhập thành công

Khi nhấn vào nút “Leader-board”, Sẽ xuất hiện trang bảng xếp hạng. Bảng này sẽ bao gồm 7 thông tin người chơi được tạo tự động trong cơ sở dữ liệu và thông tin người dùng mà các máy khách đã đăng kí thành công.

Ranking	Username	Wins	Losses	Games Played
1	test3	1	0	1
2	test4	1	0	1
3	test5	1	0	1
4	lam1	1	0	1
5	test1	0	1	1
6	test2	0	1	1
7	test6	0	1	1
8	lam2	0	1	1
9	lam	0	0	0

Update Board

Username: lam

Search Player Rank

Get My Position

Go Back to home

Hình 3.11. Giao diện bảng xếp hạng người chơi

Ta có thể tìm kiếm thứ hạng của mình hoặc của người dùng khác bằng cách nhập tên người dùng vào ô *Username*.

Ranking	Username	Wins	Losses	Games Played
1	test3	1	0	1
2	test4	1	0	1
3	test5	1	0	1
4	lam1	1	0	1
5	test1	0	1	1
6	test2	0	1	1
7	test6	0	1	1
8	lam2	0	1	1
9	lam	0	0	0

Update Board

Username: lam

Search Player Rank

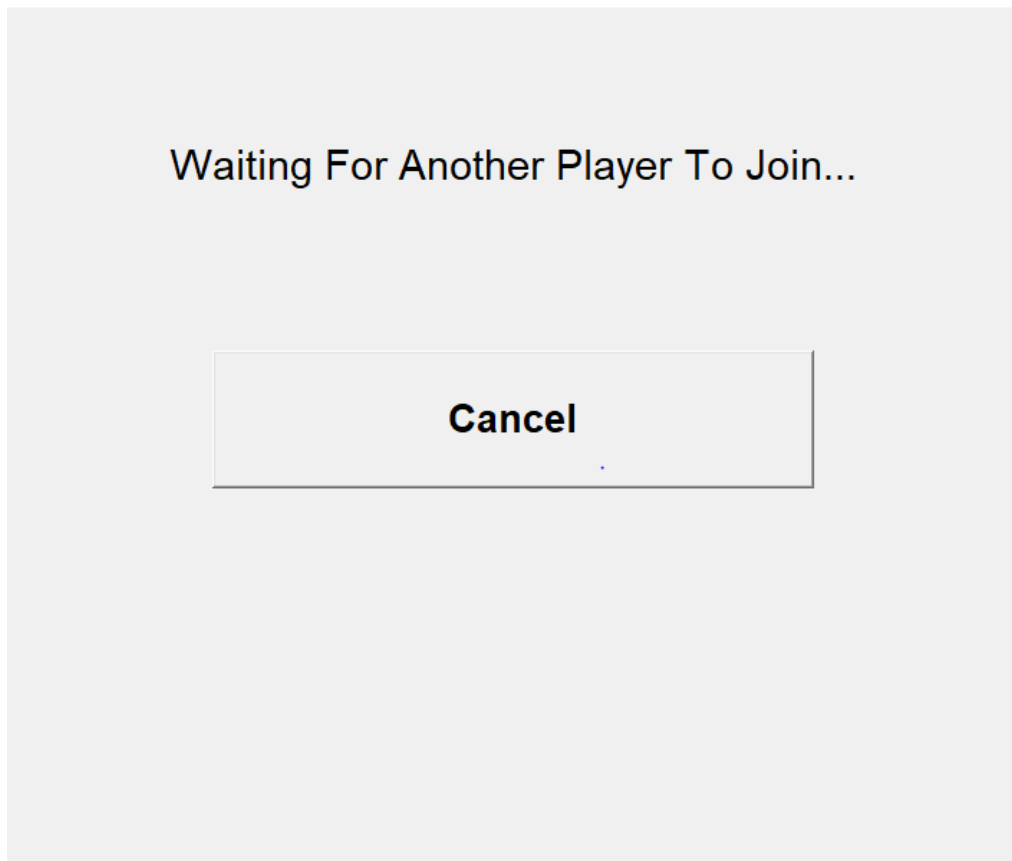
Get My Position

Go Back to home

Rank of lam: 9

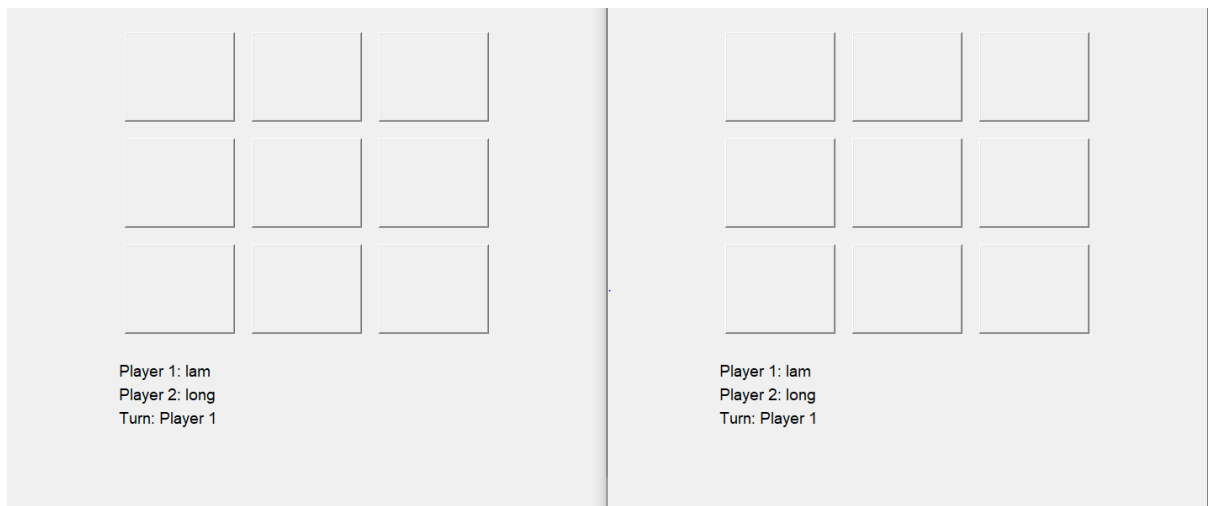
Hình 3.12. Giao diện khi tìm kiếm thứ hạng người chơi

Để có thể tham gia trò chơi, ta nhấn vào nút *Join A Game*. Màn hình đợi trò chơi sẽ xuất hiện.



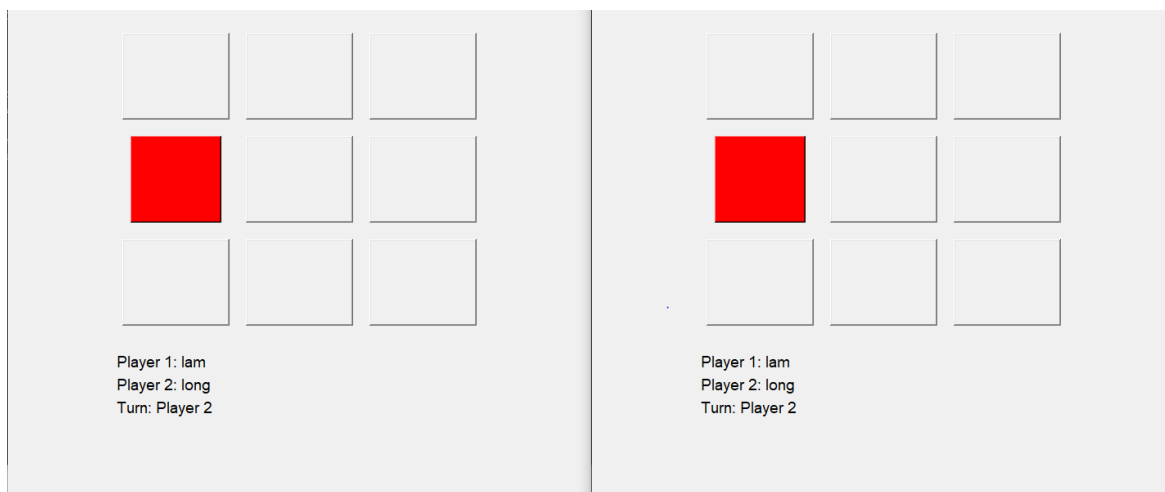
Hình 3.13. Giao diện đợi trò chơi

Khi hai người chơi đều trong **hang** đợi, yêu cầu tạo dữ liệu trò chơi sẽ được gửi đến máy chủ và thông báo với cả 2 người **chờ** rằng trò chơi đã được bắt đầu.



