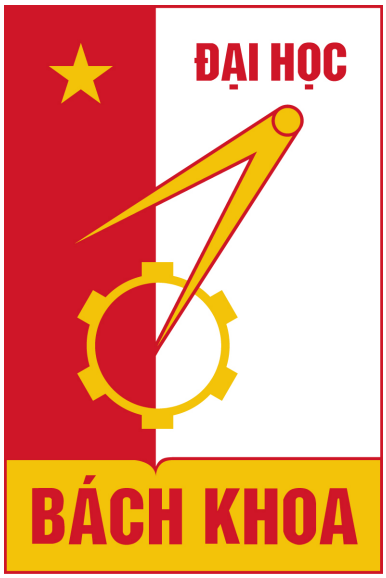
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
o0o



Báo cáo dự án: FurBook - Mạng xã hội cho thú cưng

Môn học: Đồ án 2

Giáo viên hướng dẫn: Nguyễn Thị Thu Hương
Sinh viên thực hiện: Dương Quang Hưng - 20225001
Lớp: IT1 - K67

Hà Nội - 2025

Mục lục

Lời cảm ơn

Trước tiên, em xin gửi lời tri ân sâu sắc đến cô Nguyễn Thị Thu Hương vì đã tận tình hỗ trợ và hướng dẫn em trong suốt quá trình học tập và thực hiện bài tập lớn. Đồng thời, em cũng chân thành cảm ơn các thầy cô trong khoa Công nghệ thông tin và truyền thông đã tạo điều kiện thuận lợi, giúp em có cơ hội trau dồi kiến thức. Trong quá trình thực hiện bài tập, do giới hạn về kiến thức và kinh nghiệm, em khó tránh khỏi những thiếu sót. Em rất mong nhận được sự hỗ trợ và những góp ý quý báu từ cô để có thể hoàn thiện hơn và phát triển bản thân. Cuối cùng, em xin kính chúc cô dồi dào sức khỏe, tràn đầy nhiệt huyết để tiếp tục truyền đạt tri thức cho các thế hệ sinh viên sau này.

Mở đầu

Dự án FurBook được thực hiện nhằm tạo ra một mạng xã hội dành riêng cho thú cưng và những người yêu thương chúng, hỗ trợ các tính năng cơ bản như quản lý người dùng, kết bạn, nhắn tin, đăng bài blog và tin tức về thú cưng thất lạc hoặc tìm thấy thú cưng. Ngoài ra đây cũng là cơ hội để em va chạm và học hỏi thêm kiến thức mới.

Dự án lấy cảm hứng từ kiến trúc microservices, mô phỏng với Docker và triển khai với Kubernetes cho phần backend với ngôn ngữ Golang, đồng thời sử dụng React cho phần frontend nhằm mang lại trải nghiệm người dùng mượt mà, linh hoạt. Ngoài ra, dự án tích hợp API bản đồ OpenStreetMap và thư viện Leaflet để hiển thị vị trí trên bản đồ, phục vụ cho các tính năng tìm kiếm thú cưng thất lạc.

Báo cáo này trình bày chi tiết quá trình thiết kế, triển khai và đánh giá các chức năng chính của FurBook, cùng với phân tích tổng quan hệ thống và các công nghệ được áp dụng. Qua đó, góp phần làm rõ những thuận lợi, khó khăn trong việc phát triển các ứng dụng web phức hợp sử dụng kiến trúc microservices.

Chương 1

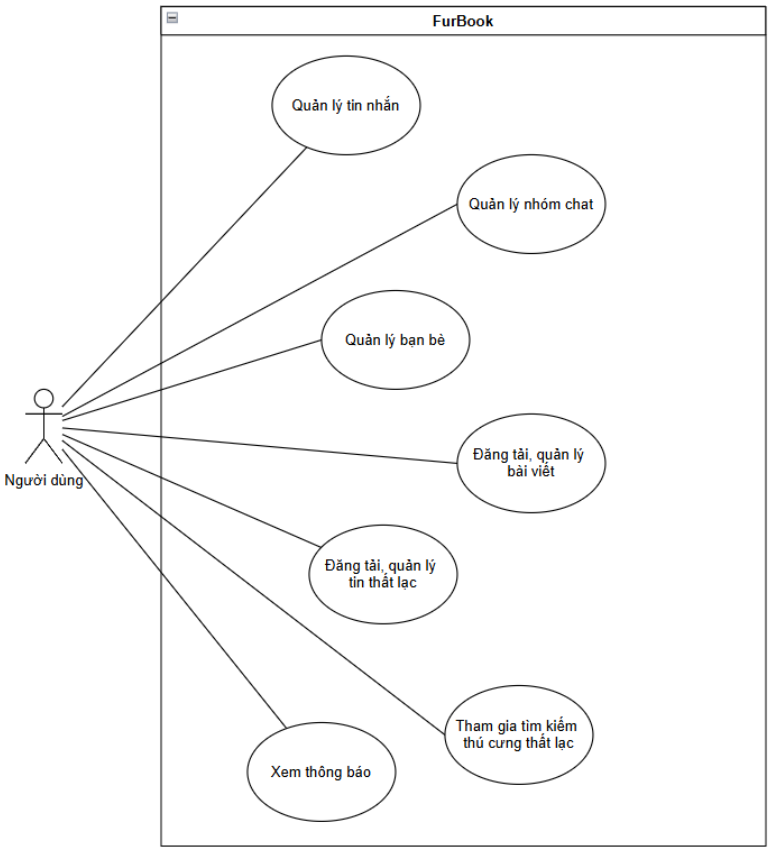
Tổng quan hệ thống

1.1 Mục tiêu dự án

Dự án FurBook nhằm xây dựng một nền tảng mạng xã hội thân thiện và tiện lợi dành cho cộng đồng người nuôi thú cưng. Mục tiêu chính bao gồm:

- Cung cấp các tính năng cơ bản của mạng xã hội: đăng ký, đăng nhập, quản lý thông tin cá nhân, kết bạn, nhắn tin.
- Cho phép người dùng đăng bài viết dạng blog, chia sẻ thông tin, hình ảnh về thú cưng.
- Tích hợp chức năng đăng tin thất lạc và tìm thấy thú cưng, hỗ trợ người dùng dễ dàng liên hệ và tìm kiếm.
- Hiển thị thông tin vị trí thú cưng thất lạc trên bản đồ, giúp gia tăng khả năng tìm kiếm hiệu quả.
- Thử nghiệm và học hỏi thêm về kiến trúc microservices

