ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення

ЛАБОРАТОРНА РОБОТА №4

з дисципліни «Веб-технології»

.(назва дисципліни)

на тему: «Створення Backend. Post сервіс»

ЛРІПЗс.2401075.04.07.ЛР

(шифр, назва)

Рівень вищої освіти                     Перший (бакалаврський)

Галузь знань                      12 «Інформаційні технології»

Спеціальність     121 «Інженерія програмного забезпечення»

Освітня програма   «Інженерія програмного забезпечення»

Студента. І  курсу, група . ІПЗс-24. \_\_\_\_\_\_\_\_\_\_\_ М.О. Лапко.

            (шифр)                         (підпис)     (ініціали, прізвище)

Викладач канд. техн. наук, доцент  .   \_\_\_\_\_\_\_\_\_\_\_ О.М. Яшина

              (посада, вчене звання, науковий ступінь)         (підпис)   (ініціали, прізвище)

Кількість балів \_\_\_\_\_\_\_

Оцінка за шкалою:

національною\_\_\_\_\_\_\_\_\_/ЄКТС\_\_\_\_

Хмельницький, 2024

ЗВІТ

**Варіант 7**

**Мета:** Розробити та впровадити сервіс постів (Post сервіс) для забезпечення постінгу контента користувачами. Реалізувати CRUD функціонал, repository layer, service layer та transport layer з rest та grpc для коректної взаємодії з іншими мікросервісами.

**ХІД РОБОТИ**

**Завдання №1.** Підготовка проекту

Налаштування структури проекту за шаблоном cmd, internal, pkg, розділення на відповідні папки для Post сервісу.

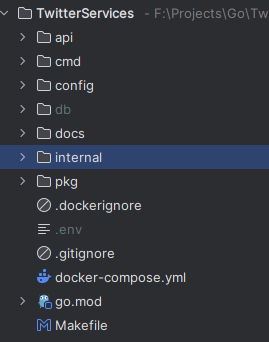


Рисунок 1 — Структура проекту

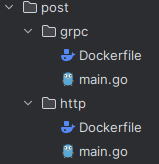


Рисунок 2 — Структура папки post в cmd

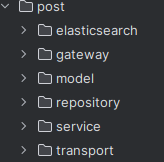


Рисунок 3 — Структура папки post в internal

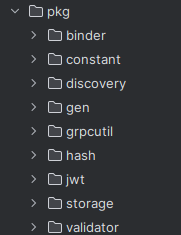


Рисунок 4 — Структура папки pkg

Додавання необхідних залежностей для роботи з базою даних та транспортування даних через REST і gRPC (наприклад, gorm.io/gorm, та інші).

**Завдання №2.** Реалізація repository layer

Створення моделей структур User, Post, Like, Comment, Tag

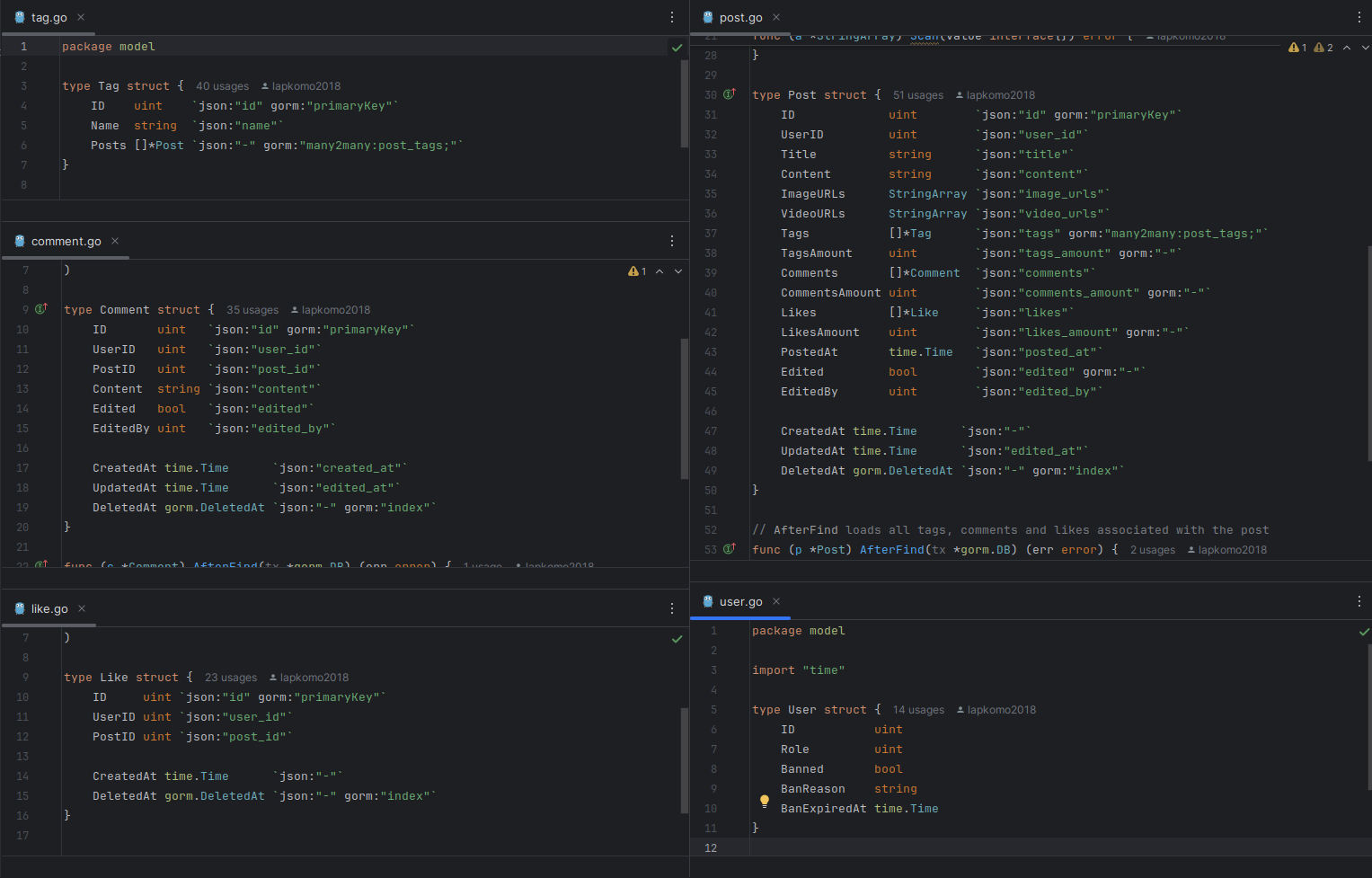


Рисунок 5 — Моделі структур User, Post, Like, Comment, Tag

Код для парсингу env виконуєтся з допомогою пакета github.com/caarlos0/env/v6 в функції init() яка визивается автоматично при запуску сервісу:



Створення repository layer для взаємодії з базою даних (MySQL).