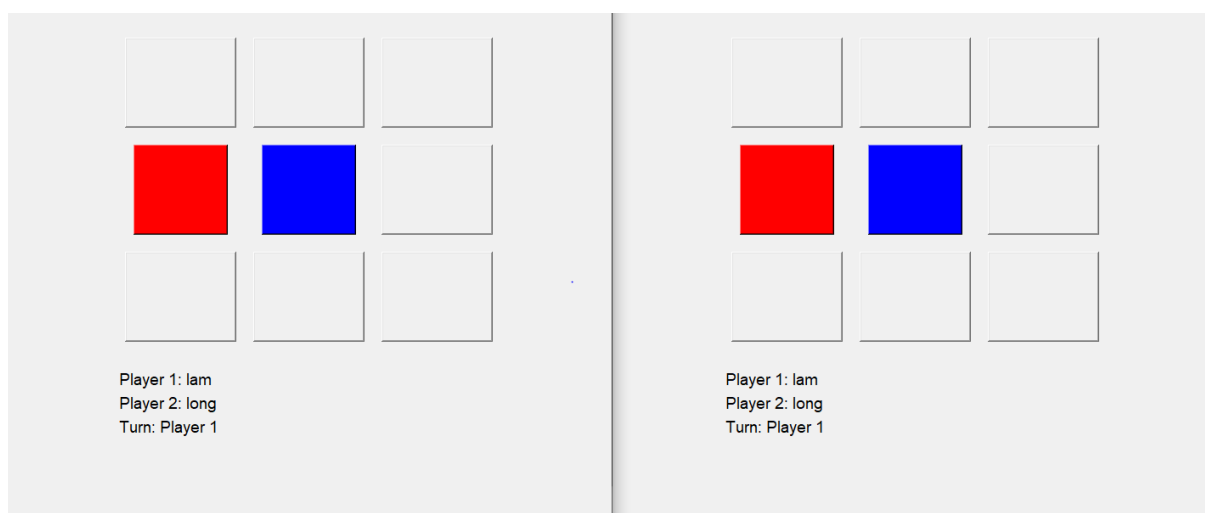
Hình 3.14. Giao diện khi vào trò chơi

Giao diện khi người chơi 1 chơi:



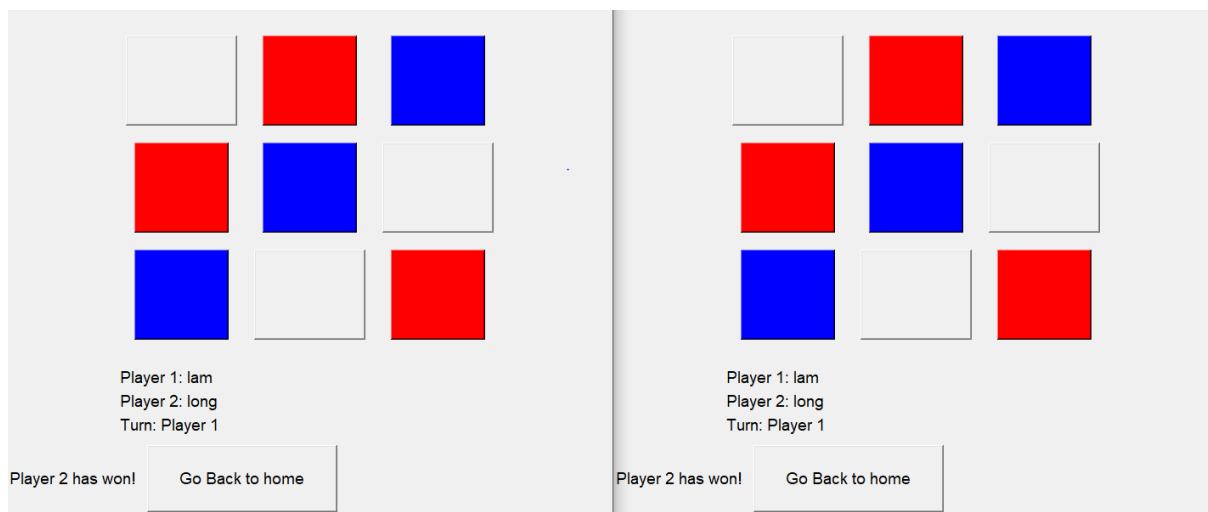
Hình 3.15. Giao diện khi người chơi 1 chơi

Giao diện khi người chơi 2 chơi:



Hình 3.16. Giao diện khi người chơi 2 chơi

Giao diện khi người chơi 2 thắng, người chơi 1 thua:



Hình 3.17. Giao diện khi người chơi 2 thắng

Sau khi người chơi 2 thắng với Username là “long”, thông tin người chơi sẽ được cập nhật lại trong bảng xếp hạng người chơi.

Ranking	Username	Wins	Losses	Games Played
1	test3	1	0	1
2	test4	1	0	1
3	test5	1	0	1
4	lam1	1	0	1
5	long	1	0	1
6	test1	0	1	1
7	test2	0	1	1
8	test6	0	1	1
9	lam2	0	1	1
10	lam	0	1	1

Update Board

Username:

Search Player Rank Get My Position

Go Back to home

Hình 3.17. Giao diện bảng xếp hạng người chơi sau khi trò chơi kết thúc

Người chơi với Username đã được cập số trận thắng tại cột “Wins” còn người chơi với Username đã được cập nhật số trận thua tại cột “Losses”. Cả 2 người chơi đều được cập nhật thêm số trận đã chơi tại cột “Games Played”.

3.4. Nhận xét

Chương trình game Cờ Caro ứng dụng Socket trong Python đã được khởi chạy và kiểm thử thành công. Ứng dụng này đã đáp ứng được các yêu cầu ban đầu thiết kế đề ra:

- ☐ Người dùng dễ dàng tương tác với giao diện chương trình
- ☐ Người dùng có thể đăng kí tài khoản và mật khẩu xác thực giúp bảo mật dữ liệu của người dùng và cơ sở dữ liệu
- ☐ Mật khẩu đã được mã hóa trong cơ sở dữ liệu
- ☐ Máy chủ (Server) và Máy khách (Client) giao tiếp truyền thông ổn định
- ☐ Trò chơi Cờ Caro hoạt động tốt, bảng xếp hạng hiển thị đầy đủ

Về phần code, các hàm đã được chú thích đầy đủ. Code được viết theo hướng đối tượng và **mo-dul**. Điều này giúp bảo trì và thay đổi code dễ dàng và thuận tiện hơn.

Chương trình là một ứng dụng đơn giản những gì đã học và tìm hiểu kiến thức về mạng máy tính và ngôn ngữ Python. Sắp tới em sẽ cập nhật và cải tiến thêm nhiều chức năng khác và sửa các lỗi còn tồn đọng trong chương trình này

KẾT LUẬN

Qua quá trình tìm hiểu và thực hiện đề tài khóa luận này, em đã hiểu thêm về ứng dụng lập trình mạng và các thuật toán để giải quyết các vấn đề trong thực tế. Ứng dụng game là một cách rất **tốt** để có thể nâng cao kiến thức về mạng **mạng** tính và kỹ năng lập trình. Tuy nhiên do hạn chế về thời gian và trình độ bản thân nên em chưa thể hoàn thiện được tất cả các chức năng cũng như thuật toán, trong quá trình thực hiện cũng không thể tránh những sai sót. Em rất mong nhận được sự đóng góp của thầy cô để ứng dụng ngày càng được hoàn thiện.