1.2 Các chức năng chính

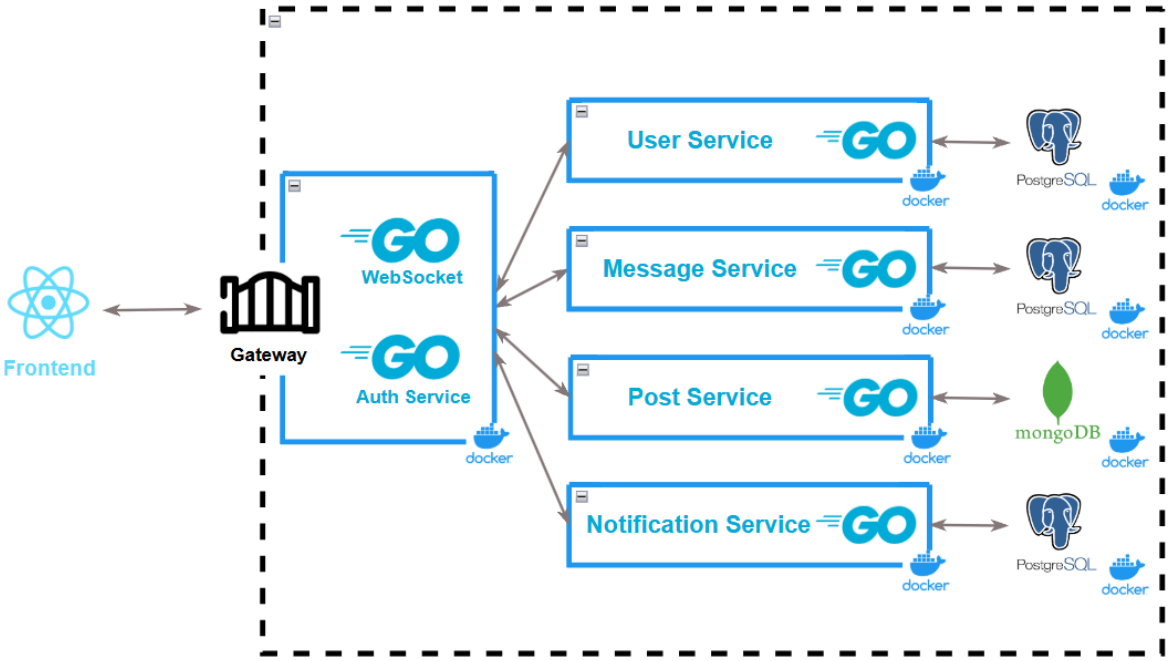


Hình 1.1: Use Case tổng quan

- **Quản lý người dùng**: Đăng ký, đăng nhập, cập nhật thông tin cá nhân.
- **Kết bạn**: Gửi và nhận yêu cầu kết bạn, quản lý danh sách bạn bè.
- **Nhắn tin**: Giao tiếp giữa người dùng với tính năng nhắn tin riêng tư.
- **Đăng bài**: Tạo bài viết blog chia sẻ thông tin, hình ảnh.
- **Thông báo**: Hiển thị, lưu trữ các thông báo để người dùng có thể theo dõi hoạt động trên các bài viết, lời mời kết bạn hay quá trình tìm kiếm thú cưng của mình hoặc người khác.
- **Tin thất lạc - tìm thấy thú cưng**: Cho phép người dùng đăng tin về thú cưng thất lạc hoặc tìm thấy, với vị trí trên bản đồ.
- **Hiển thị bản đồ**: Dùng [4]Leaflet để thể hiện các vị trí thú cưng thất lạc, giúp người dùng dễ dàng tìm kiếm.

1.3 Kiến trúc hệ thống

Hệ thống được lấy cảm hứng từ mô hình microservices chạy riêng biệt trên Docker và giao tiếp từ bên ngoài hay nội bộ đều được xử lý với HTTP, bao gồm các thành phần chính:



Hình 1.2: Kiến trúc hệ thống

- **Backend**: Các dịch vụ nhỏ độc lập được triển khai bằng ngôn ngữ [2]Golang, chịu trách nhiệm xử lý các nghiệp vụ riêng biệt như quản lý người dùng, quản lý bài viết, xử lý tin nhắn, ...
  - **Gateway**: Là nơi trung chuyển các yêu cầu từ Frontend tới các services bên trong.
  - **Auth Service**: Dịch vụ kiểm soát xác thực người dùng, cấp quyền và đăng ký tài khoản. Nó nằm cùng với Gateway
  - **WebSocket Service**: Server websocket cho phép các chức năng như thông báo hay hoạt động có thể thực hiện được, đẩy từ server về lại phía user. Cũng nằm chung với Gateway
  - **User Service**: Dịch vụ quản lý thông tin người dùng cũng như quản lý bạn bè
  - **Message Service**: Dịch vụ quản lý các nhóm chat và các tin nhắn trong các nhóm chat

  - **Post Service**: Dịch vụ quản lý các bài viết, tin thất lạc và các thực thể liên quan như bình luận, tương tác. Nó sử dụng NoSQL thay vì CSDL quan hệ để dễ dàng lấy thông tin rời rạc và lưu trữ linh hoạt hơn.
  - **Notification Service**: Dịch vụ quản lý các thông báo của hệ thống và người dùng

- **Frontend**: Ứng dụng web được xây dựng bằng [1]React, cung cấp giao diện người dùng trực quan, dễ sử dụng.
- **Cơ sở dữ liệu**: Mỗi microservice có thể sử dụng cơ sở dữ liệu riêng biệt phù hợp với nghiệp vụ, có thể là MongoDB hoặc PostgreSQL.
- **API bản đồ**: Tích hợp [3]OpenStreetMap API để lấy dữ liệu vị trí, và sử dụng thư viện [4]Leaflet để hiển thị bản đồ trong giao diện người dùng.

Chương 2

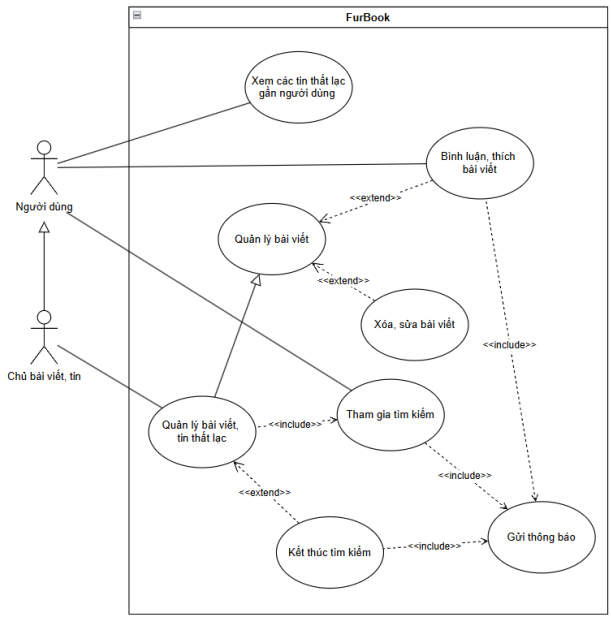
Thiết kế hệ thống

2.1 Mô hình hóa chức năng

Mô hình Use Case giúp xác định các chức năng chính của hệ thống dưới góc nhìn người dùng, miêu tả được các yêu cầu chức năng mà ứng dụng đáp ứng được. Qua đó, thể hiện các tương tác giữa người dùng và hệ thống ứng dụng.

