

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN I**

-----🙞🙜🕮🙞🙜-----

****

**BÁO CÁO BÀI TẬP LỚN**

**MÔN: LẬP TRÌNH MẠNG**

|  |  |  |
| --- | --- | --- |
| *Giảng viên hướng dẫn:* | **Ths. Nguyễn Hoàng Anh** | |
| *Nhóm lớp:* | Nhóm 14 | |
| *Nhóm bài tập lớn:* | Nhóm 06 |  |
| *Sinh viên thực hiện:* | Nguyễn Đức Hoằng | B17DCAT084 |
|  | Trần Minh Nhật | B17DCAT139 |
|  | Phạm Hải Vũ | B17DCAT214 |
|  | Nguyễn Công Thành | B17DCAT167 |

***Hà Nội, tháng 11 2020***

|  |  |
| --- | --- |
|  | **ĐỀ TÀI**  **Game AMONGCHU thi đấu đối kháng online**  *Giáo viên hướng dẫn:*  Ths. Nguyễn Hoàng Anh |

# Lời cảm ơn

Trước tiên với lòng biết ơn sâu sắc nhất, chúng em xin gửi đến quý thầy cô tại Học viện Công nghệ Bưu chính Viễn thông lời cảm ơn chân thành vì đã cùng với tri thức và tâm huyết của mình để truyền đạt vốn kiến thức quý báu cho chúng em trong thời gian học tập tại học viện.

Trong học kỳ này, Học viện đã cho chúng em được học và tiếp cận với các môn học nền tảng rất hữu ích đối với sinh viên, đặc biệt là môn học Lập trình mạng. Bước đầu đi vào thực tế, tìm hiểu sâu hơn về môn học, kiến thức của chúng em còn hạn chế và các kỹ năng còn nhiều thiếu sót. Chúng em xin chân thành cảm ơn thầy – Ths Nguyễn Hoàng Anh đã tận tâm hướng dẫn chúng em thực hiện và hoàn thành môn học này của mình. Thầy đã luôn tạo điều kiện, nhiệt tình và đưa ra những lời khuyên, bài học bổ ích cho lớp chúng em trong suốt thời gian học môn học.

Cuối cùng chúng em xin kính chúc quý thầy cô đang công tác tại Học viện Công nghệ Bưu chính Viễn thông sức khỏe, niềm tin để tiếp tục thực hiện sứ mệnh cao đẹp của mình là truyền đạt kiến thức cho thế hệ nối tiếp.

Chúng em xin chân thành cảm ơn!

Hà Nội, tháng 11 năm 2020

Nhóm 06

# Mục lục

[Lời cảm ơn 3](#_Toc57756392)

[Mục lục 4](#_Toc57756393)

[Danh mục phân công công việc 6](#_Toc57756394)

[Danh mục ký hiệu sử dụng 7](#_Toc57756395)

[Danh mục viết tắt 8](#_Toc57756396)

[Danh mục hình ảnh 9](#_Toc57756397)

[1. Tổng quan 10](#_Toc57756398)

[1.1. Giới thiệu game AmongChu thi đấu đối kháng online 10](#_Toc57756399)

[1.2. Phân tích yêu cầu ứng dụng/hệ thống 10](#_Toc57756400)

[1.2.1. Yêu cầu ứng dụng 10](#_Toc57756401)

[1.2.2. Yêu cầu hệ thống 11](#_Toc57756402)

[2. Phân tích thiết kế 12](#_Toc57756403)

[2.1. Phân tích thiết kế tổng quan 12](#_Toc57756404)

[2.1.1. Kiến trúc tổng quan 12](#_Toc57756405)

[2.1.2. Sơ đồ chức năng 12](#_Toc57756406)

[2.1.3. Biểu đồ Usecase tổng quan 12](#_Toc57756407)

[2.2. Phân tích thiết kế chi tiết 12](#_Toc57756408)

[2.2.1. Đặc tả Usecase 12](#_Toc57756409)

[2.2.2. Biểu đồ lớp 12](#_Toc57756410)

[2.2.3. Biểu đồ tuần tự 12](#_Toc57756411)

[2.2.4. Sơ đồ thực thể quan hệ (ER) 12](#_Toc57756412)

[3. Kết quả ứng dụng 13](#_Toc57756413)

[3.1. Kiến trúc ứng dụng 13](#_Toc57756415)

[3.1.1. Giới thiệu Non-blocking IO trong Java 13](#_Toc57756416)

[3.1.2. Kiến trúc Server 15](#_Toc57756417)

[3.1.2.1. Database 16](#_Toc57756418)

[3.1.2.2. ServerQueueRequest 17](#_Toc57756419)

[3.1.2.3. ServerTableControl 18](#_Toc57756420)

[3.1.3. Kiến trúc Client 19](#_Toc57756421)

[3.1.3.1. ResponseHandler 19](#_Toc57756422)

[3.1.3.2. LoginControl 19](#_Toc57756423)

[3.1.3.3. LobbyControl 19](#_Toc57756424)

[3.1.3.4. PikachuController 20](#_Toc57756425)

[3.2. Cài đặt và triển khai ứng dụng 20](#_Toc57756426)

[3.2.1. Cấu trúc thư mục 20](#_Toc57756427)

[3.2.2. Triển khai 21](#_Toc57756428)

[3.3. Kết quả thực hiện 21](#_Toc57756429)

[3.3.1. Các kết quả thực hiện thành công 21](#_Toc57756430)

[3.4. Kết luận 23](#_Toc57756431)

[3.4.1. Tổng kết 23](#_Toc57756432)

[3.4.2. Hạn chế cần khắc phục: 23](#_Toc57756433)