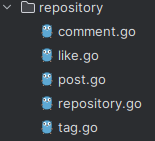


Рисунок 6 — Структура папки repository

Код comment.go:

|  |
| --- |
| package repository  import (  "log"   "github.com/SocialNetworkY/Backend/internal/post/model"  "gorm.io/gorm" )  type (  CommentSearch interface {  Index(comment \*model.Comment) error  Delete(id uint) error  Search(query string, skip, limit int) ([]uint, error)  }   CommentRepository struct {  db \*gorm.DB  s CommentSearch  } )  func NewCommentRepository(db \*gorm.DB, s CommentSearch) \*CommentRepository {  return &CommentRepository{  db: db,  s: s,  } }  func (cr \*CommentRepository) Add(comment \*model.Comment) error {  log.Printf("Adding comment: %v\n", comment)  if err := cr.db.Create(comment).Error; err != nil {  log.Printf("Error adding comment: %v\n", err)  return err  }  if err := cr.s.Index(comment); err != nil {  log.Printf("Error indexing comment: %v\n", err)  return err  }  log.Printf("Comment added successfully: %v\n", comment)  return nil }  func (cr \*CommentRepository) Save(comment \*model.Comment) error {  log.Printf("Saving comment: %v\n", comment)  if err := cr.db.Save(comment).Error; err != nil {  log.Printf("Error saving comment: %v\n", err)  return err  }  if err := cr.s.Index(comment); err != nil {  log.Printf("Error indexing comment: %v\n", err)  return err  }  log.Printf("Comment saved successfully: %v\n", comment)  return nil }  func (cr \*CommentRepository) Delete(id uint) error {  log.Printf("Deleting comment with ID: %d\n", id)  if err := cr.db.Delete(&model.Comment{ID: id}).Error; err != nil {  log.Printf("Error deleting comment: %v\n", err)  return err  }  if err := cr.s.Delete(id); err != nil {  log.Printf("Error deleting comment from search index: %v\n", err)  return err  }  log.Printf("Comment deleted successfully: %d\n", id)  return nil }  func (cr \*CommentRepository) Find(id uint) (\*model.Comment, error) {  log.Printf("Finding comment with ID: %d\n", id)  comment := &model.Comment{}  if err := cr.db.First(comment, id).Error; err != nil {  log.Printf("Error finding comment: %v\n", err)  return nil, err  }  log.Printf("Comment found: %v\n", comment)  return comment, nil }  func (cr \*CommentRepository) FindSome(skip, limit int) ([]\*model.Comment, error) {  log.Printf("Finding some comments with skip: %d and limit: %d\n", skip, limit)  var comments []\*model.Comment  if limit < 0 {  skip = -1  }  if err := cr.db.Offset(skip).Limit(limit).Find(&comments).Error; err != nil {  log.Printf("Error finding some comments: %v\n", err)  return nil, err  }  log.Printf("Found some comments: %v\n", comments)  return comments, nil }  func (cr \*CommentRepository) FindByPost(postID uint, skip, limit int) ([]\*model.Comment, error) {  log.Printf("Finding comments for post with ID: %d, skip: %d, limit: %d\n", postID, skip, limit)  var comments []\*model.Comment  if limit < 0 {  skip = -1  }  if err := cr.db.Offset(skip).Where("post\_id = ?", postID).Limit(limit).Find(&comments).Error; err != nil {  log.Printf("Error finding comments: %v\n", err)  return nil, err  }  log.Printf("Comments found: %v\n", comments)  return comments, nil }  func (cr \*CommentRepository) FindByUser(userID uint, skip, limit int) ([]\*model.Comment, error) {  log.Printf("Finding comments by user with ID: %d, skip: %d, limit: %d\n", userID, skip, limit)  var comments []\*model.Comment  if limit < 0 {  skip = -1  }  if err := cr.db.Offset(skip).Where("user\_id = ?", userID).Limit(limit).Find(&comments).Error; err != nil {  log.Printf("Error finding comments: %v\n", err)  return nil, err  }  log.Printf("Comments found: %v\n", comments)  return comments, nil }  func (cr \*CommentRepository) Search(query string, skip, limit int) ([]\*model.Comment, error) {  log.Printf("Searching comments with query: %s, skip: %d, limit: %d\n", query, skip, limit)  ids, err := cr.s.Search(query, skip, limit)  if err != nil {  log.Printf("Error searching comments: %v\n", err)  return nil, err  }  var comments []\*model.Comment  if err := cr.db.Where("id IN ?", ids).Find(&comments).Error; err != nil {  log.Printf("Error finding comments by IDs: %v\n", err)  return nil, err  }  log.Printf("Comments found: %v\n", comments)  return comments, nil } |

Код like.go:

|  |
| --- |
| package repository  import (  "log"   "github.com/SocialNetworkY/Backend/internal/post/model"  "gorm.io/gorm" )  type LikeRepository struct {  db \*gorm.DB }  func NewLikeRepository(db \*gorm.DB) \*LikeRepository {  return &LikeRepository{  db: db,  } }  func (lr \*LikeRepository) Add(like \*model.Like) error {  log.Printf("Adding like: %v\n", like)  if err := lr.db.Create(like).Error; err != nil {  log.Printf("Error adding like: %v\n", err)  return err  }  log.Printf("Like added successfully: %v\n", like)  return nil }  func (lr \*LikeRepository) Save(like \*model.Like) error {  log.Printf("Saving like: %v\n", like)  if err := lr.db.Save(like).Error; err != nil {  log.Printf("Error saving like: %v\n", err)  return err  }  log.Printf("Like saved successfully: %v\n", like)  return nil }  func (lr \*LikeRepository) Delete(id uint) error {  log.Printf("Deleting like with ID: %d\n", id)  if err := lr.db.Delete(&model.Like{ID: id}).Error; err != nil {  log.Printf("Error deleting like: %v\n", err)  return err  }  log.Printf("Like deleted successfully: %d\n", id)  return nil }  func (lr \*LikeRepository) Find(id uint) (\*model.Like, error) {  log.Printf("Finding like with ID: %d\n", id)  like := &model.Like{}  if err := lr.db.First(like, id).Error; err != nil {  log.Printf("Error finding like: %v\n", err)  return nil, err  }  log.Printf("Like found: %v\n", like)  return like, nil }  func (lr \*LikeRepository) FindSome(skip, limit int) ([]\*model.Like, error) {  log.Printf("Finding some likes with skip: %d and limit: %d\n", skip, limit)  var likes []\*model.Like  if limit < 0 {  skip = -1  }  if err := lr.db.Offset(skip).Limit(limit).Find(&likes).Error; err != nil {  log.Printf("Error finding some likes: %v\n", err)  return nil, err  }  log.Printf("Found some likes: %v\n", likes)  return likes, nil }  func (lr \*LikeRepository) FindByPost(postID uint, skip, limit int) ([]\*model.Like, error) {  log.Printf("Finding likes for post with ID: %d, skip: %d, limit: %d\n", postID, skip, limit)  var likes []\*model.Like  if limit < 0 {  skip = -1  }  if err := lr.db.Offset(skip).Where("post\_id = ?", postID).Limit(limit).Find(&likes).Error; err != nil {  log.Printf("Error finding likes: %v\n", err)  return nil, err  }  log.Printf("Likes found: %v\n", likes)  return likes, nil }  func (lr \*LikeRepository) FindByUser(userID uint, skip, limit int) ([]\*model.Like, error) {  log.Printf("Finding likes by user with ID: %d, skip: %d, limit: %d\n", userID, skip, limit)  var likes []\*model.Like  if limit < 0 {  skip = -1  }  if err := lr.db.Offset(skip).Where("user\_id = ?", userID).Limit(limit).Find(&likes).Error; err != nil {  log.Printf("Error finding likes: %v\n", err)  return nil, err  }  log.Printf("Likes found: %v\n", likes)  return likes, nil }  func (lr \*LikeRepository) FindByPostUser(postID, userID uint) (\*model.Like, error) {  log.Printf("Finding like for post with ID: %d by user with ID: %d\n", postID, userID)  var like \*model.Like  if err := lr.db.Where("post\_id = ? AND user\_id = ?", postID, userID).First(&like).Error; err != nil {  log.Printf("Error finding like: %v\n", err)  return nil, err  }  log.Printf("Like found: %v\n", like)  return like, nil } |

Код post.go:

|  |
| --- |
| package repository  import (  "log"   "github.com/SocialNetworkY/Backend/internal/post/model"  "gorm.io/gorm" )  type (  PostSearch interface {  Index(post \*model.Post) error  Delete(id uint) error  Search(query string, skip, limit int) ([]uint, error)  }   PostRepository struct {  db \*gorm.DB  s PostSearch  } )  func NewPostRepository(db \*gorm.DB, s PostSearch) \*PostRepository {  return &PostRepository{  db: db,  s: s,  } }  func (pr \*PostRepository) Add(post \*model.Post) error {  log.Printf("Adding post: %v\n", post)  if err := pr.db.Create(post).Error; err != nil {  log.Printf("Error adding post: %v\n", err)  return err  }  if err := pr.s.Index(post); err != nil {  log.Printf("Error indexing post: %v\n", err)  return err  }  log.Printf("Post added successfully: %v\n", post)  return nil }  func (pr \*PostRepository) Save(post \*model.Post) error {  log.Printf("Saving post: %v\n", post)  if err := pr.db.Save(post).Error; err != nil {  log.Printf("Error saving post: %v\n", err)  return err  }  if err := pr.s.Index(post); err != nil {  log.Printf("Error indexing post: %v\n", err)  return err  }  log.Printf("Post saved successfully: %v\n", post)  return nil }  func (pr \*PostRepository) Delete(id uint) error {  log.Printf("Deleting post: %d\n", id)  if err := pr.db.Delete(&model.Post{ID: id}).Error; err != nil {  log.Printf("Error deleting post: %v\n", err)  return err  }  if err := pr.s.Delete(id); err != nil {  log.Printf("Error deleting post from search index: %v\n", err)  return err  }  log.Printf("Post deleted successfully: %d\n", id)  return nil }  func (pr \*PostRepository) Find(id uint) (\*model.Post, error) {  log.Printf("Finding post: %d\n", id)  post := &model.Post{}  if err := pr.db.Preload("Tags").Preload("Comments").Preload("Likes").First(post, id).Error; err != nil {  log.Printf("Error finding post: %v\n", err)  return nil, err  }  log.Printf("Post found: %v\n", post)  return post, nil }  func (pr \*PostRepository) FindSome(skip, limit int) ([]\*model.Post, error) {  log.Printf("Finding some posts with skip: %d and limit: %d\n", skip, limit)  var posts []\*model.Post  if limit < 0 {  skip = -1  }  if err := pr.db.Preload("Tags").Preload("Comments").Preload("Likes").Offset(skip).Limit(limit).Find(&posts).Error; err != nil {  log.Printf("Error finding some posts: %v\n", err)  return nil, err  }  log.Printf("Found some posts: %v\n", posts)  return posts, nil }  func (pr \*PostRepository) FindByUser(userID uint, skip, limit int) ([]\*model.Post, error) {  log.Printf("Finding posts by user: %d with skip: %d and limit: %d\n", userID, skip, limit)  var posts []\*model.Post  if limit < 0 {  skip = -1  }  if err := pr.db.Preload("Tags").Preload("Comments").Preload("Likes").Where("user\_id = ?", userID).Offset(skip).Limit(limit).Find(&posts).Error; err != nil {  log.Printf("Error finding posts by user: %v\n", err)  return nil, err  }  log.Printf("Found posts by user: %v\n", posts)  return posts, nil }  func (pr \*PostRepository) FindByTag(tagID uint, skip, limit int) ([]\*model.Post, error) {  log.Printf("Finding posts by tag: %d with skip: %d and limit: %d\n", tagID, skip, limit)  var posts []\*model.Post  if limit < 0 {  skip = -1  }  if err := pr.db.Preload("Tags").Preload("Comments").Preload("Likes").Joins("JOIN post\_tags ON post\_tags.post\_id = posts.id").Where("post\_tags.tag\_id = ?", tagID).Offset(skip).Limit(limit).Find(&posts).Error; err != nil {  log.Printf("Error finding posts by tag: %v\n", err)  return nil, err  }  log.Printf("Found posts by tag: %v\n", posts)  return posts, nil }  func (pr \*PostRepository) ClearAssociations(postID uint) error {  log.Printf("Clearing associations of post: %d\n", postID)  if err := pr.db.Model(&model.Post{ID: postID}).Association("Tags").Clear(); err != nil {  log.Printf("Error clearing associations: %v\n", err)  return err  }  log.Printf("Associations cleared for post: %d\n", postID)  return nil }  func (pr \*PostRepository) AddTag(postID, tagID uint) error {  log.Printf("Adding tag: %d to post: %d\n", tagID, postID)  if err := pr.db.Model(&model.Post{ID: postID}).Association("Tags").Append(&model.Tag{ID: tagID}); err != nil {  log.Printf("Error adding tag: %v\n", err)  return err  }  log.Printf("Tag added to post: %d\n", postID)  return nil }  func (pr \*PostRepository) RemoveTag(postID, tagID uint) error {  log.Printf("Removing tag: %d from post: %d\n", tagID, postID)  if err := pr.db.Model(&model.Post{ID: postID}).Association("Tags").Delete(&model.Tag{ID: tagID}); err != nil {  log.Printf("Error removing tag: %v\n", err)  return err  }  log.Printf("Tag removed from post: %d\n", postID)  return nil }  func (pr \*PostRepository) Search(query string, skip, limit int) ([]\*model.Post, error) {  log.Printf("Searching posts by query: %s with skip: %d and limit: %d\n", query, skip, limit)  postIDs, err := pr.s.Search(query, skip, limit)  if err != nil {  log.Printf("Error searching posts: %v\n", err)  return nil, err  }   var posts []\*model.Post  if err := pr.db.Preload("Tags").Preload("Comments").Preload("Likes").Where("id IN ?", postIDs).Find(&posts).Error; err != nil {  log.Printf("Error finding posts by IDs: %v\n", err)  return nil, err  }  log.Printf("Found posts by query: %v\n", posts)  return posts, nil } |