2.1.1 Quản lý bài viết

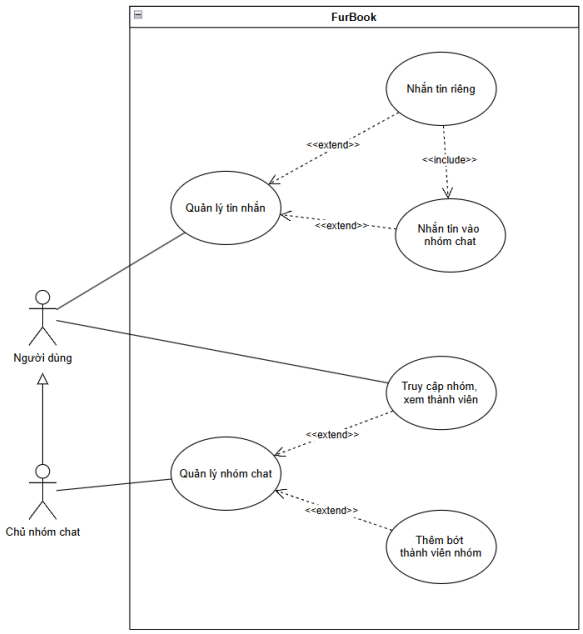
Người dùng có thể thực hiện các thao tác cơ bản của một mạng xã hội trên cả bài viết thông thường và bài viết tin thất lạc. Nhưng với bài viết thất lạc, người dùng có thểcó thể thêm người dùng cùng tham gia tìm kiếm thú cưng và kết thúc quá trình tìm kiếm nếu muốn.



Hình 2.1: Use Case quản lý bài viết

2.1.2 Quản lý nhắn tin

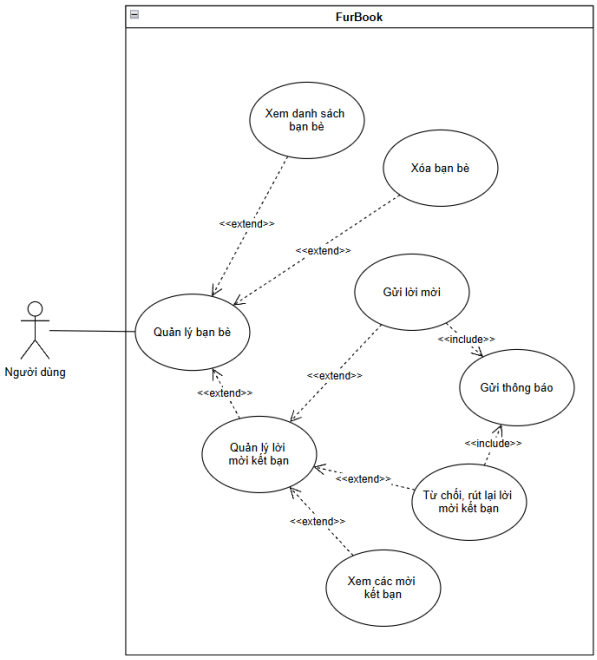
Người dùng có thể nhắn tin riêng hoặc theo nhóm, và thực hiện các thao tác quản lý tin nhắn, nhóm cơ bản.



Hình 2.2: Use Case quản lý tin nhắn

2.1.3 Quản lý bạn bè

Người dùng có thể kết bạn với người dùng khác và theo dõi các bài đăng, tin của họ và nhắn tin cho họ.

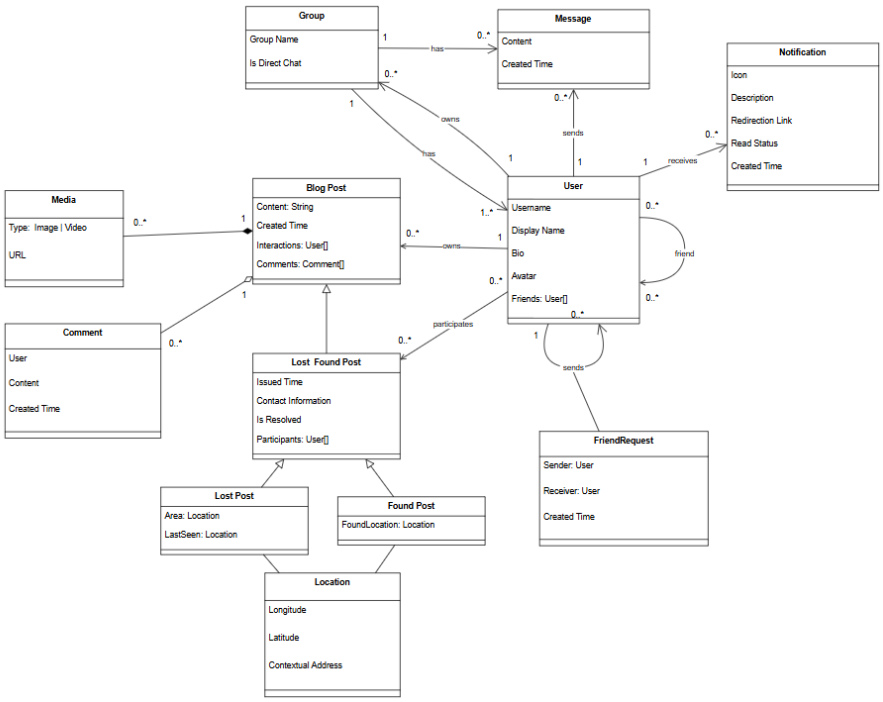


Hình 2.3: Use Case quản lý bạn bè

2.2 Mô hình hóa dữ liệu

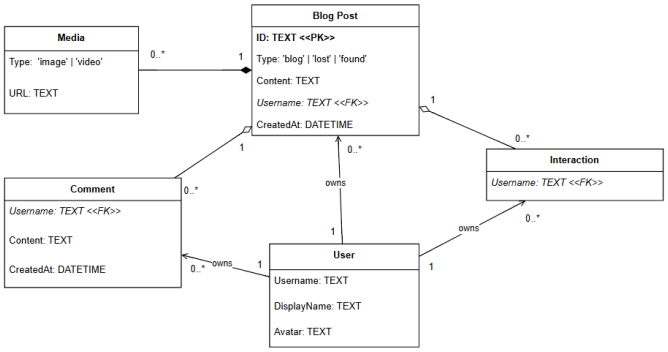
Mô hình lớp (Class Diagram) thể hiện cấu trúc dữ liệu, các lớp đối tượng cùng mối quan hệ, giúp định hướng thiết kế cơ sở dữ liệu và kiến trúc phần mềm.

Song song đó, mô hình đối tượng (Object Diagram) mô tả trạng thái và các đối tượng cụ thể của hệ thống tại một thời điểm, thể hiện sự tương tác thực tế giữa các lớp.