[Danh mục tài liệu tham khảo 24](#_Toc57756434)

# Danh mục phân công công việc

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và Tên** | **Phân công công việc** | **Tỉ lệ hoàn thành** |
| 1 | Nguyễn Đức Hoằng | Mô tả bài toán  Lâp trình Server  Dựng Database  Lắp ráp code của các thành viên khác  Deloy lên Server  Tổng hợp báo cáo |  |
| 2 | Trần Minh Nhật | Tài trợ Server  Code và viết báo cáo các phần liên quan đến đăng nhập, đăng ký. |  |
| 3 | Nguyễn Công Thành | Code và viết báo cáo các phần liên quan đến đăng xuất, thách đấu, . |  |
| 4 | Phạm Hải Vũ | Code website quản trị  Code và viết báo cáo các phần liên quan đến đăng xuất, thách đấu, . |  |

# Danh mục ký hiệu sử dụng

|  |  |  |
| --- | --- | --- |
| KHÁI NIỆM | KÝ HIỆU | Ý NGHĨA |
| Tác nhân  (Actor) |  | Một người / nhóm người hoặc một thiết bị hoặc hệ thống tác động hoặc thao tác đến chương trình. |
| Use-case |  | Một chuỗi các hành động mà hệ thống thực hiện mang lại một kết quả quan sát được đối với actor. |
| Hệ thống  (System) |  | Biểu hiện phạm vi của hệ thống. Các use-case được đặt trong khung hệ thống. |
| Lớp  (Class) |  | Là một sự trừu tượng của các đối tượng trong thế giới thực. |
| Lớp biên  (Boundary class) |  | Nắm giữ sự tương tác giữa phần bên ngoài với phần bên trong của hệ thống (giao diện chương trình). |
| Lớp điều khiển  (Control class) |  | Thể hiện trình tự xử lý của hệ thống trong một hay nhiều use-case. |
| Lớp thực thể  (Entity class) |  | Mô hình hóa các thông tin lưu trữ lâu dài trong hệ thống, nó thường độc lập với các đối tượng khác ở xung quanh. |

# Danh mục viết tắt

# Danh mục hình ảnh

[Hình 1: Kiến trúc máy chủ sử dụng Blocking socket 11](#_Toc57755029)

[Hình 2: Kiến trúc tổng quan máy chủ sử dụng Non-blocking socket 12](#_Toc57755030)

[Hình 3: Cấu trúc máy chủ cơ bản sử dụng Non-blocking socket 12](#_Toc57755031)

[Hình 4: Cấu trúc cơ sở dữ liệu 14](#_Toc57755032)

[Hình 5: Quá trình lập lịch 15](#_Toc57755033)

[Hình 6: Cấu trúc ServerTableControl 16](#_Toc57755034)

[Hình 7: Kiến trúc Client tổng quan 17](#_Toc57755035)

[Hình 8: Giao diện đăng nhập 19](#_Toc57755036)

[Hình 9: Giao diện sảnh chờ 19](#_Toc57755037)

[Hình 10: Giao diện trò chơi 20](#_Toc57755038)

# Tổng quan

## Giới thiệu game AmongChu thi đấu đối kháng online

Game Pikachu là một trong những game kinh điển, mặc dù đã được ra mắt 17 năm nhưng sức hút của trò chơi này vẫn không hề bị giảm nhiệt. Với lối chơi cuốn hút, kịch tính hiện game Pikachu vẫn là sự lựa chọn hàng đầu khi người chơi muốn xả stress sau những giờ làm việc căng thẳng và mệt mỏi. Trong phiên bản này, nhóm chúng em mong muốn giới thiệu tới thầy và các bạn một phiên bản đối kháng của game Pikachu cố điển kết hợp với những hình ảnh từ một tựa game rất nổi tiếng hiện nay mang tên “AmongUs”.

## Phân tích yêu cầu ứng dụng/hệ thống

### Yêu cầu ứng dụng

Để tham gia game AmongChu thi đấu đối kháng online, người chơi sẽ cần sử dụng một tài khoản đăng nhập vào hệ thống. Tài khoản này có thể được tạo thông qua chức năng Đăng kí. Server sau đó sẽ gửi tới người chơi thông tin về những người chơi khác đang online, điểm số hiện tại của họ cũng như của cá nhân người chơi. Người chơi có thể sử dụng tính năng thách đấu để mời một người chơi vào trận đấu với mình, đối phương nhận được lời mời có thể từ chối hoặc chấp nhận lời thách đấu.

Khi đối phương chấp nhận thách đấu, cả hai sẽ cùng được cung cấp một màn chơi AmongChu 10x10 ô là hình ảnh các nhân vật trong tựa game AmongUs.

Luật chơi:

* Người chơi cần tìm 2 ô có hình ảnh nhân vật AmongUs giống nhau và nối chúng lại để chúng biến mất khỏi màn hình, điều kiện để 2 ô này nối thành công với nhau là chúng cần phải cách nhau không quá 3 đường gấp khúc và trên đường này không chứa vật cản là bất kì một ô nào khác.
* Trò chơi sẽ kết thúc khi một trong hai người chơi hoàn thành việc nối toàn bộ những ô hình nhân vật có trong bàn chơi hoặc khi thời gian kết thúc với kết quả hòa. Thời gian giới hạn cho mỗi bàn chơi là 2 phút. Qua mỗi bàn chơi, người chơi sẽ nhận được những điểm số tương ứng:
  + Thắng nhận 1 điểm.
  + Thua nhận 0 điểm.
  + Hòa mỗi người chơi nhận 0.5 điểm.
* Điều kiện chiến thắng:
  + Người chơi hoàn thành bàn chơi trước đối thủ.
  + Đối thủ thoát bàn chơi.
  + Cả hai cùng không hoàn thành bàn chơi trước 2 phút, người chơi chiến thắng sẽ là người có số điểm cao hơn.
* Kết quả hòa chỉ xảy ra khi cả hai cùng không hoàn thành bàn chơi và cùng có số điểm sau 2 phút.
* Sau khi kết thúc bàn chơi, người dùng sẽ được hỏi có muốn tiếp tục thách đấu với đối phương hay không, nếu cả hai trả lời có, hệ thống sẽ lại xếp cặp và tạo trận đấu giữa hai người chơi.