Các kết quả đạt được:

- ☐ Tìm hiểu kiến thức về lập trình mạng, mô hình Máy khách (Client) – Máy chủ (Server), lý thuyết về Socket trong Python, kỹ thuật lập trình giao diện cơ bản trong Python.
- ☐ Chương trình ứng dụng được xây dựng đầy đủ chức năng cơ bản, Máy chủ và máy khách giao tiếp truyền thông ổn định.
- ☐ Đề tài “Xây dựng ứng dụng game đa người dùng sử dụng socket bằng ngôn ngữ python” được hoàn **thiện** đúng yêu cầu và thời gian quy định.

Các kết quả chưa đạt được:

- ☐ Giao diện trò chơi vẫn chưa được **đẹp** mắt.
- ☐ Một số chức năng vẫn chưa được hoàn thiện, chưa xử lý được.
- ☐ Số lượng người dùng trong cơ sở dữ liệu vẫn bị giới hạn.

PHỤ LỤC

1. Mã nguồn bên phía Máy chủ (Sever)

1.1. File server.py

```
from server_sql_connection import SqlServerConnection
import socket # used to run the socket server
import select # used to manage cuncurent connections to the server socket
import pickle
import uuid
import hashlib

class SocketServer(socket.socket):
    def __init__(self):
        super().__init__(socket.AF_INET, socket.SOCK_STREAM)

        # gets the current host, internal network. replace with ("", PORT) for external
        network

        socketHostData = (socket.gethostname(), 4201)

        # binds the socket server to the current host name and listens to active
        connections

        self.bind(socketHostData)

        print(
            f"server started on Host: {socketHostData[0]} , Port: {socketHostData[1]}")

        # set max connection to 6

        self.listen(6)

        self.sockets_list = [self]

        # this SET will keep track of which client/s what is waiting to join a game

        self.waiting_queue = set()

        # this will keep track of all on going games

        self.onGoingGames = {}

        # this will store all connected users

        self.clients = {}
```



```

# define the size of the length of the header
self.HEADERSIZE = 10

# connect to the database
self.DB = SqlServerConnection()

# Actions that authenticated users can call
self.actions = {
    "[JOIN GAME]": self.joinGame,
    "[CANCEL GAME]": self.cancelGame,
    "[TAKE TURN]": self.takeTurn,
    "[GET ALL PLAYER STATS]": self.getAllPlayerStats
}

self._action_handler()
def _action_handler(self):
    while True:
        # defines our read, write and errored listed sockets
        read_sockets, _write, error_sockets = select.select(
            self.sockets_list, [], self.sockets_list)
        for user_socket in read_sockets:
            if user_socket == self:
                """
                These will be the client sockets trying to connect to the server socket.
                So in this block i will be authenticating client sockets.
                """

                # accept connections from server, this is so i can read the package
                client_socket, client_address = self.accept()
                print( client_socket, client_address)
                # handle the incomming document action
                client_action_document = self.recv_doc_manager(

```

```

        client_socket)
    if client_action_document is False:
        # disconnection, left the sockect
        continue

    # HANDLE UNAUTHENTICATED USERS
    if client_action_document["action"] == '[USER LOGIN]': # Login
attempt
        """
        authenticate the user, this should return the user data in the database
        if the username and password is correct.
        """
        user = self.login_manager(
            client_action_document["data"]) # get user data from database
        if user["result"] is False:
            # client failed to authenticate
            # user[data] is the error message
            client_socket.send(self.pkg_doc_manager(
                "[USER LOGIN - FAIL]", user["data"]))
            continue
        # login user successful
        # user[data] is the users account data
        client_socket.send(self.pkg_doc_manager(
            "[USER LOGIN - SUCCESS]", user["data"]))
        self.sockets_list.append(client_socket) # Accept future request from
this socket.
        self.clients[client_socket] = user["data"] # keep track of who is logged
in.
        print(
            f'Accepted new connection from {client_address[0]}:
{client_address[1]}, \

```

```

        action_type: {client_action_document["action"]}, Username:
{user["data"][0]})
        continue
    if client_action_document["action"] == '[USER REGISTER]': # Register
attempt
        # register the user in the db
        create_account_status = self.registration_manager(
            client_action_document["data"])
        if create_account_status["result"] is False:
            # something went wrong whilst creating the user account on the
database
            client_socket.send(self.pkg_doc_manager(
                "[USER REGISTER - FAIL]", create_account_status["msg"]))
            continue
            # tell the client that they have successfully created an account tn the
database
            client_socket.send(self.pkg_doc_manager(
                "[USER REGISTER - SUCCESS]", create_account_status["msg"]))
            print('Created new account from {}: {}, action_type: {}'.format(
                *client_address, client_action_document["action"]))
            continue
    else:
        """
        At this point the connected client socket has already been authenticated
        """
        client_action_document = self.recv_doc_manager(
            user_socket)
        if client_action_document is False:
            # disconnection, left the sockect
            print(fClosed connection from User: {self.clients[user_socket][0]})

```

```

        self.sockets_list.remove(user_socket)

        del self.clients[user_socket]

        continue

    # handle any other action being passed to the server
    print(f'{client_action_document["action"]}: UserName:
{self.clients[user_socket][0]}')

    action = self.actions[client_action_document["action"]]

    if action is False:

        # this happens when the action type sent to the server isnt known
        user_socket.send(self.pkg_doc_manager(
            "[ERROR - ACTION]", f"Unregistered action type
{client_action_document['action']}"))

        continue

    # start a new thread so that the action that is being sent doesn't halt
    reading other client messages

    _thread.start_new_thread(
        action, (user_socket, client_action_document["data"]))

    continue

def recv_doc_manager(self, client_socket):
    try:
        # Get the header of the package.
        message_header = client_socket.recv(self.HEADERSIZE)

        # This occurs if the user disconnects or sends back no data.
        if not len(message_header):
            return False

        # Remove the extra spaces that we added in the HEADERSIZE and cast the
        # string into an integer.
        document_length = int(message_header.decode('utf-8').strip())

        # Get and store the actual action document that was sent to the server.
        doc = client_socket.recv(document_length)

```

```

    return pickle.loads(doc) # Turn bytes into a python object.

except:
    return False

def pkg_doc_manager(self, action, document):
    """
    Format the document that the server wants to send to the client socket to bytes.
    This will be done by pickling the doc object and adding the heading (length of
    the object in bytes).

    action = The action type that is being packaged up.
    document = The data attached to the action.
    """
    # check if the action and its data are not left empty.
    if not action:
        raise('You can not send an empty action type')
    if not document:
        raise('You can not send an empty document')
    doc = {"action": action, "data": document}

    # This turns the python class into bytes that can be sent to the server.
    pkged_doc = pickle.dumps(doc)

    # The header will contain the length of the pkged_doc in bytes.
    pkged_doc_with_header = bytes(
        f'{len(pkged_doc):<{self.HEADERSIZE}}', 'utf-8')+pkged_doc

    return pkged_doc_with_header # this can be sent over the socket.

```

```

def registration_manager(self, userCredentials):
    """
    Register the username on the database only if the username isnt taken

    userCredentials = ("username", "password")
    """
    try:
        c = self.DB.connection.cursor()
        c.execute("SELECT username FROM users WHERE username = :username",
        {
            "username": userCredentials[0]})
        userCredentials_fromDB = c.fetchone()
        # checks if there already a player with that username

        if userCredentials_fromDB == None:
            # create new user WITH USERNAME AND PASSWORD because there is
            no user with the desired username

            hashedPassword = hashlib.md5() # set hashing algorithm to MD5 <not
            suitable for production>

            hashedPassword.update(bytes(userCredentials[1], 'utf-8')) # hash the
            provided password

            c.execute(
                "INSERT INTO users (username, password) VALUES (?, ?)",
                (userCredentials[0], hashedPassword.hexdigest()))

            self.DB.connection.commit()

            return {"result": True, "msg": "Account was created successfully."}
        else:
            return {"result": False, "msg": "Username already exists."}
    except BaseException as e:

```

```

    print(e)

    return {"result": False, "msg": "Error when creating client's account."}

def login_manager(self, userCredentials: ("username", "password")):
    """
    Handle the authentication of the client.

    This function will query the database for the desired username and compare
    hashed passwords

    if they match, this will return the desired userdata <minus the hashed password>
    """
    try:
        c = self.DB.connection.cursor()

        c.execute("SELECT username, password FROM users WHERE username
= :username", {
            "username": userCredentials[0]})
        userCredentials_fromDB = c.fetchone()

        if userCredentials_fromDB == None:
            # there is no accounts with the passed in username, return error
            return {"result": False, "data": f"No user found with the username:
{(userCredentials[0])}"}

        hashedPassword = hashlib.md5() # set hashing algorithm to MD5 <not suitable
for production>

        hashedPassword.update(bytes(userCredentials[1], 'utf-8')) # hash the provided
password

        # check if hashed passwords match

        if userCredentials_fromDB == (userCredentials[0],
hashedPassword.hexdigest()):

            c.execute("SELECT username, wins, loses, games_played FROM users
WHERE username = ?",
                (userCredentials[0],))

```

```

        userData = c.fetchone()

        return {"result": True, "data": userData} # return user data
    else:
        return {"result": False, "data": "Incorrect password"} # return error
except BaseException as e:
    print(e)
    return {"result": False, "data": "Error when authenticating client's account."}

def joinGame(self, client, data):
    """
    Adds the client to the waiting_queue
    if the client is the first one in the queue they will become host
    - the host will create the game session
    else they will get told to join a game by the server from the host request

    <difficulty> the server can't send the socket class
    """
    try:
        # add user to game queue - session
        self.waiting_queue.add(client)

        host = False

        while (len(self.waiting_queue) < 2): # this client is the only player in the game
            queue

            host = True # this make setting up the game easier as one client will be
            responsible for setting it up

            if client in self.waiting_queue: # if the client is still waiting in the for a
            game lobby

                client.send(self.pkg_doc_manager(

```



```

        "[JOIN GAME - WAITING]", data))
    else:
        # "Client left the queue"
        return
    if host:
        gameId = uuid.uuid1()
        Game_Board_Data = {
            'id': gameId,
            'player_data': [],
            'board': [[0, 0, 0], [0, 0, 0], [0, 0, 0]],
            'player_turn': 1,
        }
        players = []
        for i in range(2): # 2 player game
            player_clinet = self.waiting_queue.pop()
            players.append(player_clinet)
            # let the clinet know that they are getting connected to a game.
            # the socket client and their userData
            Game_Board_Data['player_data'].append(
                self.clients[player_clinet])
        for i, client in enumerate(players):
            client.send(self.pkg_doc_manager(
                "[JOIN GAME - SUCCESS]", Game_Board_Data))
        Game_Board_Data['clients'] = players
        # add the Game_Board_Data to the ongoing games data record
        self.onGoingGames[gameID] = Game_Board_Data
    return
except:

```

```

        self.waiting_queue.remove(client)

    return

def cancelGame(self, client, data):
    """
    remove client from waiting queue
    """

    if client in self.waiting_queue:
        self.waiting_queue.remove(client)
        client.send(self.pkg_doc_manager(
            "[CANCEL GAME - SUCCESS]", "Cancelled"))

    else:
        client.send(self.pkg_doc_manager(
            "[CANCEL GAME - FAIL]", "Not waiting for a game"))

    return

def takeTurn(self, client, data):
    """
    - the client will pass the game id that they want to update
    - then the server will send both clients the updated board

    <difficulty> the server can't send the socket class so i have to create a clone of
    the data that the client sent
    """

    try:
        # update game board
        self.onGoingGames[data["id"]]["board"] = data["board"]
    """

    Update player turn
    example:
        player 1 starts:
        1 % 2 = 1, +1 = 2 turn

```

```

    2 % 2 = 0, + 1 = 1 turn
"""

data["player_turn"] = (data["player_turn"] % 2) + 1
self.onGoingGames[data["id"]]["player_turn"] = data["player_turn"]
# check if any players won
isWinner = self.check_if_winner(
    self.onGoingGames[data["id"]]["board"])
if isWinner != 0:
    # notify players that the game has a winner
    data["winner"] = isWinner
    # go through all players in the current game
    for i, client in enumerate(self.onGoingGames[data["id"]]["clients"]):
        # update database
        if i+1 == data["winner"]:
            # this client won the game
            self.update_user_data_after_game(client, True)
        else:
            # this client lost or drew
            self.update_user_data_after_game(client)
            data["updated_userData"] = self.clients[client]
            client.send(self.pkg_doc_manager("[GAME - END]", data))
    # close game session
    del self.onGoingGames[data["id"]]
    return
for client in self.onGoingGames[data["id"]]["clients"]:
    client.send(self.pkg_doc_manager(
        "[GAME - TURN]", data))
return

```

```

except:

    # close game session

    del self.onGoingGames[data["id"]]

    raise

def check_if_winner(self, board: "Array: board state") -> int:
    """
    checks if there is a winning state
    """

    def allCheck(linearArray):
        """
        this function checks if the first element is the same as the rest of the array
        """

        return linearArray.count(linearArray[0]) == len(linearArray) and
linearArray[0] != 0

    columns = [[], [], []] # used to check if a player won Vertically
    zero_count = 0 # used to determine if the game board is a draw
    for row in board:

        # this will check each horizontal row

        if allCheck(row):

            return row[0]

        for columnIndex, col in enumerate(row):

            # split each col into a new array

            columns[columnIndex].append(col)

            if col == 0:

                zero_count += 1

    for col in columns:

        # this will check each Vertical row

        if allCheck(col):

            return col[0]

```

```

# check top-right to bottom left
if len(set([board[i][i] for i in range(len(board))])) == 1:
    return board[0][0]

# check top-left to bottom right
if len(set([board[i][len(board)-i-1] for i in range(len(board))])) == 1:
    return board[0][len(board)-1]

if zero_count == 0:
    # there is no more position to play/ the game is in a draw state
    return 3

return 0 # no end state (the game is still playable - ongoing)

def update_user_data_after_game(self, client, won=False):
    """
    update the players data on the server and the database after a game
    """
    try:
        c = self.DB.connection.cursor()

        if won:
            c.execute("UPDATE users SET wins = wins+1,
games_played=games_played+1 WHERE username = :username", {
                "username": self.clients[client][0]})
            self.DB.connection.commit()
            print(self.clients[client][0], "won")
        else:
            c.execute("UPDATE users SET loses=loses+1,
games_played=games_played+1 WHERE username = :username", {
                "username": self.clients[client][0]})
            self.DB.connection.commit()
            print(self.clients[client][0], "lost")
    
```

```

        c.execute("SELECT username, wins, loses, games_played FROM users
WHERE username = ?",

                (self.clients[client][0],))

        userCredentials_fromDB = c.fetchone

        # update client's data on the server

        self.clients[client] = userCredentials_fromDB

    except:

        raise

def getAllPlayerStats(self, client, data):
    """
    Query the database for all users statistics and return them in an array
    """
    try:

        c = self.DB.connection.cursor()

        c.execute("SELECT username, wins, loses, games_played FROM users")

        userStatistics_fromDB = c.fetchall()

        client.send(self.pkg_doc_manager(

            "[GET ALL PLAYER STATS - SUCCESS]",
            userStatistics_fromDB))

    except:

        client.send(self.pkg_doc_manager(

            "[GET ALL PLAYER STATS - FAIL]", "Error whilst getting all
player statistics"))

if __name__ == "__main__": # only run this code if this python file is the root file
    execution

    server = SocketServer()

```

1.2. File server_sql_connection.py

```

import sqlite3 # used to talk to the SQL database

class SqlServerConnection():

```

```

def __init__(self, databaseCredential="application.db"):
    # connect with the database
    # check_same_thread needs to be false because it will be running on different
    threads.
    self.connection = sqlite3.connect(databaseCredential, check_same_thread=False)
    # self.connection = sqlite3.connect(":memory:") # testing
    # init db
    self.setup_db()

def setup_db(self):
    # this will create the users and leader board table in the sql database if they dont
    already exists
    c = self.connection.cursor()
    c.execute("""
CREATE TABLE IF NOT EXISTS users (
    username VARCHAR(25) NOT NULL UNIQUE,
    password VARCHAR(25) NOT NULL,
    wins INT UNSIGNED DEFAULT 0,
    loses INT UNSIGNED DEFAULT 0,
    games_played int UNSIGNED DEFAULT 0
);
""")
    self.connection.commit()