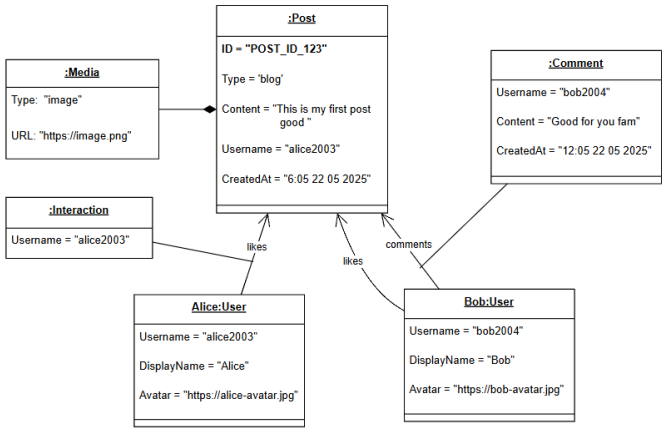


Hình 2.4: Mô hình dữ liệu tổng quát

2.2.1 Quản lý bài viết

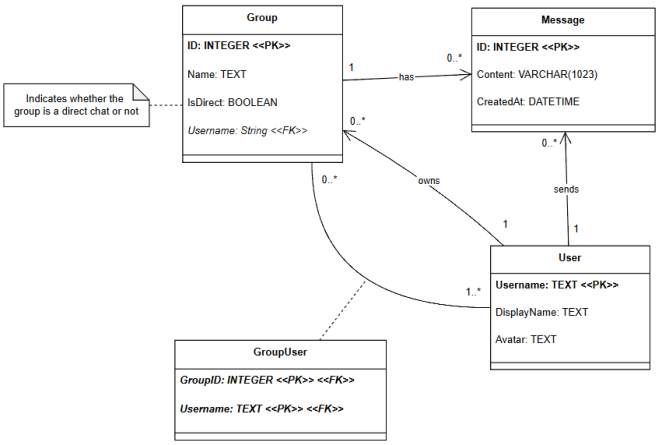


Hình 2.5: Class Diagram quản lý bài viết blog

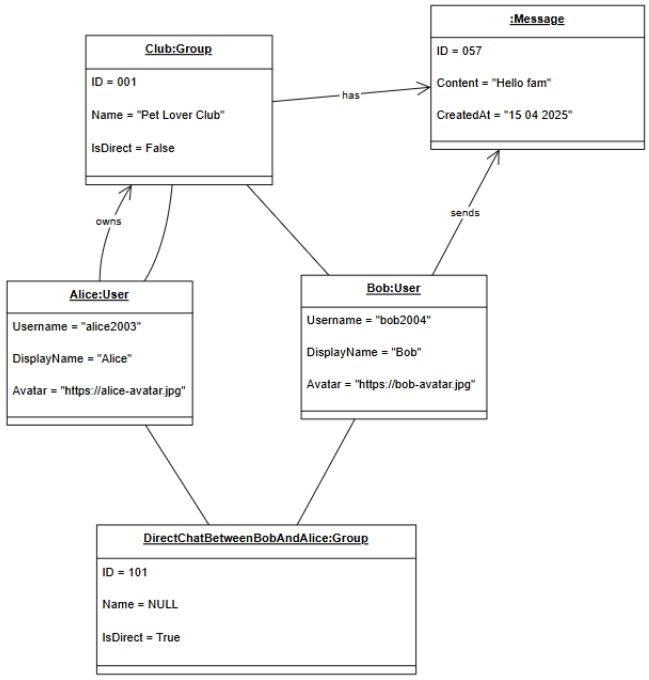


Hình 2.6: Object Diagram quản lý bài viết blog

2.2.2 Quản lý nhắn tin

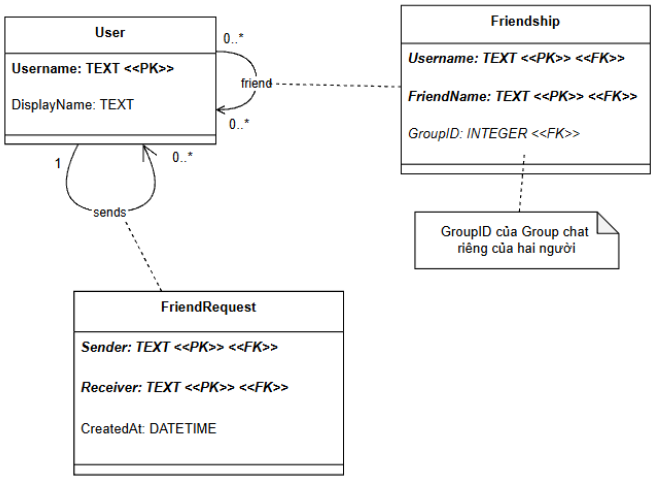


Hình 2.7: Class Diagram quản lý nhắn tin

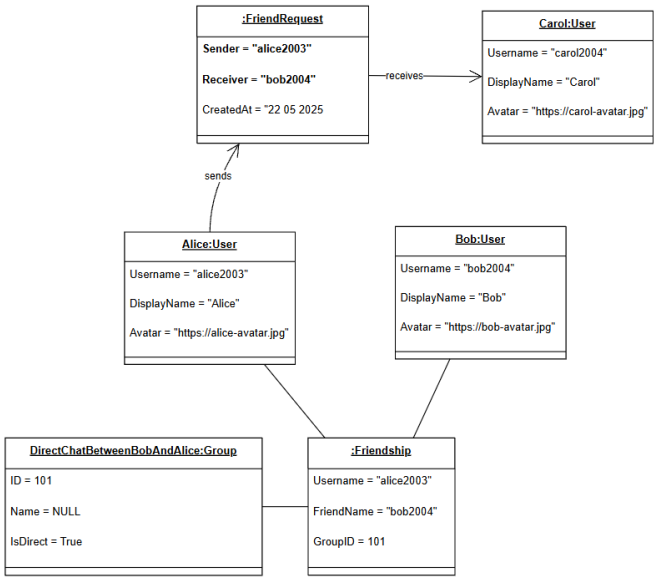


Hình 2.8: Object Diagram quản lý nhắn tin

2.2.3 Quản lý bạn bè

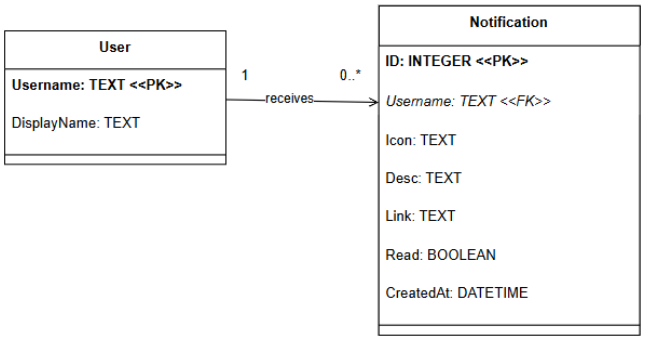


Hình 2.9: Class Diagram quản lý bạn bè

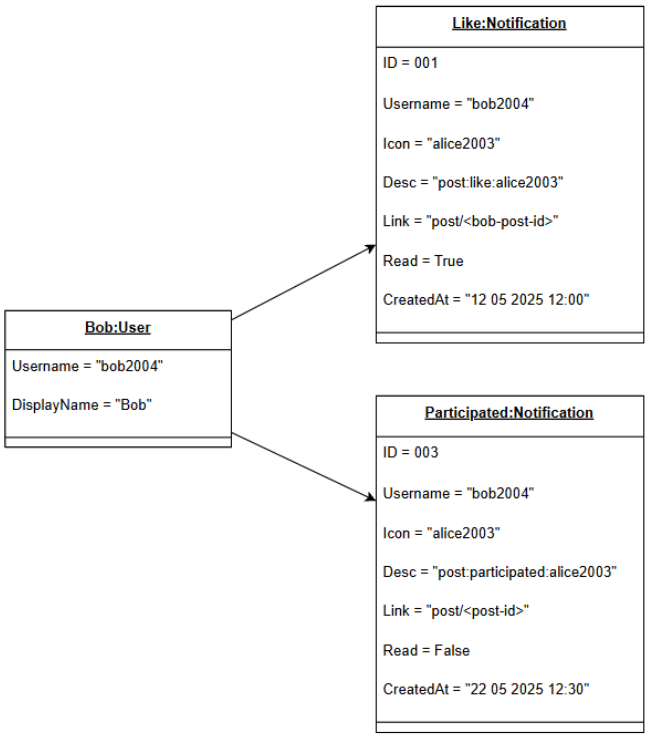


Hình 2.10: Object Diagram quản lý bạn bè

2.2.4 Quản lý thông báo