Kết quả các trận đấu được lưu vào Server. Mỗi người chơi đều có thể vào xem bảng xếp hạng các người chơi trong toàn bộ hệ thống, theo lần lượt các tiêu chí: tổng số điểm (giảm dần), trung bình điểm của các đối thủ đã gặp (giảm dần), trung bình thời gian kết thúc trong các trận thắng (tăng dần).

### Yêu cầu hệ thống

# Phân tích thiết kế

## Phân tích thiết kế tổng quan

### Kiến trúc tổng quan

### Sơ đồ chức năng

### Biểu đồ Usecase tổng quan

## Phân tích thiết kế chi tiết

### Đặc tả Usecase

### Biểu đồ lớp

### Biểu đồ tuần tự

### Sơ đồ thực thể quan hệ (ER)

# Kết quả ứng dụng

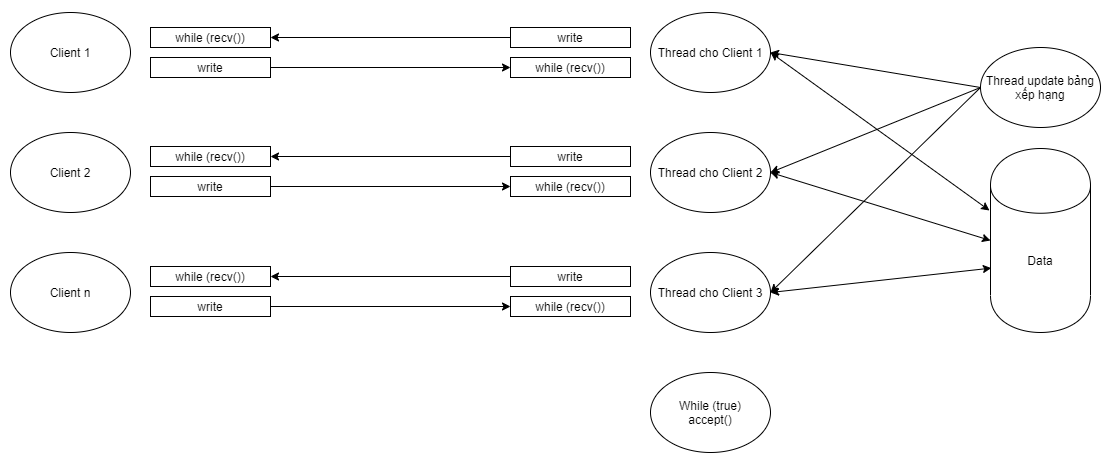


## Kiến trúc ứng dụng

### Giới thiệu Non-blocking IO trong Java

Trong phần này, chúng em mong muốn giới thiệu với người đọc về một kiến trúc được sử dụng rất rộng rãi trong việc dựng máy chủ hiện nay, đó chính là Asynchronous Non-blocking IO. Kiến trúc này còn được biết đến trên hệ điều hành Linux qua cái tên Epoll và được triển khai rất nhiều thông qua libuv. Trong phạm vi bài tập lớn này, nhóm chúng em sẽ sử dụng thư viện java.nio được hỗ trợ trong ngôn ngữ lập trình Java để tạo nên kiến trúc chính trong ứng dụng của mình.

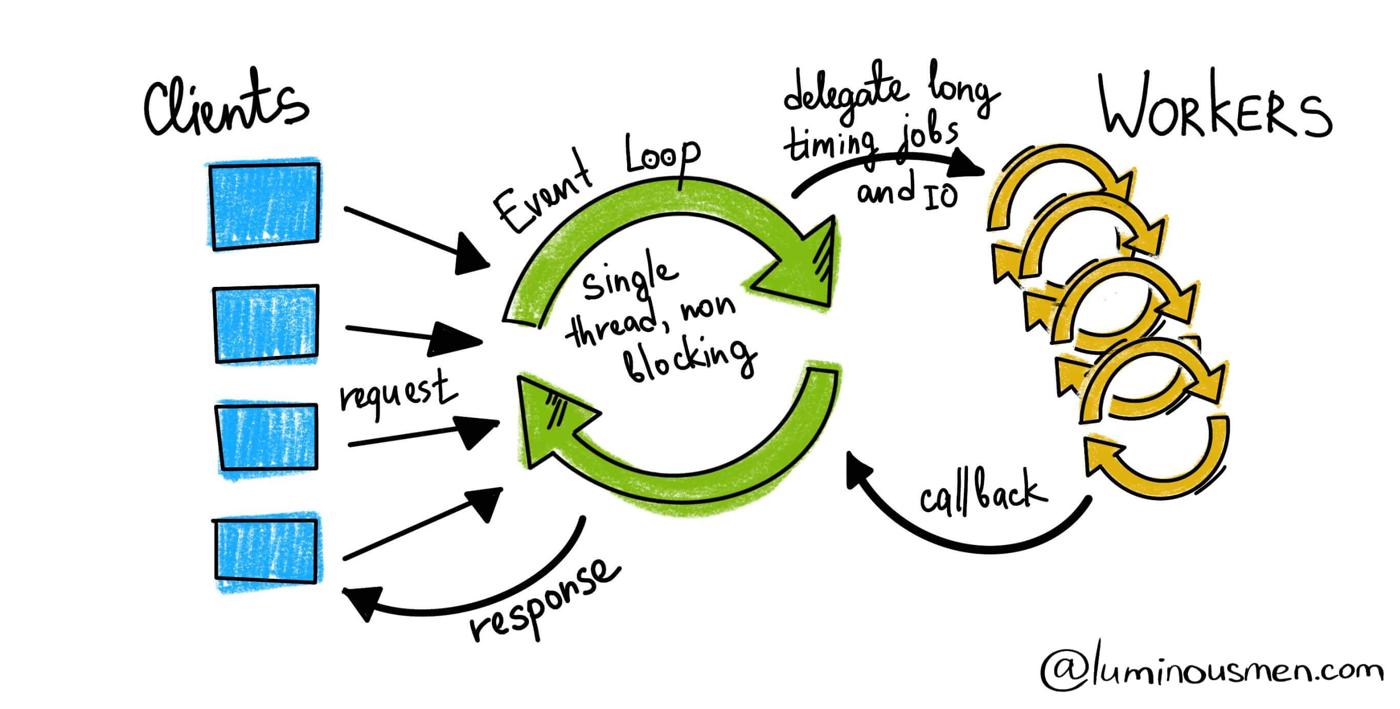
Đầu tiên, chúng em sẽ chỉ ra những hạn chế mà nhóm đã gặp phải trong việc sử dụng Blocking IO socket:



Hình : Kiến trúc máy chủ sử dụng Blocking socket

* Với đa số máy chủ thông dụng theo kiến trúc cũ này, thời gian và khả năng xử lí của CPU hầu hết được dành cho việc IO hay đơn giản là không làm gì cả. Với mỗi Client riêng biệt, Server sẽ phải dành ra 2 Thread: Một cho việc xử lí và ghi dữ liệu, một cho việc đọc dữ liệu tương ứng. Điều này gây ra sự lãng phí hiệu năng rất lớn khi mà số lượng Client tăng lên hoặc tăng số cổng/số host mà Server/Client phải kết nối đến.
* Vấn đề về Race condition: Với kiến trúc trên, các Thread của mỗi Client sẽ hoạt động độc lập. Tuy nhiên một số dữ liệu và bộ nhớ vẫn được dùng chung ví dụ như: danh sách Client, bảng xếp hạng,… Điều này đặt ra một bài toán rất lớn về việc xử lí một khối lượng khổng lồ các lỗi Race condition có thể gặp phải hay chính xác hơn là bài toán lập lịch cho Server.

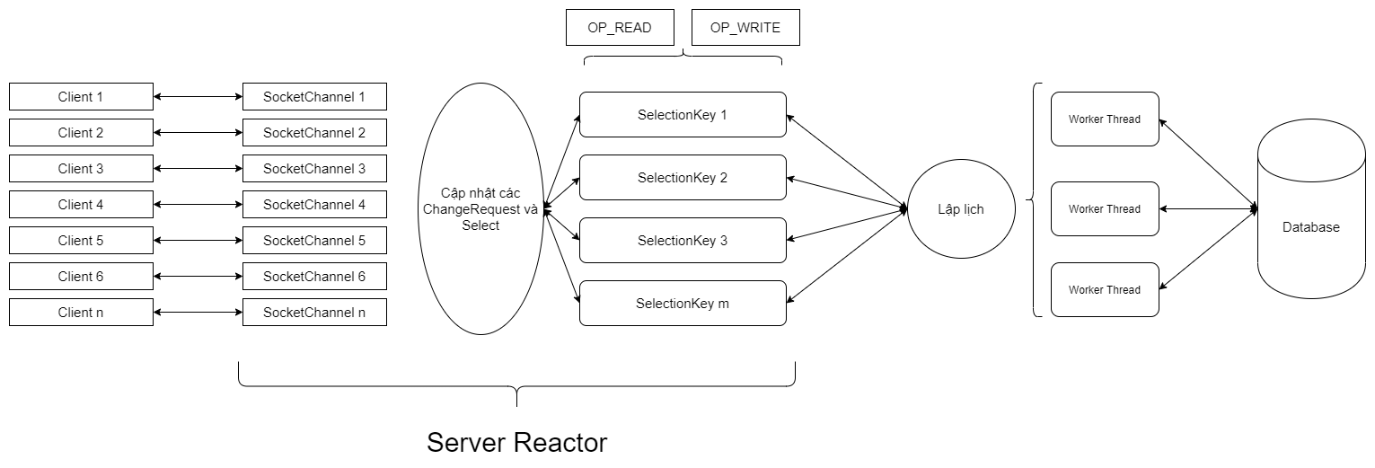
Để khắc phục những điều trên, nhóm chúng em đã sử dụng Non-blocking IO trong Java với kiến trúc như sau:



Hình : Kiến trúc tổng quan máy chủ sử dụng Non-blocking socket

Đây là một kiến trúc tách biệt hoàn toàn việc IO và xử lí dữ liệu. Nó có thể chia ra 2 phần:

* Server Reactor: Bao gồm 1 thread với chức năng select các request cần được đọc/ghi và xử lí các thao tác tương ứng đối với mỗi Client.
* Server Worker: Có thể bao gồm 1 hoặc nhiều thread. Những thread này sẽ chịu trách nhiệm nhận truy vấn từ Server Reactor, lập lịch, xử lí và trả dữ liệu về cho Server Reactor nếu cần IO với Client.