Код tag.go:

|  |
| --- |
| package repository  import (  "log"   "github.com/SocialNetworkY/Backend/internal/post/model"  "gorm.io/gorm" )  type (  TagSearch interface {  Index(tag \*model.Tag) error  Delete(id uint) error  Search(query string, skip, limit int) ([]uint, error)  }  TagRepository struct {  db \*gorm.DB  s TagSearch  } )  func NewTagRepository(db \*gorm.DB, s TagSearch) \*TagRepository {  return &TagRepository{  db: db,  s: s,  } }  func (tr \*TagRepository) Add(tag \*model.Tag) error {  log.Printf("Adding tag: %v\n", tag)  if err := tr.db.Create(tag).Error; err != nil {  log.Printf("Error adding tag: %v\n", err)  return err  }  if err := tr.s.Index(tag); err != nil {  log.Printf("Error indexing tag: %v\n", err)  return err  }  log.Printf("Tag added successfully: %v\n", tag)  return nil }  func (tr \*TagRepository) Save(tag \*model.Tag) error {  log.Printf("Saving tag: %v\n", tag)  if err := tr.db.Save(tag).Error; err != nil {  log.Printf("Error saving tag: %v\n", err)  return err  }  if err := tr.s.Index(tag); err != nil {  log.Printf("Error indexing tag: %v\n", err)  return err  }  log.Printf("Tag saved successfully: %v\n", tag)  return nil }  func (tr \*TagRepository) Delete(id uint) error {  log.Printf("Deleting tag by id: %d\n", id)  if err := tr.db.Delete(&model.Tag{ID: id}).Error; err != nil {  log.Printf("Error deleting tag: %v\n", err)  return err  }  if err := tr.s.Delete(id); err != nil {  log.Printf("Error deleting tag from search index: %v\n", err)  return err  }  log.Printf("Tag deleted successfully: %d\n", id)  return nil }  func (tr \*TagRepository) Find(id uint) (\*model.Tag, error) {  log.Printf("Finding tag by id: %d\n", id)  tag := &model.Tag{}  if err := tr.db.Preload("Posts").First(tag, id).Error; err != nil {  log.Printf("Error finding tag: %v\n", err)  return nil, err  }  log.Printf("Tag found: %v\n", tag)  return tag, nil }  func (tr \*TagRepository) FindByName(name string) (\*model.Tag, error) {  log.Printf("Finding tag by name: %s\n", name)  tag := &model.Tag{}  if err := tr.db.Preload("Posts").Where("name = ?", name).First(tag).Error; err != nil {  log.Printf("Error finding tag: %v\n", err)  return nil, err  }  log.Printf("Tag found: %v\n", tag)  return tag, nil }  func (tr \*TagRepository) FindSome(skip, limit int) ([]\*model.Tag, error) {  log.Printf("Finding some tags with skip: %d and limit: %d\n", skip, limit)  var tags []\*model.Tag  if limit < 0 {  skip = -1  }  if err := tr.db.Preload("Posts").Offset(skip).Limit(limit).Find(&tags).Error; err != nil {  log.Printf("Error finding some tags: %v\n", err)  return nil, err  }  log.Printf("Found some tags: %v\n", tags)  return tags, nil }  func (tr \*TagRepository) FindByPost(postID uint, skip, limit int) ([]\*model.Tag, error) {  log.Printf("Finding tags by post: %d\n", postID)  var tags []\*model.Tag  if limit < 0 {  skip = -1  }  if err := tr.db.Preload("Posts").Joins("JOIN post\_tags ON post\_tags.tag\_id = tags.id").Where("post\_tags.post\_id = ?", postID).Offset(skip).Limit(limit).Find(&tags).Error; err != nil {  log.Printf("Error finding tags: %v\n", err)  return nil, err  }  log.Printf("Tags found: %v\n", tags)  return tags, nil }  func (tr \*TagRepository) ClearAssociations(tagID uint) error {  log.Printf("Clearing associations of tag: %d\n", tagID)  if err := tr.db.Model(&model.Tag{ID: tagID}).Association("Posts").Clear(); err != nil {  log.Printf("Error clearing associations: %v\n", err)  return err  }  log.Printf("Associations cleared for tag: %d\n", tagID)  return nil }  func (tr \*TagRepository) AddPost(tagID, postID uint) error {  log.Printf("Adding post: %d to tag: %d\n", postID, tagID)  if err := tr.db.Model(&model.Tag{ID: tagID}).Association("Posts").Append(&model.Post{ID: postID}); err != nil {  log.Printf("Error adding post to tag: %v\n", err)  return err  }  log.Printf("Post: %d added to tag: %d\n", postID, tagID)  return nil }  func (tr \*TagRepository) RemovePost(tagID, postID uint) error {  log.Printf("Removing post: %d from tag: %d\n", postID, tagID)  if err := tr.db.Model(&model.Tag{ID: tagID}).Association("Posts").Delete(&model.Post{ID: postID}); err != nil {  log.Printf("Error removing post from tag: %v\n", err)  return err  }  log.Printf("Post: %d removed from tag: %d\n", postID, tagID)  return nil }  func (tr \*TagRepository) Search(query string, skip, limit int) ([]\*model.Tag, error) {  log.Printf("Searching tags by query: %s with skip: %d and limit: %d\n", query, skip, limit)  ids, err := tr.s.Search(query, skip, limit)  if err != nil {  log.Printf("Error searching tags: %v\n", err)  return nil, err  }   var tags []\*model.Tag  if err := tr.db.Preload("Posts").Where("id IN ?", ids).Find(&tags).Error; err != nil {  log.Printf("Error finding tags by IDs: %v\n", err)  return nil, err  }  log.Printf("Tags found: %v\n", tags)  return tags, nil } |