Hình 2.11: Class Diagram quản lý thông báo



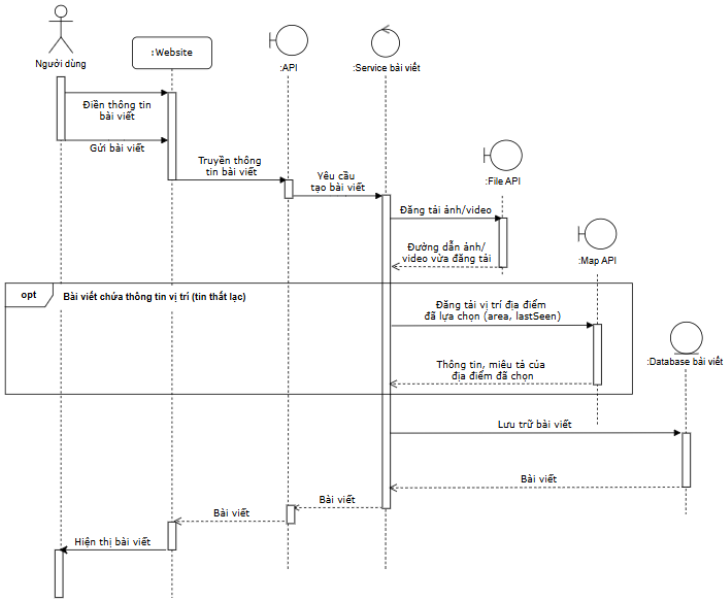
Hình 2.12: Object Diagram quản lý thông báo

2.3 Mô hình hóa hành vi

Mô hình hành vi tập trung mô tả quy trình nghiệp vụ, các bước xử lý logic và luồng dữ liệu trong hệ thống FurBook thông qua sơ đồ hoạt động (Activity Diagram) và sơ đồ luồng hệ thống (System Flow Diagram).

2.3.1 Quản lý bài viết

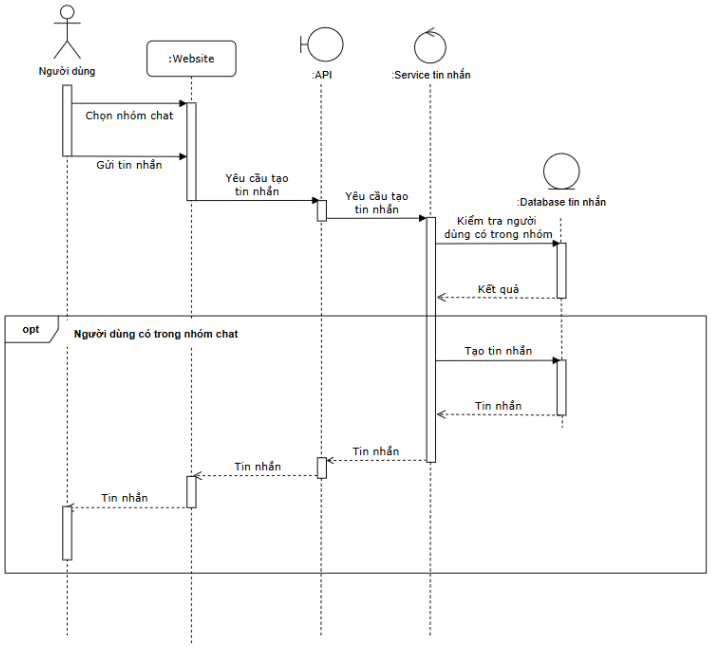
Khi đăng tải bài viết, trước tiên thì hệ thóng sẽ phải tải những file phương tiện lên một file service trước (ở đây gọi tới một api bên ngoài) và lấy link phương tiện để lưu về. Ngoài ra, nếu là vài viết thất lạc thì sẽ dùng thông tin vị trí để lấy mô tả vị trí để lưu về từ một api Map bên ngoài.