Hình : Cấu trúc máy chủ cơ bản sử dụng Non-blocking socket

Chúng em sẽ chỉ ra những điểm mạnh của cấu trúc này thông qua quá trình hoạt động:

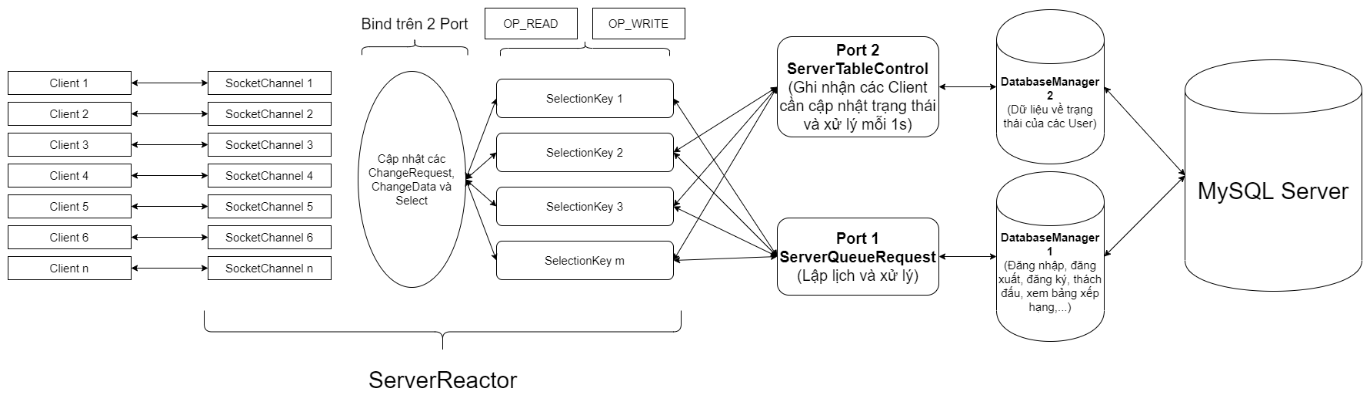
* ServerReactor khi bắt đầu hoạt động sẽ đăng kí các ServerSocketChannel (có tác dụng bind và listen trên các interface) hoặc SocketChannel (Kết nối tới Server khác) với một Selector. Điều này giúp người sử dụng chỉ cần với 1 thread duy nhất vừa có thể đóng vai trò một hay nhiều Server và cùng lúc đó kết nối tới một hoặc nhiều Server khác.
* Selector này có nhiệm vụ chọn ra những ServerSocketChannel hoặc SocketChannel có thể tương tác (ví dụ như read, write, accept, connect) và trả về một List các SelectionKey. Các hoạt động (Operation) tương tác này có thể được dự đoán trước thông qua việc sử dụng:

SelectionKey.interestOps([OP\_READ hoặc OP\_WRITE])

Thông thường trong quá trình hoạt động, một SelectionKey luôn được đặt trong chế độ OP\_READ để lắng nghe các truy vấn từ Server/Client. Ngược lại OP\_WRITE chỉ được đặt khi Client/Server hiện tại muốn gửi đi một truy vấn tới bất kì một Server/Client nào khác. Điều này giúp cho người sử dụng dễ dàng tùy biến được dữ liệu đọc, ghi tới bất kì một trong những kết nối nào đang được mở. Ngoài ra, việc các truy vấn được xử lí theo một List các SelectionKey sẽ giúp các truy vấn được thực hiện theo một thứ tự nhất định thay vì đồng loạt xử lí.

* Dữ liệu sau khi được tiếp nhận bởi ServerReactor sẽ được lập lịch xử lí và chuyển tới các WorkerThread giúp tránh Race condition trong quá trình vận hành. Trong trường hợp cần trả về dữ liệu ngược lại cho Client/Server, các WorkerThread này sẽ trả về các ChangeRequest và ChangeData về cho ServerReactor. Các request này sẽ được thực thi trước mỗi lần Selector thực hiện select.

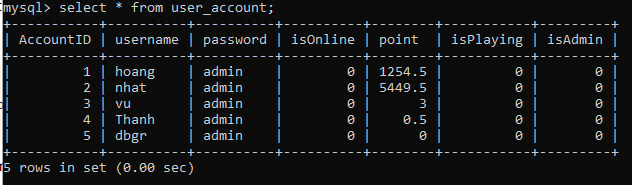
### Kiến trúc Server



Hình 4: Kiến trúc Server tổng quan

Tương tự kiến trúc bên trên đã trình bày, Server được dựng lên theo kiến trúc Non-blocking IO, trong đó chỉ sử dụng một thread duy nhất cho việc lập lịch và xử lý là ServerQueueRequest và một thread giúp trả lời các yêu cầu về bảng trạng thái mang tên ServerTableControl. Hai Port tương ứng cũng được bind cho mỗi chức năng. Mỗi thread sẽ tạo một handle riêng biệt tới MySQL Server.