```

2. Mã nguồn bên phía máy khách

2.1. File client.py

```

import tkinter as tk # Ui builder
from tkinter import ttk # leaderboard UI builder

```

```

import asyncio

# the client socket controller that is used by the ui/client, this will have all the actions
# that the client can perform.

from client_socket_connection import ClientServerSocket

import _thread

class Application(tk.Tk):

    # can pass in anything as an array

    def __init__(self, *args):
        super().__init__(*args) # inits the inherited Tk class

        # adds some meta data to the application
        self.title("Tic Tac Toa Project")

        # locked the app min dimensions
        self.minsize(700, 600)

        # self.maxsize(700, 600) # i can add a max dimension here

        """

        self is the tk.TK root instance

        creates the first frame and passes the root TK instance to it. This means that all the
        function that are generated in the Application class will be accessible to the container
        frame

        - A frame is like a view for tkinter, im going to be rebuilding frames and
        bringing it to the front of the application window to simulate navigation.

        """

        container = tk.Frame(self)

        self.PreviousFrame = None

        # use full width and height and fill it. define the columns and rows
        container.pack(side="top", fill="both", expand=True, anchor='center')

        container.grid_rowconfigure(0, weight=1)
        container.grid_rowconfigure(1, weight=1)

```



```
container.grid_columnconfigure(0, weight=1)
```

```
container.grid_columnconfigure(1, weight=1)
```

```
# set the socket server connection variable to None, with will be initialize the  
ClientServerSocket class once the user successfully authenticates
```

```
self.SocketConnection: ClientServerSocket = None
```

```
# self.frames will contain all the different views of the application. ie login, lobby  
and game views.
```

```
self.frames = {}
```

```
# go through all the tk.frames and render them, one on top the other
```

```
for FView in (HomePage, JoinGamePage, GamePage, LeaderBoardPage,  
AuthenticationPage):
```

```
# makes a reference to the frame class and feeds it the required information.  
container is the the passed down TK and the single view that will get updated every  
time i want to switch views
```

```
frame = FView(container, self)
```

```
# saves the instance in self.frames
```

```
self.frames[FView] = frame
```

```
# adds the initial widget rendered.
```

```
# frame.grid(row=0, sticky="news")
```

```
# Goes to the login page
```

```
self.PreviousFrame = self.frames[AuthenticationPage]
```

```
self.switch_frame_to(AuthenticationPage)
```

```
def switch_frame_to(self, frame: tk.Frame):
```

```
# tries to find the frame instance in self.frames
```

```
frame = self.frames[frame]
```

```
if frame != None: # check if there is an initialize frame that has been created.
```

```

        self.PreviousFrame.grid_remove() # remove the last frame so the page will
        resize to best fit height and width

        frame.grid(row=0, sticky="news")

        self.PreviousFrame = frame

        # then re-build the frame but with the new frame.

        frame.render() # each frame view should have this method, it allows me to re-
        render views with updated Data

        frame.tkraise() # brings the frame to the front of the window.

    return

def authenticate_user(self, Frame: tk.Frame, userCredentials: "(username: String,
Passowrd: String)", socketHostData: "(HostName: String, Port: Number)":
    try:
        if ((len(userCredentials[0]) > 0) and (len(userCredentials[1]) > 5)):
            print("Attempt to authenticate client", userCredentials[0])

            # initialize the ClientServerSocket class and point to the socket server with
            its address and port

            self.SocketConnection = ClientServerSocket(socketHostData)

            # attempt to authenticate the client and wait for the login function to return.
            result = asyncio.run(
                self.SocketConnection.login(userCredentials))

            if self.SocketConnection.isAuth == True:
                print("Client authenticated: True")

                # Switch to the home page

                Frame.err_label.grid_remove()

                self.switch_frame_to(HomePage)

                return

            else:
                print("Client authenticated: Fail")

                # Display any additional information

```

```

        Frame.ERROR_MSG.set(result)

        Frame.err_label.grid()
    else:
        # display error message

        Frame.ERROR_MSG.set("Password must be 6 characters or more" if (len(
            userCredentials[0]) > 0) else "Username must be 1 character or more")

        Frame.err_label.grid()
except ConnectionError as e:
    print(e)

    Frame.ERROR_MSG.set("Can't Reach the Server Socket")

    Frame.err_label.grid()


def register_user(self, Frame: tk.Frame, userCredentials: "(username: String,
Passowrd: String)", socketHostData: "(HostName: String, Port: Number)":

    try:
        if ((len(userCredentials[0]) > 0) and (len(userCredentials[1]) > 5)):
            print("Attempt to create new user account", userCredentials[0])

            # initialize the ClientServerSocket class and point to the socket server with
its address and port

            self.SocketConnection = ClientServerSocket(socketHostData)

            result = asyncio.run(

                self.SocketConnection.register(userCredentials)) # attempt to create a
new user account in the database and wait for the register function to return.

            # Display any additional information

            Frame.ERROR_MSG.set(result)

            Frame.err_label.grid()
        else:
            # display error message

```

```

        Frame.ERROR_MSG.set("Password must be 6 characters or more") if (len(
            userCredentials[0]) > 0) else Frame.ERROR_MSG.set("Username must
            be 1 character or more")

```

```

        Frame.err_label.grid()

```

```

    except ConnectionError as e:

```

```

        print(e)

```

```

        Frame.ERROR_MSG.set("Can't Reach the Server Socket")

```

```

        Frame.err_label.grid()

```

```

class HomePage(tk.Frame): # inherit from the tk frame

```

```

    """

```

```

    The main page - shown when the user is successfully authenticated

```

```

    - The client will be presented with the home page rendered view which will
    include the following:

```

```

        - The currently logged in user's account statistics.

```

```

        - "Join game" button that will tell the server that this client wants to join a game
        and to add it in the waiting list queue

```

```

        - "View leader board" button that will switch the view to the Leader Board Page.

```

```

    + at the bottom of the view there is a section for displaying any error, the message
    is the value of self.ERROR_MSG

```

```

    """

```

```

def __init__(self, parent, controller):

```

```

    tk.Frame.__init__(self, parent) # inits the inherited frame class

```

```

    # parent is the class that called this class

```

```

    # print(self.__dict__) # shows me the attributes of this variable

```

```

    self.controller = controller

```

```

    self.parent = parent

```

```

    self.ERROR_MSG = tk.StringVar()

```

```

    self.render()

```

```

def render(self):
    """
    Render the home page view
    """

    if self.controller.SocketConnection != None:

        # "Welcome: " + username

        # - Wins: {self.controller.SocketConnection.userData[1]} - Loses:
        {self.controller.SocketConnection.userData[2]} - Total games played:
        {self.controller.SocketConnection.userData[3]}

        tk.Label(self, text=f"Welcome:
        {self.controller.SocketConnection.userData[0]}", font=(
            "arial", 15, "bold")).grid(row=0, column=0, ipadx=20, pady=10, padx=60,
            sticky="news")

        tk.Button(self, text='Join A Game', font=("arial", 20, "bold"), command=lambda:
        self.join_game(
            )).grid(row=1, padx=130, ipadx=80, ipady=20, pady=10, sticky="news")

        tk.Button(self, text='Leader-board', font=("arial", 20, "bold"), command=lambda:
        self.controller.switch_frame_to(LeaderBoardPage)).grid(row=2,
            padx=130, ipadx=80, ipady=20, pady=90, sticky="news")

        self.err_label = tk.Label(
            self, textvariable=self.ERROR_MSG, font=("arial", 20, "bold"))
        self.err_label.grid(row=3, rowspan=2, ipadx=10, sticky="ews")

def join_game(self):
    """
    join game lobby queue,
    if successful-> enter game session.
    else -> go back to the home page if the player decides to quit the queue.
    """

```

```

    print("Join game queue")

    self.controller.switch_frame_to(JoinGamePage)

    # this allows the player to cancel at any time whilst waiting for someone else to
    join the game.

    _thread.start_new_thread(self.waiting_to_Join, ())

def waiting_to_Join(self):
    try:
        res = asyncio.run(self.controller.SocketConnection.joinGame())

        except AttributeError: # this will throw an error if the client is trying to get into a
game without being authenticated by the server

            # send the client to the login page

            self.controller.switch_frame_to(AuthenticationPage)

            raise UserWarning("Currently not signed in")

    if self.controller.SocketConnection.isInGame is True:

        self.controller.switch_frame_to(GamePage)

        # start game loop

        asyncio.run(

self.controller.SocketConnection.startGameLoop(self.controller.frames[GamePage]))

        self.ERROR_MSG.set("")

    else:

        self.controller.switch_frame_to(HomePage)

        self.ERROR_MSG.set(res)

    return

class JoinGamePage(tk.Frame): # inherit from the tk frame
    """

```

The Join game page - shown when the client attempts to join a game session and is waiting for a player to join the game.