Код repository.go:

|  |
| --- |
| package repository  import (  "log"   "gorm.io/gorm"  "gorm.io/gorm/logger"   "github.com/SocialNetworkY/Backend/internal/post/model" )  type Repository struct {  Post \*PostRepository  Comment \*CommentRepository  Like \*LikeRepository  Tag \*TagRepository }  func New(dialector gorm.Dialector, ps PostSearch, cs CommentSearch, ts TagSearch) (\*Repository, error) {  log.Printf("Connecting %s...\n", dialector.Name())  db, err := gorm.Open(dialector, &gorm.Config{  Logger: logger.Default.LogMode(logger.*Silent*),  })  if err != nil {  return nil, err  }  log.Printf("Connected %s\n", dialector.Name())   log.Println("Starting AutoMigrating...")  if err := db.AutoMigrate(&model.Post{}, &model.Like{}, &model.Comment{}, &model.Tag{}); err != nil {  return nil, err  }  log.Println("AutoMigrating completed")   return &Repository{  Post: NewPostRepository(db, ps),  Comment: NewCommentRepository(db, cs),  Like: NewLikeRepository(db),  Tag: NewTagRepository(db, ts),  }, nil } |

**Завдання №3.** Реалізація service layer

Реалізація service layer з використанням repository layer та пакетів з pkg в стилі dependency injection.

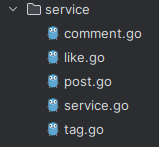


Рисунок 7 — Структура папки service

Код comment.go:

|  |
| --- |
| package service  import (  "log"   "github.com/SocialNetworkY/Backend/internal/post/model" )  type (  CommentRepo interface {  Add(comment \*model.Comment) error  Save(comment \*model.Comment) error  Delete(id uint) error  Find(id uint) (\*model.Comment, error)  FindSome(skip, limit int) ([]\*model.Comment, error)  FindByPost(postID uint, skip, limit int) ([]\*model.Comment, error)  FindByUser(userID uint, skip, limit int) ([]\*model.Comment, error)  Search(query string, skip, limit int) ([]\*model.Comment, error)  }   CommentService struct {  repo CommentRepo  } )  func NewCommentService(r CommentRepo) \*CommentService {  return &CommentService{r} }  // Find returns a comment by its ID func (cs \*CommentService) Find(id uint) (\*model.Comment, error) {  log.Printf("Finding comment with ID: %d\n", id)  comment, err := cs.repo.Find(id)  if err != nil {  log.Printf("Error finding comment: %v\n", err)  return nil, err  }  log.Printf("Comment found: %v\n", comment)  return comment, nil }  // FindByPost returns comments for a post func (cs \*CommentService) FindByPost(postID uint, skip, limit int) ([]\*model.Comment, error) {  log.Printf("Finding comments for post with ID: %d, skip: %d, limit: %d\n", postID, skip, limit)  comments, err := cs.repo.FindByPost(postID, skip, limit)  if err != nil {  log.Printf("Error finding comments: %v\n", err)  return nil, err  }  log.Printf("Comments found: %v\n", comments)  return comments, nil }  // CommentPost adds a comment to a post func (cs \*CommentService) CommentPost(postID, userID uint, content string) error {  log.Printf("Adding comment to post with ID: %d by user with ID: %d\n", postID, userID)  err := cs.repo.Add(&model.Comment{  UserID: userID,  PostID: postID,  Content: content,  })  if err != nil {  log.Printf("Error adding comment: %v\n", err)  return err  }  log.Printf("Comment added successfully to post with ID: %d\n", postID)  return nil }  // Edit updates a comment func (cs \*CommentService) Edit(id, userID uint, content string) error {  log.Printf("Editing comment with ID: %d by user with ID: %d\n", id, userID)  comment, err := cs.repo.Find(id)  if err != nil {  log.Printf("Error finding comment: %v\n", err)  return err  }   comment.Content = content  comment.Edited = true  comment.EditedBy = userID  err = cs.repo.Save(comment)  if err != nil {  log.Printf("Error saving comment: %v\n", err)  return err  }  log.Printf("Comment with ID: %d edited successfully\n", id)  return nil }  // Delete removes a comment func (cs \*CommentService) Delete(id uint) error {  log.Printf("Deleting comment with ID: %d\n", id)  err := cs.repo.Delete(id)  if err != nil {  log.Printf("Error deleting comment: %v\n", err)  return err  }  log.Printf("Comment with ID: %d deleted successfully\n", id)  return nil }  func (cs \*CommentService) DeleteByPost(postID uint) error {  log.Printf("Deleting comments for post with ID: %d\n", postID)  comments, err := cs.repo.FindByPost(postID, 0, 0)  if err != nil {  log.Printf("Error finding comments: %v\n", err)  return err  }   for \_, comment := range comments {  if err := cs.Delete(comment.ID); err != nil {  return err  }  }  log.Printf("Comments for post with ID: %d deleted successfully\n", postID)  return nil }  func (cs \*CommentService) DeleteByUser(userID uint) error {  log.Printf("Deleting comments by user with ID: %d\n", userID)  comments, err := cs.repo.FindByUser(userID, 0, 0)  if err != nil {  log.Printf("Error finding comments: %v\n", err)  return err  }   for \_, comment := range comments {  if err := cs.Delete(comment.ID); err != nil {  return err  }  }  log.Printf("Comments by user with ID: %d deleted successfully\n", userID)  return nil }  func (cs \*CommentService) Search(query string, skip, limit int) ([]\*model.Comment, error) {  log.Printf("Searching comments with query: %s, skip: %d, limit: %d\n", query, skip, limit)  comments, err := cs.repo.Search(query, skip, limit)  if err != nil {  log.Printf("Error searching comments: %v\n", err)  return nil, err  }  log.Printf("Comments found: %v\n", comments)  return comments, nil } |