#### Database



Hình : Cấu trúc cơ sở dữ liệu

* Các cột trong Database bao gồm:

CREATE TABLE `user\_account` (

`AccountID` int NOT NULL AUTO\_INCREMENT,

`username` varchar(255) NOT NULL,

`password` varchar(255) NOT NULL,

`isOnline` tinyint(1) DEFAULT '0',

`point` float DEFAULT '0',

`isPlaying` tinyint(1) DEFAULT '0',

`isAdmin` tinyint(1) DEFAULT '0',

PRIMARY KEY (`AccountID`)

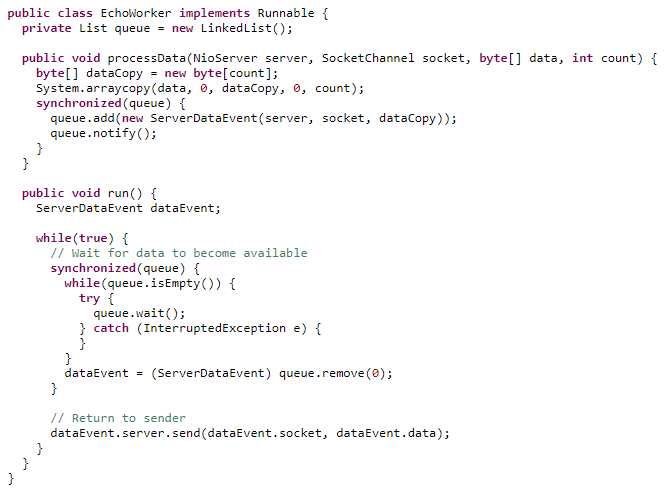
)

* Trong đó:
  + username: Tên tài khoản của người dùng.
  + password: Mật khẩu của người dùng.
  + isOnline: Được set giá trị True khi User online, ngược lại nhận giá trị False.
  + point: Số điểm hiện tại.
  + isPlaying: Trạng thái của người chơi khi đã online (Trong trận đấu/Rảnh rỗi).
  + isAdmin: Người chơi có phải Admin (dùng cho quản trị trên nền tảng web).

#### ServerQueueRequest

##### Lập lịch

Những request tương ứng với Port 1 sẽ được Thread này lập lịch và xử lý và trả về cho ServerReactor.



Hình : Quá trình lập lịch

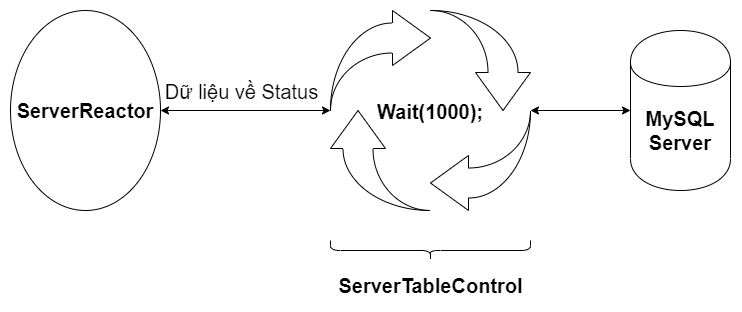
Trong hình 6 bên trên, khi ServerReactor nhận được dữ liệu từ phía Client sẽ gọi tới hàm processData(), hàm này sẽ có nhiệm vụ thêm mới request vào queue để lập lịch xử lý. Sau khi thêm mới xong, hàm sẽ notify phần ThreadMain đang chờ đợi dữ liệu xử lý.

##### Xử lý

ServerQueueRequest sẽ thực hiện xử lý các truy vấn sau:

* **Đăng nhập:** ClientMessage.REQUEST.LOGIN
* **Tạo tài khoản:** ClientMessage.REQUEST.REGISTER
* **Đăng xuất:** Được xử lí tương tự như khi Client ngắt kết nối đột ngột.
* **Thách đấu:** ClientMessage.REQUEST.CHALLENGE
* **Chiến thắng:** ClientMessage.REQUEST.WIN (nhận được khi Client hoàn thành bàn chơi trước đối thủ).
* **Thoát bàn chơi:** ClientMessage.REQUEST.QUIT (nhận được khi Client thoát khi đang trong bàn chơi với đối thủ khác).
* **Hết thời gian:** ClientMessage.REQUEST.MATCH (nhận được khi Client và đối thủ cùng không hoàn thành bàn chơi sau 2 phút, cả 2 sẽ phải gửi thông tin điểm số về Server)
* **Khi Client disconnect:** Server sẽ thực thi:
  + Đặt giá trị isOnline = False và isPlaying = False;
  + Xóa các lời mời thách đấu (nếu có)
  + Nếu người chơi đang trong một trận đấu, điểm số sẽ được cộng cho đối thủ cùng với thông báo về việc thoát của người chơi. Đồng thời lưu lại dữ liệu về trận đấu.