- The client will be presented with the rendered view which will include the following:

- A message of what's happening ("Waiting For Another Player To Join...")
- A button to leave the queue and to go back to the home view

"""

```
def __init__(self, parent, controller):
```

```
    tk.Frame.__init__(self, parent) # inits the inherited frame class
```

```
    # parent is the class that called this class
```

```
    # print(self.__dict__) # shows me the attributes of this variable
```

```
    self.controller = controller
```

```
    self.parent = parent
```

```
    self.render()
```

```
def render(self):
```

```
    tk.Label(self, text="Waiting For Another Player To Join...", font=("", 22,)).grid(
        row=0, column=0, columnspan=3, padx=80, ipadx=30, ipady=90,
        sticky="news")
```

```
    tk.Button(self, text="Cancel", font=("arial", 20, "bold"),
        command=self.cancel_game).grid(
        row=1, column=1, pady=20, ipadx=80, ipady=20, sticky="ews")
```

```
def cancel_game(self):
```

```
    """
```

```
    Leave the game waiting queue and navigate back to the home page
```

```
    """
```

```
    try:
```

```

        asyncio.run(self.controller.SocketConnection.cancelGame())

    except AttributeError: # this will throw an error if the user manages to get un
authenticated by the server and tries to leave game the queue

        # send the client to the login page

        self.controller.switch_frame_to(AuthenticationPage)

        raise UserWarning("Currently not signed in")

```

```

class GamePage(tk.Frame): # inherit from the tk frame
    """
    The main page - shown when the user is successfully authenticated
    """

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent) # inits the inherited frame class
        # parent is the class that called this class
        # print(self.__dict__) # shows me the attributes of this variable
        self.controller = controller
        self.parent = parent

        self.MSG = tk.StringVar()

        # render view
        self.render()

    def render(self):
        # game board section
        if self.controller.SocketConnection != None:
            gameBoard_section = tk.Frame(self)

```



```

self.renderBoard(gameBoard_section)

gameBoard_section.grid(sticky="news", row=0,
                        column=0, padx=130, pady=20)

tk.Label(self, text=f"Player 1:
{self.controller.SocketConnection.gameData['player_data'][0][0]}", font=(
    "arial", 14)).grid(row=1, sticky="nsw", padx=130)

tk.Label(self, text=f"Player 2:
{self.controller.SocketConnection.gameData['player_data'][1][0]}", font=(
    "arial", 14)).grid(row=2, sticky="nsw", padx=130)

tk.Label(self, text=f"Turn: Player
{self.controller.SocketConnection.gameData['player_turn']}", font=(
    "arial", 14)).grid(row=3, sticky="nsw", padx=130)

# Setup Message Label
end_of_gmae_section = tk.Frame(self)

self.end_game_btn = tk.Button(end_of_gmae_section, text="Go Back to
home", font=(
    "arial", 14), command=lambda: self.controller.switch_frame_to(HomePage))
self.end_game_btn.grid(
    row=0, column=1, ipadx=30, ipady=20, padx=10, pady=10)
self.end_game_btn.grid_remove()
self.msg_label = tk.Label(
    end_of_gmae_section, textvariable=self.MSG, font=("arial", 14))
self.msg_label.grid(row=0, sticky="news")
self.msg_label.grid_remove()
end_of_gmae_section.grid(row=4, sticky="news")

def renderBoard(self, boardSection):
    """

```

- from the board array it will output the game board in a frame.

<difficulty> Tkinter doesn't like buttons being generated in a for loop, (all buttons will have the same command as the last button generated)

ie in a 3 x 3 board you will always select position (2,2) AKA the last box.

"""

```
for row in range(len(self.controller.SocketConnection.gameData["board"])):
    for column in range(len(self.controller.SocketConnection.gameData["board"]
[ row ])):
        slotState = self.controller.SocketConnection.gameData["board"][row]
[ column]
        if slotState == 2:
            tk.Button(boardSection, bg="#0000FF", state="disabled").grid(
                row=row, column=column, ipadx=50, ipady=40, padx=10, pady=10)
        elif slotState == 1:
            tk.Button(boardSection, bg="#FF0000", state="disabled").grid(
                row=row, column=column, ipadx=50, ipady=40, padx=10, pady=10)
        else:
            # <difficulty> so i have to explicitly tell the lambda function that i want to
            use the variables in passed at the time of the button was created
            tk.Button(boardSection, command=lambda row=row, col=column:
self.take_turn(row, col)).grid(
                row=row, column=column, ipadx=60, ipady=40, padx=10, pady=10)
```

```
def take_turn(self, row, Column):
    rowColumn = (row, Column)
    try:
        if (self.controller.SocketConnection.userData[0] ==
self.controller.SocketConnection.gameData["player_data"]
[self.controller.SocketConnection.gameData["player_turn"]-1][0]):
            asyncio.run(
                self.controller.SocketConnection.take_turn(rowColumn))
```

```

        self.msg_label.grid_remove()
except:
    self.MSG.set("It is not your turn")
    self.msg_label.grid()
pass

```

```

class LeaderBoardPage(tk.Frame): # inherit from the tk frame

```

```

    """

```

The leaderBoard page - shows the top players, and lets the user search up any players rank (including thier own)

```

    """

```

```

def __init__(self, parent, controller):

```

```

    tk.Frame.__init__(self, parent) # inits the inherited frame class

```

```

    # parent is the class that called this class

```

```

    # print(self.__dict__) # shows me the attributes of this variable

```

```

    self.controller = controller

```

```

    self.parent = parent

```

```

    # variables

```

```

    self.MSG = tk.StringVar()

```

```

    self.MSG.set("Your Rank Is:")

```

```

    self.userDatas = []

```

```

    # render view

```

```

    self.render()

```

```

def render(self):

```

```

if self.controller.SocketConnection != None:

    # username will be used to lookup a single user's ranking
    username = tk.StringVar()
    username.set(self.controller.SocketConnection.userData[0])

    asyncio.run(self.get_all_userdata_sorted())

    Leaderboard = ttk.Treeview(self, columns=('username', 'wins', 'losses',
'games_played'))
    Leaderboard.column('#0', width=60, anchor='center')
    Leaderboard.heading('#0', text='Ranking')
    Leaderboard.column('username', width=120, anchor='center')
    Leaderboard.heading('username', text='Username')
    Leaderboard.column('wins', width=60, anchor='center')
    Leaderboard.heading('wins', text='Wins')
    Leaderboard.column('losses', width=60, anchor='center')
    Leaderboard.heading('losses', text='Losses')
    Leaderboard.column('games_played', width=120, anchor='center')
    Leaderboard.heading('games_played', text='Games Played')

    for index in range(0, (20 if (len(self.userDatas) > 19) else len(self.userDatas))):
# Display top 20 players!
        Leaderboard.insert("", tk.END, text=str(index+1),
values=self.userDatas[index] )

    Leaderboard.grid(row=0, padx=10, pady=20,
                    ipady=140, sticky="nw")

# right hand side section
right_action_section = tk.Frame(self)