Hình 2.13: Luồng đăng tải bài viết

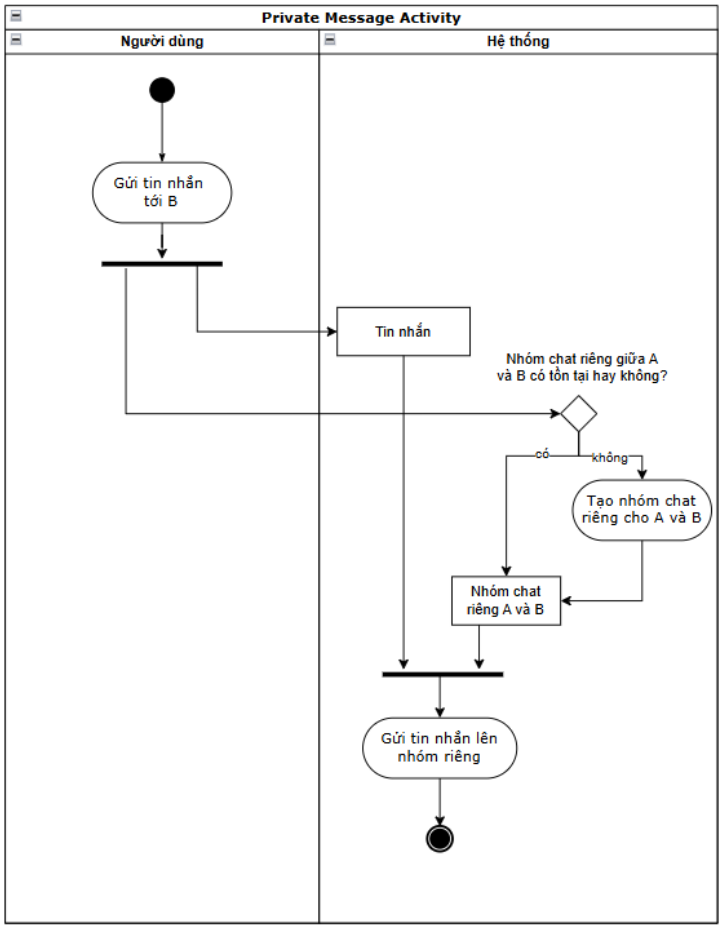
2.3.2 Quản lý nhắn tin

Mỗi tin nhắn sẽ đều thuộc về một nhóm chat, vậy nên trước tiên sẽ phải kiểm tra người dùng có quyền nhắn tin vào nhóm đó hoặc nhóm đó có tồn tại hay không rồi mới tạo tin nhắn. Mặc dù chưa được miêu tả, nhưng tin nhắn sau khi gửi lên Message Service, nếu tạo thành công sẽ chuyển tiếp cho Web Socket Service để kiểm tra người dùng đó hiện có đang kết nối không để gửi trực tiếp tin nhắn cho họ.



Hình 2.14: Luồng nhắn tin

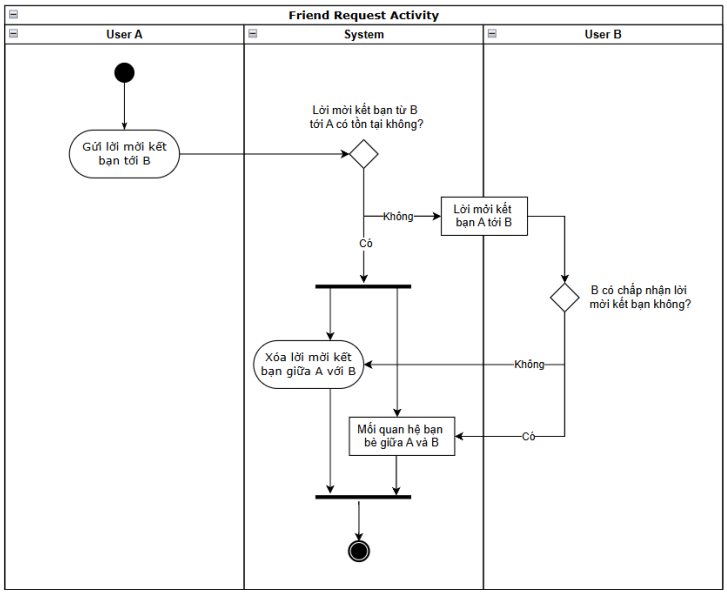
Kể cả là nhắn tin riêng cũng cần có nhóm chat nên hệ thống sẽ kiểm tra có nhóm chat riêng giữa 2 người chưa, nếu chưa thì tạo mới và gửi tin nhắn vào.



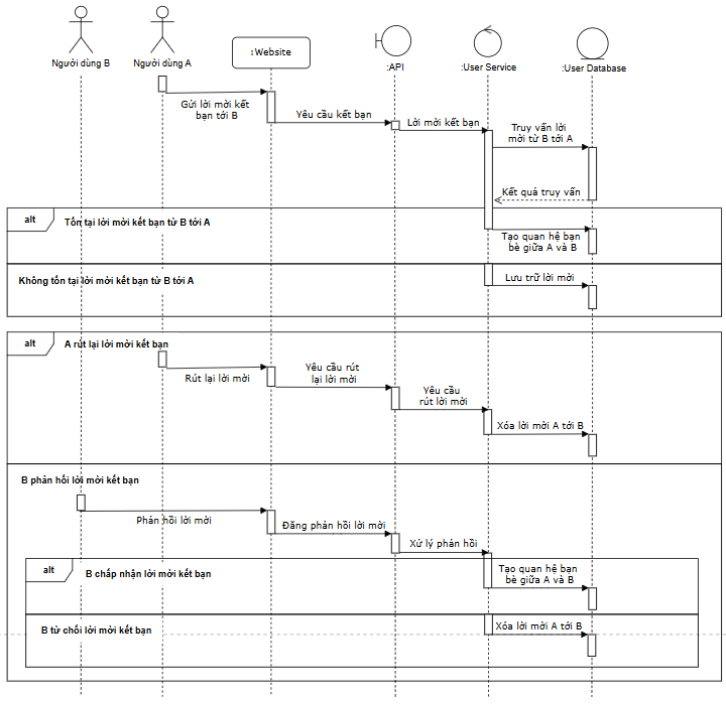
Hình 2.15: Hoạt động nhắn tin riêng

2.3.3 Quản lý bạn bè

Khi gửi lời mời kết bạn tới cho một người dùng, trước tiên hệ thống sẽ kiểm tra nếu lời mời kết bạn từ người dùng đó tới bạn có tồn tại không đã, nếu có thì cả hai sẽ lập tức trở thành bạn không thì lưu lời mời kết bạn vào CSDL. Hoạt động này giúp tránh xung đột và việc dữ liệu bị sót lại trong CSDL.



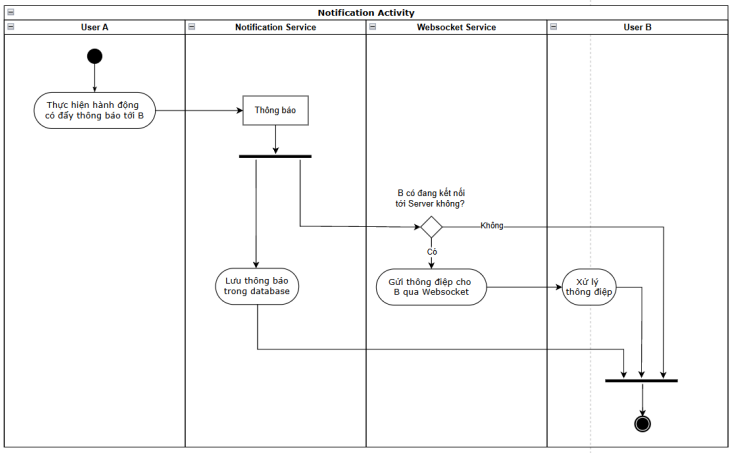
Hình 2.16: Hoạt động gửi lời mời kết bạn



Hình 2.17: Luồng gửi lời mời kết bạn

2.3.4 Quản lý thông báo

Khi một yêu cầu tạo thông báo được gửi lên Service (có thể là từ hoạt động thích, comment hay tham gia tìm kiếm) thì sẽ được tạo ở cả CSDL và chuyển tiếp qua cho Web Socket Service để đẩy thông báo cho người dùng nếu họ đang kết nối trên giao diện với hệ thống.