#### ServerTableControl

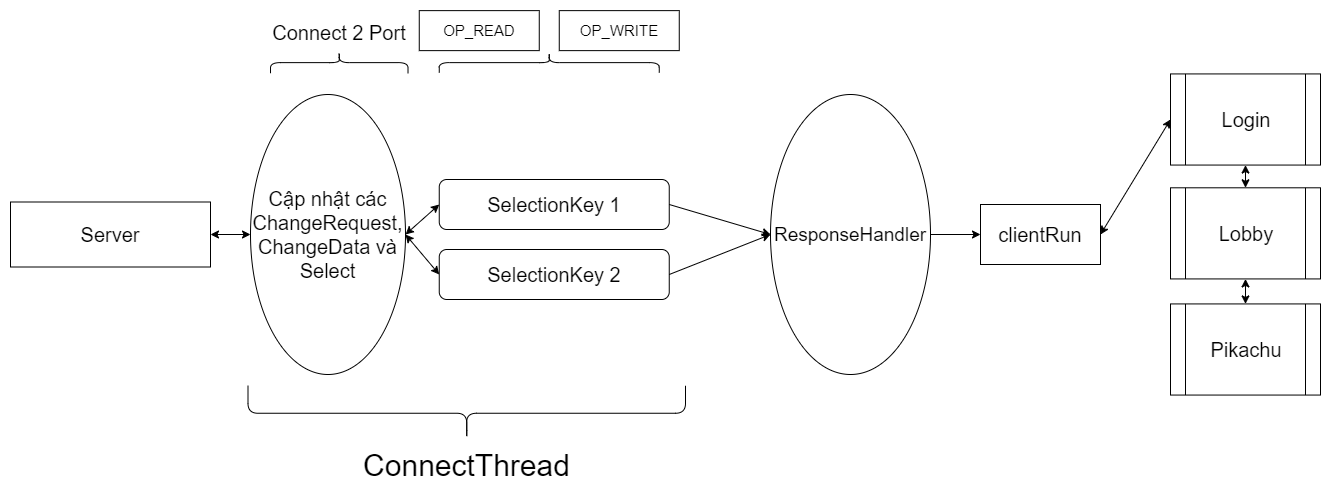


Hình : Cấu trúc ServerTableControl

ServerTableControl sẽ có những nhiệm vụ tương ứng với Port 2 như sau:

* Ghi nhận những Client kết nối đến Port 2 vào một List.
* Mỗi 1 giây thực hiện những công việc sau:
  + Đọc dữ liệu về trạng thái của các User từ MySQL Server.
  + Tạo các ChangeRequest và ChangeData thông qua ServerReactor gửi tới Client.

### Kiến trúc Client



Hình : Kiến trúc Client tổng quan

Tương tự kiến trúc của Server, Client được dựng lên theo kiến trúc Non-blocking IO, trong đó chỉ sử dụng một thread duy nhất cho việc lập lịch và xử lý là ResponseHandler, một thread mang tên ConnectThread chịu trách nhiệm nhận và gửi các bàn tin tới Server.

#### ResponseHandler

##### Lập lịch

* Việc lập lịch của Client hoàn toàn giống Server

##### Xử lý

ResponseHandler sẽ xử lý những thông điệp từ Server sau:

* **Kết quả đăng nhập:** ServerMessage.REQUEST.LOGIN
* **Kết quả đăng ký:** ServerMessage.REQUEST.REGISTER
* **Dữ liệu về trạng thái của các User:** ServerMessage.REQUEST.TABLE
* **Lời mời thách đấu:** ServerMessage.REQUEST.CHALLENGE
* **Kết quả lời mời thách đấu:** ServerMessage.REQUEST.CHALLENGE

#### LoginControl

LoginControl thực hiện những công việc sau:

* Khởi tạo LoginForm.
* Khởi tạo kết nối đến Port 1 trên Server.
* Listen các Action liên quan đến đăng nhập, đăng ký từ LoginForm, tạo các ClientMessage tương ứng gửi lên Server và thông báo kết quả.

#### LobbyControl

LoginControl thực hiện những công việc sau:

* Khởi tạo LobbyForm.
* Khởi tạo kết nối tới Port 2 trên Server.
* Cập nhật dữ liệu về trạng thái người dùng, điểm số lên LobbyForm.
* Listen các Action liên quan đến thách đấu, đăng xuất từ LobbyForm, tạo các ClientMessage tương ứng gửi lên Server và thông báo kết quả.

#### PikachuController

PikachuController thực hiện những công việc sau:

* Khởi tạo PlayGameView.
* Tính toán thời gian, progress bar cho PlayGameView.
* Listen các Action liên quan đến click nhân vật, thoát game từ PlayGameView, trò chơi kết thúc, … và tạo các ClientMessage tương ứng gửi lên Server sau đó thông báo kết quả.

## Cài đặt và triển khai ứng dụng

### Cấu trúc thư mục

* Cấu trúc thư mục Client:
  + Controller:
    - clientRun.java
    - ConnectThread.java
    - ResponseHandler.java
    - LoginControl.java
    - LobbyControl.java
    - PikachuController.java
  + Model:
    - ClientDataEvent.java
    - Pikachu.java
  + View:
    - JpanelBackground.java
    - LobbyForm.java
    - LoginForm.java
    - PlayGameView.java
    - RequestForm.java
* Resource (Chứa các tập tin hình ảnh nhân vật AmongUs, icon,…)
* Cấu trúc thư mục Server:
  + Controller:
    - DatabaseManager.java
    - ServerQueueRequest.java
    - ServerReactor.java
    - serverRun.java
    - ServerTableControl.java
  + Model:
    - Message:
      * ClientMessage.java
      * ServerMessage.java
    - ChangeRequest.java
    - Match.java
    - Matrix.java
    - ServerDataEvent.java
    - UserAccount.java
    - UserTable.java
    - UserTableData.java
* Utils:
  + Utils.java (Chứa thông tin cài đặt về Port 1, Port 2, Hostname,…)

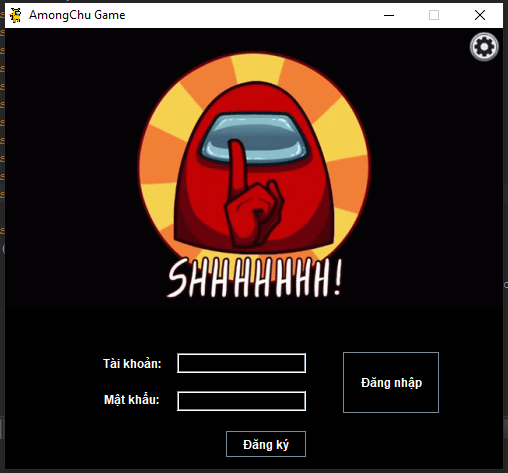
### Triển khai

* Game hiện tại đã được triển khai tại host: [\\34.92.139.146](file:///\\34.92.139.146) với 2 port 12346 và 12347.
* Toàn bộ source code được public tại: <https://github.com/tconqueror/AmongChu>