```

```

right_action_section.grid(
    row=0, rowspan=2, column=1, ipady=120, pady=20)

tk.Button(right_action_section, text="Update Board", font=(
    "arial", 16), command=self.render).grid(row=0, columnspan=2,
sticky="new", ipadx=10, ipady=10)

# search player section
search_player_section = tk.Frame(right_action_section)
search_player_section.grid(
    row=1, rowspan=2, columnspan=2, padx=20, pady=70)

tk.Label(
    search_player_section, text="Username:", font=("arial", 14)).grid(row=0,
column=0, sticky="news")

tk.Entry(
    search_player_section, textvariable=username).grid(row=0, column=1,
pady=20, padx=10, ipady=10, ipadx=20, sticky="swe")

tk.Button(search_player_section, text="Search \nPlayer Rank", font=(
    "arial", 16), command=lambda:
self.findUserRank(username.get())).grid(row=1, column=0, sticky="new", padx=5,
ipadx=20)

tk.Button(search_player_section, text="Get My \nPosition", font=(
    "arial", 16), command=lambda:
self.findUserRank(self.controller.SocketConnection.userData[0])).grid(row=1,
column=1, sticky="new", padx=5, ipadx=20)

tk.Button(search_player_section, text="Go Back to home", font=(

```

```

        "arial", 14), command=lambda:
self.controller.switch_frame_to(HomePage)).grid(row=3, columnspan=2, column=0,
pady=10, padx=20, ipadx=30, ipady=20)

```

```

        tk.Label(search_player_section, textvariable=self.MSG, wrap=255, font=(
        "arial", 14)).grid(row=4, columnspan=2, sticky="sw", pady=20)

```

```

async def get_all_userdata_sorted(self):
    """
    get all user datas from the database via the server connection
    """
    # get all user stats data from the server (username, wins, loses, games_played)
    # fetch_all_user_stats = [("isaac", 12,0,12), ("test1", 0,10,10), ("test2", 4,10,14),
    ("test3", 10,4,14)]
    try:
        self.MSG.set("")
        fetch_all_user_stats = await
self.controller.SocketConnection.getAllPlayerData()
        if fetch_all_user_stats != True:
            self.MSG.set(fetch_all_user_stats)
            self.userDatas = self.controller.SocketConnection.leaderboard
    except:
        raise

def findUserRank(self, username: str):
    for i, userdata in enumerate(self.userDatas):
        if userdata[0] == username:
            return self.MSG.set(f"Rank of {username}: {i+1} ")
        else:
            pass
    return self.MSG.set(f"Could not find {username}")

```

```

class AuthenticationPage(tk.Frame): # inherit from the tk frame
    """
    The main page - shown when the user is successfully authenticated
    """

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent) # inits the inherited frame class
        # parent is the class that called this class
        # print(self.__dict__) # shows me the attributes of this variable
        self.controller = controller
        self.parent = parent
        self.ERROR_MSG = tk.StringVar()
        self.render()

    def render(self):
        """
        Render the authentication page view
        """
        # variables
        username = tk.StringVar()
        password = tk.StringVar()
        hostname = tk.StringVar()
        port_number = tk.IntVar()

        # testing login info:
        # username.set("test1")

```

```

# password.set("test12")

# set up default server connection value
# server is running on the current computer

# hostname.set("localhost") # server is running on the current computer
port_number.set(4201) # on this port number

# authentication section
authentication_section = tk.LabelFrame(
    self, text="Authentication", font=("arial", 20, "bold"))

username_label = tk.Label(
    authentication_section, text="Username:", font=("arial", 14))
username_entry = tk.Entry(
    authentication_section, textvariable=username)
password_label = tk.Label(
    authentication_section, text="Password:", font=("arial", 14))
password_entry = tk.Entry(
    authentication_section, show="*", textvariable=password)

```

"""

placing Widgets:

placement:

- row = vertical placement
- column = horizontal placement

alignment:

- sticky = n="top", e="right", s="bottom", w="left" alignment

padding:

- padx = the extra space around the x axis
- pady = the extra space around the y axis
- ipadx = the extra space inside the widget on the x axis
- ipady = the extra space inside the widget on the y axis

```

"""
username_label.grid(row=0, column=0, sticky="nsw", ipadx=40, ipady=10)
username_entry.grid(row=0, column=1, sticky="nsw",
                    ipadx=40, ipady=10, padx=10, pady=10)
password_label.grid(row=1, column=0, sticky="nsw", ipadx=40, ipady=10)
password_entry.grid(row=1, column=1, sticky="nsw",
                   ipadx=40, ipady=10, padx=10, pady=10)
authentication_section.grid(
    row=0, sticky="news", padx=130, ipadx=20, pady=20, ipady=10)

# server info section
server_info_section = tk.LabelFrame(
    self, text="Server Info", font=("arial", 20, "bold"))

hostname_label = tk.Label(
    server_info_section, text="Host name:", font=("arial", 15))
hostname_entry = tk.Entry(server_info_section, textvariable=hostname)
port_number_label = tk.Label(
    server_info_section, text="Port number:", font=("arial", 15))
port_number_entry = tk.Entry(
    server_info_section, textvariable=port_number)

```

```

hostname_label.grid(row=0, column=0, sticky="nsw", padx=40, ipady=10)
hostname_entry.grid(row=0, column=1, sticky="nsw",
                    padx=40, ipady=10, pady=10, padx=10, pady=10)
port_number_label.grid(
    row=1, column=0, sticky="nsw", padx=40, ipady=10)
port_number_entry.grid(row=1, column=1, sticky="nsw",
                      padx=40, ipady=10, pady=10, padx=10, pady=10)
server_info_section.grid(row=1, sticky="news",
                        padx=130, ipadx=20, pady=20, ipady=10)

# action section
action_section = tk.Frame(self)
# make the action section share single column
action_section.grid_columnconfigure(0, weight=1)
action_section.grid_columnconfigure(1, weight=1)
action_section.grid_rowconfigure(0, weight=1)
action_section.grid_rowconfigure(1, weight=1)

# Make sure that the fields are not empty
loginBtn = tk.Button(action_section, text='Login', font=("arial", 20, "bold"),
command=lambda: self.controller.authenticate_user(
    self, (username.get(), password.get()), (hostname.get(), port_number.get()))
registerBtn = tk.Button(action_section, text='Register', font=("arial", 20, "bold"),
command=lambda: self.controller.register_user(
    self, (username.get(), password.get()), (hostname.get(), port_number.get()))

loginBtn.grid(row=0, column=0, sticky="news",
             padx=5, pady=5, ipady=10, padx=10)
registerBtn.grid(row=0, column=1, sticky="news",

```

```

        ipadx=5, pady=5, ipady=10, padx=10)
    action_section.grid(row=2, sticky="news", padx=130,
        ipadx=20, pady=20, ipady=10)

    # Setup Error Message Label
    self.err_label = tk.Label(self, textvariable=self.ERROR_MSG)
    self.err_label.grid(row=3, sticky="news")
    self.err_label.grid_remove()

if __name__ == "__main__": # only run this code if this python file is the root file
    execution
    try:
        # starts the application
        app = Application()
        app.mainloop()
    except:
        # catches for any unauthenticated.
        # print("Unable to get Authenticated user")
        pass

```

2.2. File client_sql_connection

```

import socket # used to create the client socket connection with the server socket.
import pickle # parser that is used to accept and send any python class.
# set up the socket
class ClientServerSocket(socket.socket):
    def __init__(self, socketHostData=(socket.gethostname(), 4201)):
        super().__init__(socket.AF_INET, socket.SOCK_STREAM)
        """

```

- socket.AF_INET is saying our socket host's IP is going to be a IPv4 (Internet Protocol version 4)