Hình 2.18: Hoạt động gửi, nhận thông báo

Chương 3

Cài đặt

Link dự án

`https://github.com/hungq1205/furbook`

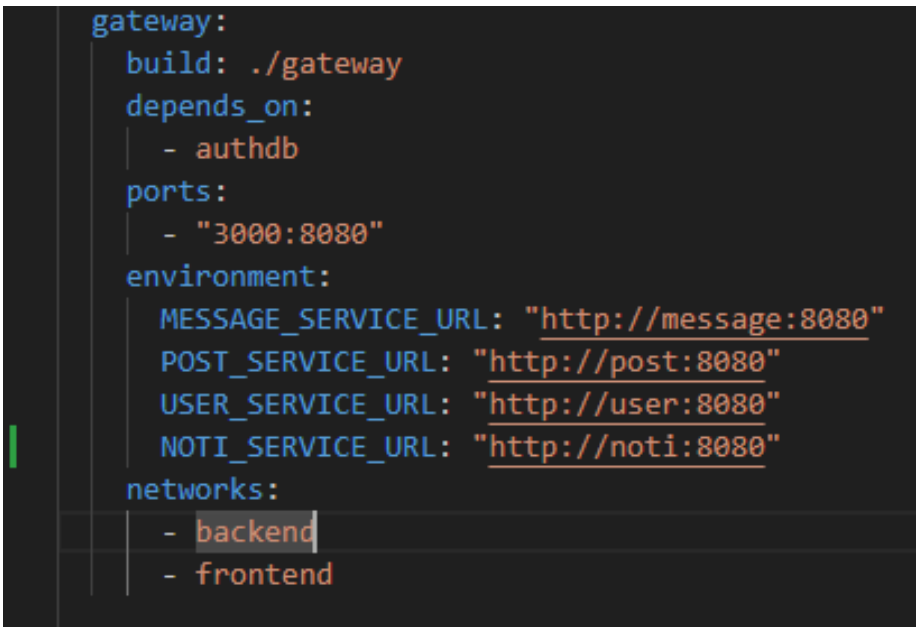
3.1 API Gateway

Để mô phỏng chức năng của một API Gateway, proxy được sử dụng để điều hướng các yêu cầu tới service phù hợp dữa trên đường dẫn của nó.

```go
func MakeGatewayHandler(app *gin.Engine) {
    group := app.Group("")
    group.Use(internal.AuthMiddleware())

    group.Any("/api/message", ProxyTo(messageServiceURL))
    group.Any("/api/message/*path", ProxyTo(messageServiceURL))

    group.Any("/api/group", ProxyTo(messageServiceURL))
    group.Any("/api/group/*path", ProxyTo(messageServiceURL))

    group.Any("/api/post", ProxyTo(postServiceURL))
    group.Any("/api/post/*path", ProxyTo(postServiceURL))

    group.Any("/api/user", ProxyTo(userServiceURL))
    group.Any("/api/user/*path", ProxyTo(userServiceURL))

    group.Any("/api/noti", ProxyTo(notiServiceURL))
    group.Any("/api/noti/*path", ProxyTo(notiServiceURL))
}
```

Hình 3.1: Enter Caption

Trong file Docker compose, các đường dẫn cụ thể tới các service được khai báo theo tên và sẽ được dịch ra đường dẫn cụ thể bởi Docker.



Hình 3.2: Enter Caption

3.2 Authentication Service

Authentication Service sẽ nằm chung một domain với gateway để yêu cầu không cần phải chuyển tiếp lần nữa cho các tác vụ sử dụng nhiều.

Người dùng khi lập tài khoản sẽ được gán cho salt riêng và lưu vào CSDL, cản trở việc dò mật khẩu của đối tượng muốn chiếm quyền sử dụng tài khoản. Mật khẩu sau khi thêm salt sẽ được lưu dưới dạng Hash trong CSDL, vậy nên quản trị viên cũng không thể biết dược mật khẩu.

```go
func (r *Repository) Authenticate(username, password string) (bool, error) {
    var user User
    err := r.db.Where("username = ?", username).First(&user).Error
    if errors.Is(err, gorm.ErrRecordNotFound) {
        return false, nil
    }
    if err != nil {
        return false, err
    }
    return internal.CompareHashAndPassword(user.PasswordHashed, password, user.Salt), nil
}

func (r *Repository) CreateUser(username, password string) error {
    salt := internal.GenerateSalt()
    hashedPassword := internal.Hash(password, salt)
    return r.db.Create(&User{Username: username, PasswordHashed: hashedPassword, Salt: salt}).Error
}
```

Hình 3.3: Cơ chế Hash

Sau khi xác thực thì hệ thống sẽ trả về JWT Token cho người dùng để lưu thông tin xác thực, có thể dùng cho các lời gọi tiếp theo.

```go
func GenerateJwt(username string) (string, error) {
    token := jwt.NewWithClaims(jwt.SigningMethodHS256, jwt.MapClaims{
        "username": username,
    })
    return token.SignedString(secretKey)
}

func ParseJwt(tokenStr string) (string, error) {
    token, err := jwt.Parse(tokenStr, func(token *jwt.Token) (interface{}, error) {
        return secretKey, nil
    })
    if err != nil {
        return "", err
    }
    claims := token.Claims.(jwt.MapClaims)
    username := claims["username"].(string)
    return username, nil
}
```

Hình 3.4: Sử dụng JWT

3.3 WebSocket Service

WebSocket từ người dùng sẽ được kết nối qua đường dẫn /ws còn các đường dẫn còn lại được sử dụng bởi các service bên trong như thông báo và nhắn tin để gửi trực tiếp thông điệp cho người dùng đang kết nối.

```
func MakeHandler(app *gin.Engine, groupClient client.GroupClient) {
    app.GET("/ws", func(c *gin.Context) {
        handleWebsocket(c, groupClient)
    })

    app.POST("/ws/message", func(c *gin.Context) {
        handleChatMessage(c, groupClient)
    })

    app.POST("/ws/noti", func(c *gin.Context) {
        handleNotificationMessage(c)
    })
}

func handleChatMessage(c *gin.Context, groupClient client.GroupClient) {···
}

func handleNotificationMessage(c *gin.Context) {···
}

func handleWebsocket(c *gin.Context, groupClient client.GroupClient) {···
}
```