## Kết quả thực hiện

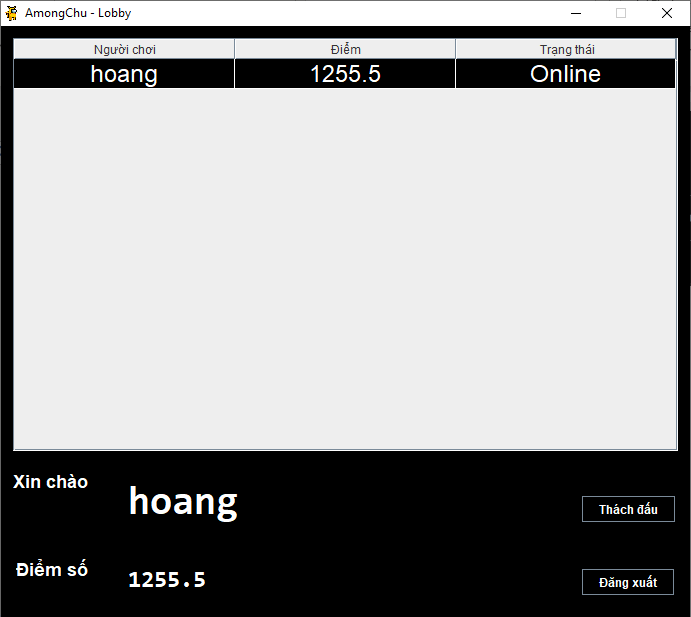
Danh sách kết quả thực hiện thành công

* Đăng nhập:



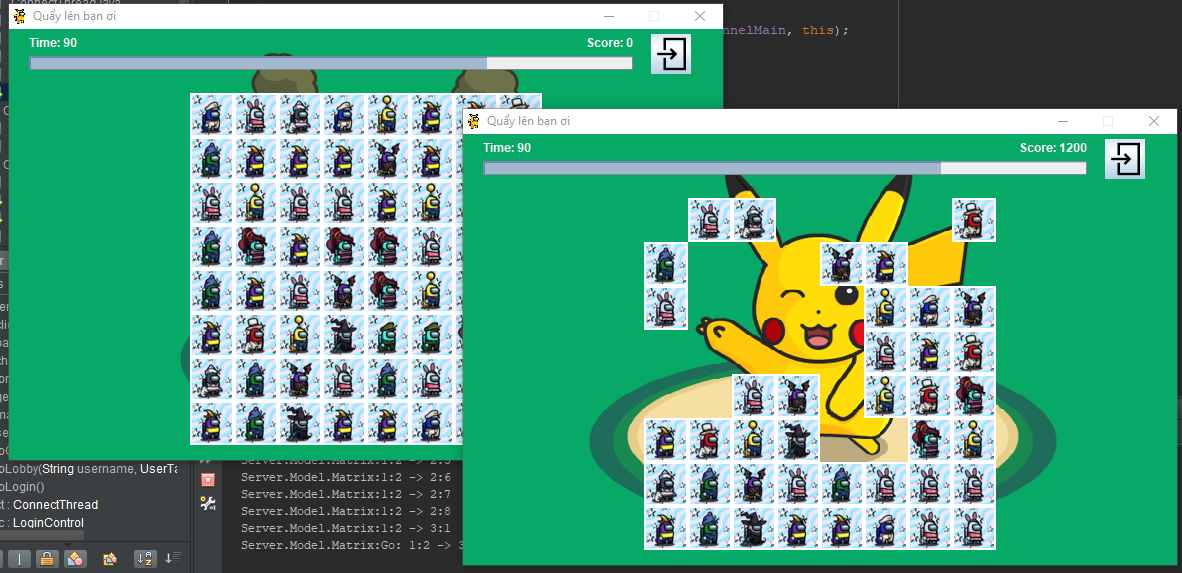
Hình : Giao diện đăng nhập

* + Đăng nhập thành công.
  + Đăng nhập thất bại.
  + Đã đăng nhập nơi khác.
* Đăng ký:
  + Đăng ký thành công
  + Tài khoản đã tồn tại
* Sảnh chờ



Hình : Giao diện sảnh chờ

* Đăng xuất
* Thách đấu: Một người chơi hoàn toàn có thể thách đấu nhiều người một lúc
  + Người chơi hiện không Online
  + Người chơi đã trong trận đấu
* Chấp nhận thách đấu: Một người chơi có thể nhận được nhiều lời mời thách đấu trong cùng một thời điểm. Khi người chơi click chấp thuận, các lời mời còn lại sẽ bị xóa khỏi giao diện và người chơi được đưa vào giao diện thi đấu.
* Chơi game:



Hình : Giao diện trò chơi

* + Thời gian đếm ngược
  + Thoát game
  + Gửi thông báo kết quả trận đấu
  + Toàn bộ trường hợp trong game có thể xảy ra.
* Đấu lại
* Xử lí toàn bộ các Exception xảy ra khi Client mất kết nối.

## Kết luận

### Tổng kết

Pikachu là một tựa game rất nổi tiếng và đã được nhóm chúng em làm mới lại theo thể thức thi đấu đối kháng kết hợp với những hình ảnh sinh động từ tựa game AmongUs với cái tên “AmongChu”

### Hạn chế cần khắc phục:

* Lỗi SQL Injection.
* Cần kiểm tra dữ liệu Table tại Server thay vì Client.
* Giao diện chưa được đẹp mắt.
* Bảng xếp hạng chưa hoàn thiện.

# Danh mục tài liệu tham khảo