- socket.SOCK_STREAM is saying that the port that the socket will be using is a TCP (Transmission Control Protocol)

```
"""
```

```
try:
```

```
    self.connect(socketHostData)
```

```
except BaseException as e:
```

```
    print(e)
```

```
    raise ConnectionError(f"Could not connect to the HostName:  
{socketHostData[0]}, Port: {socketHostData[1]} - It may not exist or be down.")
```

```
    # define the size of the length of the header needs to be the same on the server
```

```
    self.HEADERSIZE = 10
```

```
    # is the user successfully authenticated?
```

```
    self.isAuth = False
```

```
    self.userData = None
```

```
    self.gameData = None
```

```
    # is client waiting to join a game?
```

```
    self.isWaiting = False
```

```
    # is client currently in a game?
```

```
    self.isInGame = False
```

```
    # Leaderboard
```

```
    self.leaderboard = None
```

```
    # if self.isAuth is False:
```

```
    #     raise(BaseException("Password Or Username Was Incorrect"))
```

```
async def recv_doc_manager(self):
```

```
    try:
```

```
        # get the header size
```

```

message_header = self.recv(self.HEADERSIZE)

# this occurs if the user disconnects or sends back no data
if not len(message_header):
    return False

# this line will remove the extra spaces that we added in the HEADERSIZE
and cast the string into a integer
document_length = int(message_header.decode('utf-8').strip())

# this will store the actual document that was sent to the server
doc = self.recv(document_length)
return pickle.loads(doc)

except BlockingIOError:
    return

except:
    return False

def pkg_doc_manager(self, action, document):
    if not action:
        raise(BaseException("You can't send an empty action type"))
    if not document:
        raise(BaseException("You can't send an empty document"))
    doc = {"action": action, "data": document}
    # this turns the python class into bytes that can be sent to the server
    pkged_doc = pickle.dumps(doc)
    # The header will contain the length of the pkged_document bytes
    pkged_doc_with_header = bytes(
        f'{len(pkged_doc):<{self.HEADERSIZE}}', 'utf-8') + pkged_doc
    return pkged_doc_with_header

async def login(self, userCredentials):
    """

```

Attempt to authenticate the client with a username and password on the socket client

```
"""
```

```
# checks if the client isn't already authenticated
```

```
if self.isAuth is False:
```

```
    packaged_auth_login_document = self.pkg_doc_manager(
```

```
        "[USER LOGIN]", userCredentials)
```

```
    self.send(packaged_auth_login_document)
```

```
    results = await self.recv_doc_manager()
```

```
    # nothing was sent back, something broke on the server (disconnected)
```

```
    if results is None:
```

```
        return False
```

```
    print(results["action"])
```

```
    if results["action"] == "[USER LOGIN - FAIL]":
```

```
        # user failed to authenticate client
```

```
        return results["data"]
```

```
    # successfully authenticated client's account
```

```
    self.userData = results["data"]
```

```
    self.isAuth = True
```

```
    return True
```

```
async def register(self, userCredentials):
```

```
    """
```

```
    Register the user on the socket server
```

```
    """
```

```
# checks if the client isn't already authenticated
```

```
if self.isAuth is False:
```

```
    packaged_auth_register_document = self.pkg_doc_manager(
```

```
        "[USER REGISTER]", userCredentials)
```

```
    self.send(packaged_auth_register_document)
```

```

results = await self.recv_doc_manager()

# nothing was sent back, something broke on the server (disconnected)
if results is None:
    return "Error: no connection to the socket"

if results["action"] == "[USER REGISTER - FAIL]":
    # failed to create user account client
    return results["data"]

# successfully created user account
return results["data"]

async def joinGame(self):
    # checks if the client is already authenticated
    if self.isAuth is True and self.isWaiting is False and self.isInGame is False:
        packaged_join_game_request_document = self.pkg_doc_manager(
            "[JOIN GAME]", self.userData[0])
        self.send(packaged_join_game_request_document)
        self.isWaiting = True
        results = await self.recv_doc_manager()

        if results is None: # nothing was sent back, something broke on the server
            (disconnected)
            self.isWaiting = False
            return "Error: no connection to the socket"

        while results["action"] == "[JOIN GAME - WAITING]":
            results = await self.recv_doc_manager()
            if results is None:
                self.isWaiting = False
                return "Error: no connection to the socket"

        if results["action"] == "[CANCEL GAME - FAIL]":
            # no in the waiting game queue
            return results["data"]

```

```

self.isWaiting = False

# if results["action"] == "[JOIN GAME - CANCELLED]":
#     # cancelled game
#     return results["data"]
#     # the client is connecting
if results["action"] == "[JOIN GAME - SUCCESS]":
    # successfully joined a game
    self.isInGame = True
    self.gameData = results["data"]
    return True

if results["action"] == "[CANCEL GAME - SUCCESS]":
    # cancelled game
    self.isWaiting = False
    return results["data"]

async def cancelGame(self):
    # checks if the client is already authenticated
    if self.isAuth is True and self.isWaiting is True and self.isInGame is False:
        packaged_leave_game_queue_document = self.pkg_doc_manager(
            "[CANCEL GAME]", self.userData[0])
        self.send(packaged_leave_game_queue_document)

async def startGameLoop(self, frame):
    try:
        results = await self.recv_doc_manager()

        if results is None: # nothing was sent back, something broke on the server
            (disconnected)
            self.isWaiting = False
            return "Error: no connection to the socket"

```



```

while results["action"] == "[GAME - TURN]":
    self.gameData = results["data"]
    frame.render()
    results = await self.recv_doc_manager()
    if results is None:
        self.isInGame = False
        return "Error: no connection to the socket"

if results["action"] == "[GAME - END]":
    self.gameData = results["data"]
    self.userData = self.gameData["updated_userData"]
    # game ended
    if self.gameData["winner"] == 3:
        frame.MSG.set("Draw!")
    else:
        frame.MSG.set("Player "+ str(self.gameData["winner"])+ " has won!")
    frame.render()
    frame.msg_label.grid()
    frame.end_game_btn.grid()
    self.isInGame = False
    return
except:
    self.isInGame = False
    self.gameData = None
    raise

async def take_turn(self, rowCol):
    # make sure the player is in a game
    if self.isAuth is True and self.isInGame is True:

```

```

        # update board

        self.gameData["board"][rowCol[0]][rowCol[1]] =
self.gameData["player_turn"]

        # send the take turn action to the server

        packaged_game_board_action_document = self.pkg_doc_manager(
            "[TAKE TURN]", self.gameData)

        self.send(packaged_game_board_action_document)

    return

async def getAllPlayerData(self):

    # make sure the client is authenticated and not in a game

    try:

        if self.isAuth is True and self.isInGame is False:

            packaged_get_player_all_data_action_document = self.pkg_doc_manager(
                "[GET ALL PLAYER STATS]", self.userData[0])

            self.send(packaged_get_player_all_data_action_document)

            results = await self.recv_doc_manager()

            # nothing was sent back, something broke on the server (disconnected)

            if results is None:

                return "Error: no connection to the socket"

            if results["action"] == "[GET ALL PLAYER STATS - FAIL]":

                # failed to get user statistics

                self.leaderboard = None

                return results["data"]

            # successfully get user statistics

            self.leaderboard = self.insertion_sort(results["data"]) # sort the player data

            return True

    except:

        raise

def insertion_sort(self, userStats):

```

```

# TODO: implement insertion sort here

for userIndex in range(1, len(userStats)): # go through the whole array so each
user has been sorted.
    # i dont have to so minus 1 cause in range(starting point, endpoint but not
including >)
    current_user = userStats[userIndex]
    position = userIndex
    while (position > 0) and (userStats[position-1][1] < current_user[1]):
        # checks if the lower user index is lower and the index number is checking
both index 1 and 0
        userStats[position] = userStats[position-1]
        position -= 1

    # place the current userdata in its highest possition
    userStats[position] = current_user

# userStats is now sorted - with the players with the highest numbers of wins
last.

return userStats

if __name__ == "__main__": # only run this code if this python file is the root file
execution
    try:
        s = ClientServerSocket()
        s.login(("test", "tEst3_14159"))

    except ConnectionError as e:
        print(e)

```

TÀI LIỆU THAM KHẢO

TIẾNG VIỆT

- [1]. *Giáo trình mạng máy tính* Biên tập bởi: Ngô Bá Hùng, Phạm Thế Phi
- [2]. *Giáo trình nhập môn mạng máy tính* – Hồ Đắc Phương

TIẾNG ANH

- [3] *Networking: A Beginner's Guide*, Second Edition BRUCE HALLBERG pp.15-45, 75-94,
- [4] *Learning Python Network Programming* Dr.M.O Faruque Sarker Sam Washington pp.177-208
- [5] *SimpleSQLite Documentation* Release 1.3.0 Tsuyoshi Hombashi
- [6] Trang chủ tài liệu của Python <https://docs.python.org/3.7/>