Hình 3.5: Controller của WebSocket Service

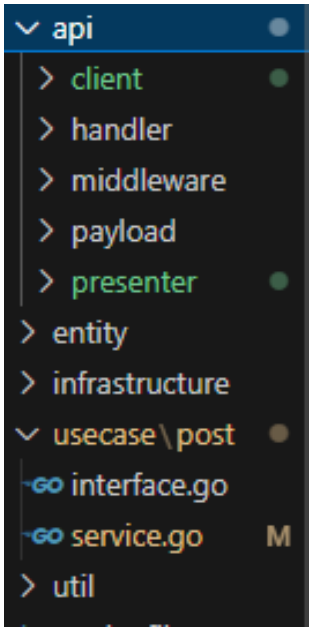
Các thông điệp qua WebSocket được thống nhất theo định dạng của Message như trong ảnh và phần data sẽ được đóng gói, frontend sẽ xử lý ép kiểu và sử dụng dữ liệu hợp lý.

```go
type Message struct {
    Type    MessageType     `json:"type"`
    Payload json.RawMessage `json:"payload"`
}

type AuthPayload struct {
    Token string `json:"token"`
}

type ChatPayload struct {
    MessageID int       `json:"messageId"`
    Username  string    `json:"username"`
    GroupID   int       `json:"groupId"`
    Content   string    `json:"content"`
    CreatedAt time.Time `json:"createdAt"`
}

type NotificationPayload struct {
    ID        int       `json:"id"`
    Username  string    `json:"username"`
    Icon      string    `json:"icon"`
    Desc      string    `json:"desc"`
    Link      string    `json:"link"`
    Read      bool      `json:"read"`
    CreatedAt time.Time `json:"created_at"`
}
```

Hình 3.6: Định dạng thông điệp WebSocket

3.4 User, Post, Message, Notification Service

Các services còn lại như user, post, message, notification sẽ được triển khai như thông thường, và tôi cố gắng triển theo kiến trúc layer. Với Code base như sau:



Hình 3.7: Enter Caption

- **client**: Chứa các logic để xử lý lời gói tới và trở về từ các serivces nội bộ khác
- **handler**: Chứa logic của Controller và chuyển đổi các thực thể hệ thống sang các thực thể biểu diễn tương ứng và ngược lại (như DTO).
- **middleware**: Chứa các middleware cụ thể, chủ yếu là middleware cho việc xác thực.
- **payload, presenter**: Định nghĩa các thực thể được sử dụng giao tiếp với frontend hay xử lý trên layer gần controller (như DTO)
- **entity**: Định nghĩa thực thể lưu trong hệ thống và chứa logic cho GORM (thư viện ORM cho golang)
- **infrastructure**: Chứa logic để backend giao tiếp với hệ thống lưu trữ, ví dụ như database. Có thể ví như DAO, DAM hay Repository.
- **usecase**: Triển khai các logic nghiệp vụ được định nghĩa ở phần thiết kế
- **util**: Các logic bổ trợ

Ví dụ một file trong usecase được định nghĩa theo usecase như sau. Nó sẽ định nghĩa các logic cần có để đấp ứng chức năng trong phần thiết kế và sẽ có lớp triển khai những logic đó.

```
type UseCase interface {
    GetPost(ctx context.Context, id string) (*entity.Post, error)
    GetNearLostPosts(ctx context.Context, latitude float64, longitude float64) ([]*entity.Post, error)
    GetPostsOfUser(ctx context.Context, username string, pagination util.Pagination) ([]*entity.Post, error)
    GetPostsOfUsers(ctx context.Context, usernames []string, pagination util.Pagination) ([]*entity.Post, error)
    CheckOwnership(ctx context.Context, username, postId string) (bool, error)
    CreateBlogPost(ctx context.Context, username, content string, medias []entity.Media) (*entity.Post, error)
    CreateLostPetPost(ctx context.Context, username, contactInfo string, postType entity.PostType, content string
    PatchContent(ctx context.Context, id, content string, medias []entity.Media) (*entity.Post, error)
    PatchFound(ctx context.Context, id string, found bool) error
    DeletePost(ctx context.Context, id string) error

    CreateComment(ctx context.Context, postId, username, content string) error
    DeleteComment(ctx context.Context, postId, username string) error
    GetComments(ctx context.Context, postId string) ([]entity.Comment, error)

    UpsertInteraction(ctx context.Context, postId, username string, itype entity.InteractionType) error
    DeleteInteraction(ctx context.Context, postId, username string, itype entity.InteractionType) error
}
```

Hình 3.8: Ví dụ file thủ tục cho file triển khai logic nghiệp vụ

Ví dụ một file trong handler được viết như sau. Nó sẽ định nghĩa các logic cho hoạt động của API để service khác và frontend có thể giao tiếp

```
func MakeHandler(app *gin.Engine, postService *post.Service, userClient client.UserClient) {
    postGroup := app.Group("/api/post")
    {
        postGroup.GET("/:postID", func(c *gin.Context) {
            GetPost(c, postService, userClient)
        })

        postGroup.GET("/lost", func(c *gin.Context) {
            GetNearLostPosts(c, postService, userClient)
        })

        postGroup.GET("/ofUser/:username", func(c *gin.Context) {
            GetPostsOfUser(c, postService, userClient)
        })

        postGroup.POST("/ofUsers", func(c *gin.Context) {
            GetPostsOfUsers(c, postService, userClient)
        })

        postGroup.GET("/:postID/comments", func(c *gin.Context) {
            GetComments(c, postService, userClient)
        })

        authGroup := postGroup.Group("", middleware.MustAuthorizeMiddleware())

        authGroup.POST("/blog", func(c *gin.Context) {
            CreateBlogPost(c, postService)
        })

        authGroup.POST("/lost", func(c *gin.Context) {
            CreateLostPetPost(c, postService, userClient)
        })

        authGroup.PATCH("/:postID/content", func(c *gin.Context) {
            PatchContentPost(c, postService)
        })
```

Hình 3.9: Ví dụ một Controller bên trong handler

Chương 4

Kết luận

Dự án FurBook là cơ hội giúp em áp dụng và củng cố kiến thức về lập trình web. Qua dự án, nhóm đã học được cách xây dựng hệ thống microservices với Golang, luyện tập sử dụng Docker và Kubernetes , phát triển giao diện người dùng bằng React, tích hợp bản đồ với Leaflet và OpenStreetMap, cũng như triển khai và đánh giá một hệ thống thực tế. Những kinh nghiệm này là nền tảng vững chắc cho việc phát triển các ứng dụng web.

Tài liệu tham khảo

[1] ReactJS Documentation, https://reactjs.org/docs/getting-started.html

[2] Go Programming Language, https://golang.org/doc/

[3] OpenStreetMap, https://www.openstreetmap.org/about

[4] Leaflet.js - An open-source JavaScript library for interactive maps, https://leafletjs.com/

[5] Microservices Architecture - Martin Fowler, https://martinfowler.com/articles/microservices.html