Код like.go:

|  |
| --- |
| package service  import (  "errors"  "log"   "github.com/SocialNetworkY/Backend/internal/post/model" )  type (  LikeRepo interface {  Add(like \*model.Like) error  Delete(id uint) error  FindByPost(postID uint, skip, limit int) ([]\*model.Like, error)  FindByUser(userID uint, skip, limit int) ([]\*model.Like, error)  FindByPostUser(postID, userID uint) (\*model.Like, error)  }   LikeService struct {  repo LikeRepo  } )  func NewLikeService(r LikeRepo) \*LikeService {  return &LikeService{r} }  // LikePost adds a like to a post func (ls \*LikeService) LikePost(postID, userID uint) error {  log.Printf("Adding like to post with ID: %d by user with ID: %d\n", postID, userID)  like, \_ := ls.repo.FindByPostUser(postID, userID)  if like != nil {  log.Printf("Error: user already liked the post\n")  return errors.New("user already liked the post")  }   err := ls.repo.Add(&model.Like{  UserID: userID,  PostID: postID,  })  if err != nil {  log.Printf("Error adding like: %v\n", err)  return err  }   log.Printf("Like added successfully to post with ID: %d by user with ID: %d\n", postID, userID)  return nil }  // UnlikePost removes a like from a post func (ls \*LikeService) UnlikePost(postID, userID uint) error {  log.Printf("Removing like from post with ID: %d by user with ID: %d\n", postID, userID)  like, \_ := ls.repo.FindByPostUser(postID, userID)  if like == nil {  log.Printf("Error: user has not liked the post\n")  return errors.New("user has not liked the post")  }   err := ls.repo.Delete(like.ID)  if err != nil {  log.Printf("Error removing like: %v\n", err)  return err  }   log.Printf("Like removed successfully from post with ID: %d by user with ID: %d\n", postID, userID)  return nil }  func (ls \*LikeService) FindByPostUser(postID, userID uint) (\*model.Like, error) {  log.Printf("Finding like for post with ID: %d by user with ID: %d\n", postID, userID)  like, err := ls.repo.FindByPostUser(postID, userID)  if err != nil {  log.Printf("Error finding like: %v\n", err)  return nil, err  }  log.Printf("Like found: %v\n", like)  return like, nil }  func (ls \*LikeService) FindByPost(postID uint, skip, limit int) ([]\*model.Like, error) {  log.Printf("Finding likes for post with ID: %d, skip: %d, limit: %d\n", postID, skip, limit)  likes, err := ls.repo.FindByPost(postID, skip, limit)  if err != nil {  log.Printf("Error finding likes: %v\n", err)  return nil, err  }  log.Printf("Likes found: %v\n", likes)  return likes, nil }  func (ls \*LikeService) FindByUser(userID uint, skip, limit int) ([]\*model.Like, error) {  log.Printf("Finding likes by user with ID: %d, skip: %d, limit: %d\n", userID, skip, limit)  likes, err := ls.repo.FindByUser(userID, skip, limit)  if err != nil {  log.Printf("Error finding likes: %v\n", err)  return nil, err  }  log.Printf("Likes found: %v\n", likes)  return likes, nil }  func (ls \*LikeService) Delete(id uint) error {  log.Printf("Deleting like with ID: %d\n", id)  err := ls.repo.Delete(id)  if err != nil {  log.Printf("Error deleting like: %v\n", err)  return err  }  log.Printf("Like with ID: %d deleted successfully\n", id)  return nil }  func (ls \*LikeService) DeleteByPost(postID uint) error {  log.Printf("Deleting likes for post with ID: %d\n", postID)  likes, err := ls.repo.FindByPost(postID, 0, 0)  if err != nil {  log.Printf("Error finding likes: %v\n", err)  return err  }   for \_, like := range likes {  if err := ls.Delete(like.ID); err != nil {  return err  }  }   log.Printf("Likes for post with ID: %d deleted successfully\n", postID)  return nil }  func (ls \*LikeService) DeleteByUser(userID uint) error {  log.Printf("Deleting likes by user with ID: %d\n", userID)  likes, err := ls.repo.FindByUser(userID, 0, 0)  if err != nil {  log.Printf("Error finding likes: %v\n", err)  return err  }   for \_, like := range likes {  if err := ls.Delete(like.ID); err != nil {  return err  }  }   log.Printf("Likes by user with ID: %d deleted successfully\n", userID)  return nil } |

Код post.go:

|  |
| --- |
| package service  import (  "context"  "log"   "github.com/SocialNetworkY/Backend/internal/post/model" )  type (  PostRepo interface {  Add(post \*model.Post) error  Save(post \*model.Post) error  Delete(id uint) error  Find(id uint) (\*model.Post, error)  FindSome(skip, limit int) ([]\*model.Post, error)  FindByUser(userID uint, skip, limit int) ([]\*model.Post, error)  FindByTag(tagID uint, skip, limit int) ([]\*model.Post, error)  ClearAssociations(postID uint) error  Search(query string, skip, limit int) ([]\*model.Post, error)  }   ReportGateway interface {  DeletePostReports(ctx context.Context, postID uint) error  }   PostService struct {  repo PostRepo  rg ReportGateway  ts \*TagService  cs \*CommentService  ls \*LikeService  } )  func NewPostService(r PostRepo, rg ReportGateway, ts \*TagService, cs \*CommentService, ls \*LikeService) \*PostService {  return &PostService{  repo: r,  rg: rg,  ts: ts,  cs: cs,  ls: ls,  } }  // Create creates a new post func (ps \*PostService) Create(post \*model.Post) error {  log.Printf("Creating post: %v\n", post)  if err := ps.processTags(post); err != nil {  log.Printf("Error processing tags: %v\n", err)  return err  }   if err := ps.repo.Add(post); err != nil {  log.Printf("Error adding post: %v\n", err)  return err  }   log.Printf("Post created successfully: %v\n", post)  return nil }  // Update updates a post func (ps \*PostService) Update(post \*model.Post) error {  log.Printf("Updating post: %v\n", post)  if err := ps.repo.ClearAssociations(post.ID); err != nil {  log.Printf("Error clearing associations: %v\n", err)  return err  }   if err := ps.processTags(post); err != nil {  log.Printf("Error processing tags: %v\n", err)  return err  }   if err := ps.repo.Save(post); err != nil {  log.Printf("Error saving post: %v\n", err)  return err  }   log.Printf("Post updated successfully: %v\n", post)  return nil }  // Find returns a post by its ID func (ps \*PostService) Find(id uint) (\*model.Post, error) {  log.Printf("Finding post with ID: %d\n", id)  post, err := ps.repo.Find(id)  if err != nil {  log.Printf("Error finding post: %v\n", err)  return nil, err  }   log.Printf("Post found: %v\n", post)  return post, nil }  // Delete deletes a post by its ID func (ps \*PostService) Delete(id uint) error {  log.Printf("Deleting post with ID: %d\n", id)  if err := ps.repo.ClearAssociations(id); err != nil {  log.Printf("Error clearing associations: %v\n", err)  return err  }   if err := ps.ls.DeleteByPost(id); err != nil {  log.Printf("Error deleting likes: %v\n", err)  return err  }   if err := ps.cs.DeleteByPost(id); err != nil {  log.Printf("Error deleting comments: %v\n", err)  return err  }   if err := ps.rg.DeletePostReports(context.Background(), id); err != nil {  log.Printf("Error deleting reports: %v\n", err)  return err  }   if err := ps.repo.Delete(id); err != nil {  log.Printf("Error deleting post: %v\n", err)  return err  }   log.Printf("Post deleted successfully: %d\n", id)  return nil }  // DeleteByUser deletes all posts by a user func (ps \*PostService) DeleteByUser(userID uint) error {  log.Printf("Deleting posts by user with ID: %d\n", userID)  posts, err := ps.repo.FindByUser(userID, 0, 0)  if err != nil {  log.Printf("Error finding posts: %v\n", err)  return err  }   for \_, post := range posts {  if err := ps.Delete(post.ID); err != nil {  return err  }  }   log.Printf("Posts by user with ID %d deleted successfully\n", userID)  return nil }  // AddTag adds a tag to a post func (ps \*PostService) AddTag(postID uint, tagName string) error {  log.Printf("Adding tag %s to post %d\n", tagName, postID)  post, err := ps.repo.Find(postID)  if err != nil {  log.Printf("Error finding post: %v\n", err)  return err  }   tag, err := ps.ts.FindOrCreate(tagName)  if err != nil {  log.Printf("Error finding or creating tag: %v\n", err)  return err  }   // Check if the tag is already in the post  for \_, t := range post.Tags {  if t.ID == tag.ID {  return nil  }  }   post.Tags = append(post.Tags, tag)   if err := ps.repo.Save(post); err != nil {  log.Printf("Error saving post: %v\n", err)  return err  }   log.Printf("Tag %s added to post %d successfully\n", tagName, postID)  return nil }  // FindSome returns some posts with pagination parameters func (ps \*PostService) FindSome(skip, limit int) ([]\*model.Post, error) {  log.Printf("Finding some posts with skip: %d and limit: %d\n", skip, limit)  posts, err := ps.repo.FindSome(skip, limit)  if err != nil {  log.Printf("Error finding some posts: %v\n", err)  return nil, err  }   log.Printf("Found some posts: %v\n", posts)  return posts, nil }  // FindByUser returns some posts by user ID with pagination parameters func (ps \*PostService) FindByUser(userID uint, skip, limit int) ([]\*model.Post, error) {  log.Printf("Finding posts by user with ID: %d, skip: %d, limit: %d\n", userID, skip, limit)  posts, err := ps.repo.FindByUser(userID, skip, limit)  if err != nil {  log.Printf("Error finding posts by user: %v\n", err)  return nil, err  }   log.Printf("Found posts by user: %v\n", posts)  return posts, nil }  // processTags processes tags of a post func (ps \*PostService) processTags(post \*model.Post) error {  log.Printf("Processing tags: %v\n", post.Tags)  for i, tag := range post.Tags {  var err error  if post.Tags[i], err = ps.ts.FindOrCreate(tag.Name); err != nil {  log.Printf("Error finding or creating tag: %v\n", err)  return err  }  }   log.Printf("Tags processed successfully: %v\n", post.Tags)  return nil }  func (ps \*PostService) Search(query string, skip, limit int) ([]\*model.Post, error) {  log.Printf("Searching posts with query: %s, skip: %d, limit: %d\n", query, skip, limit)  posts, err := ps.repo.Search(query, skip, limit)  if err != nil {  log.Printf("Error searching posts: %v\n", err)  return nil, err  }   log.Printf("Found posts: %v\n", posts)  return posts, nil } |

Код tag.go:

|  |
| --- |
| package service  import (  "log"   "github.com/SocialNetworkY/Backend/internal/post/model" )  type (  TagRepo interface {  Add(tag \*model.Tag) error  Save(tag \*model.Tag) error  Delete(id uint) error  Find(id uint) (\*model.Tag, error)  FindByName(name string) (\*model.Tag, error)  FindSome(skip, limit int) ([]\*model.Tag, error)  FindByPost(postID uint, skip, limit int) ([]\*model.Tag, error)  ClearAssociations(tagID uint) error  Search(query string, skip, limit int) ([]\*model.Tag, error)  }   TagService struct {  repo TagRepo  } )  func NewTagService(r TagRepo) \*TagService {  return &TagService{  repo: r,  } }  // Add adds a new tag func (ts \*TagService) Add(tag \*model.Tag) error {  log.Printf("Adding tag: %v\n", tag)  err := ts.repo.Add(tag)  if err != nil {  log.Printf("Error adding tag: %v\n", err)  return err  }  log.Printf("Tag added successfully: %v\n", tag)  return nil }  // Delete deletes a tag by id func (ts \*TagService) Delete(id uint) error {  log.Printf("Deleting tag with ID: %d\n", id)  if err := ts.repo.ClearAssociations(id); err != nil {  log.Printf("Error clearing associations: %v\n", err)  return err  }   if err := ts.repo.Delete(id); err != nil {  log.Printf("Error deleting tag: %v\n", err)  return err  }   log.Printf("Tag deleted successfully: %d\n", id)  return nil }  // DeleteByName deletes a tag by name func (ts \*TagService) DeleteByName(name string) error {  log.Printf("Deleting tag with name: %s\n", name)  tag, err := ts.repo.FindByName(name)  if err != nil {  log.Printf("Error finding tag: %v\n", err)  return err  }   if err := ts.Delete(tag.ID); err != nil {  return err  }   log.Printf("Tag with name %s deleted successfully\n", name)  return nil }  // Find finds a tag by id func (ts \*TagService) Find(id uint) (\*model.Tag, error) {  log.Printf("Finding tag with ID: %d\n", id)  tag, err := ts.repo.Find(id)  if err != nil {  log.Printf("Error finding tag: %v\n", err)  return nil, err  }  log.Printf("Tag found: %v\n", tag)  return tag, nil }  // FindByName finds a tag by name func (ts \*TagService) FindByName(name string) (\*model.Tag, error) {  log.Printf("Finding tag with name: %s\n", name)  tag, err := ts.repo.FindByName(name)  if err != nil {  log.Printf("Error finding tag: %v\n", err)  return nil, err  }  log.Printf("Tag found: %v\n", tag)  return tag, nil }  // Exists checks if a tag exists func (ts \*TagService) Exists(id uint) (bool, error) {  log.Printf("Checking if tag with ID: %d exists\n", id)  \_, err := ts.repo.Find(id)  exists := err == nil  if !exists {  log.Printf("Tag with ID: %d does not exist\n", id)  }  return exists, err }  // ExistsByName checks if a tag exists func (ts \*TagService) ExistsByName(name string) bool {  log.Printf("Checking if tag with name: %s exists\n", name)  \_, err := ts.repo.FindByName(name)  exists := err == nil  if !exists {  log.Printf("Tag with name: %s does not exist\n", name)  }  return exists }  // FindOrCreate finds a tag by name or creates a new one func (ts \*TagService) FindOrCreate(name string) (\*model.Tag, error) {  log.Printf("Finding or creating tag: %s\n", name)  tag, err := ts.repo.FindByName(name)  if err != nil {  tag = &model.Tag{  Name: name,  }   if err := ts.repo.Add(tag); err != nil {  log.Printf("Error adding tag: %v\n", err)  return nil, err  }  }   log.Printf("Tag found or created: %v\n", tag)  return tag, nil }  func (ts \*TagService) Search(query string, skip, limit int) ([]\*model.Tag, error) {  log.Printf("Searching tags with query: %s, skip: %d, limit: %d\n", query, skip, limit)  tags, err := ts.repo.Search(query, skip, limit)  if err != nil {  log.Printf("Error searching tags: %v\n", err)  return nil, err  }  log.Printf("Tags found: %v\n", tags)  return tags, nil } |

Код service.go:

|  |
| --- |
| package service  type Service struct {  Post \*PostService  Tag \*TagService  Like \*LikeService  Comment \*CommentService }  func New(postRepo PostRepo, tagRepo TagRepo, likeRepo LikeRepo, commentRepo CommentRepo, rg ReportGateway) \*Service {  ts := NewTagService(tagRepo)  ls := NewLikeService(likeRepo)  cs := NewCommentService(commentRepo)  ps := NewPostService(postRepo, rg, ts, cs, ls)   return &Service{  Post: ps,  Tag: ts,  Like: ls,  Comment: cs,  } } |

**Завдання №4.** Реалізація transport layer

Реалізація http сервера з підключенням API v1.

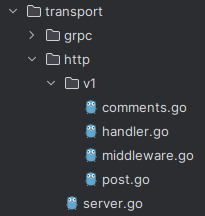


Рисунок 8 — Структура http в transport layer

Код server.go:

|  |
| --- |
| package http  import (  "fmt"  "github.com/SocialNetworkY/Backend/internal/post/transport/http/v1"  "github.com/SocialNetworkY/Backend/pkg/binder"  "github.com/SocialNetworkY/Backend/pkg/validator"  "log"  "net/http"   "github.com/labstack/echo/v4"  "github.com/labstack/echo/v4/middleware" )  type Server struct {  echo \*echo.Echo  addr string }  func New(bodyLimit string, allowedOrigins []string, port int) \*Server {  log.Printf("Creating rest server with port: %d", port)   e := echo.New()  e.Binder = binder.NewEchoCustomBinder()  e.Validator = validator.NewEchoCustomValidator()  e.Use(middleware.Recover())   e.Use(middleware.BodyLimit(bodyLimit))  e.Use(middleware.LoggerWithConfig(middleware.LoggerConfig{  Format: "${time\_custom} | ${status} | ${latency\_human} | ${remote\_ip} | ${method} | ${uri} | ${error}\n",  CustomTimeFormat: "2006-01-02 15:04:05",  }))   corsConfig := middleware.CORSConfig{  AllowOrigins: allowedOrigins,  }  e.Use(middleware.CORSWithConfig(corsConfig))   e.Pre(middleware.RemoveTrailingSlash())   e.IPExtractor = echo.ExtractIPFromRealIPHeader()   e.GET("/ping", func(c echo.Context) error {  return c.String(http.*StatusOK*, "pong")  })   return &Server{  echo: e,  addr: fmt.Sprintf(":%d", port),  } }  // AddStaticFolder adds a static folder to the server func (s \*Server) AddStaticFolder(path string, folder string) \*Server {  s.echo.Static(path, folder)  return s }  func (s \*Server) Init(ps v1.PostService, ls v1.LikeService, cs v1.CommentService, ag v1.AuthGateway, us v1.UserGateway, fs v1.FileStorage) \*Server {  log.Println("Initializing server...")  log.Println("Initializing api...")  handlerV1 := v1.New(ps, ls, cs, ag, us, fs)  api := s.echo.Group("/api")  {  handlerV1.Init(api)  }   return s }  func (s \*Server) Run() error {  log.Println("Starting server")  return s.echo.Start(s.addr) } |

Код v1/handler.go:

|  |
| --- |
| package v1  import (  "context"  "io"  "log"   "github.com/SocialNetworkY/Backend/internal/post/model"  "github.com/labstack/echo/v4" )  type (  PostService interface {  Create(post \*model.Post) error  Update(post \*model.Post) error  Delete(id uint) error  Find(id uint) (\*model.Post, error)  FindSome(skip, limit int) ([]\*model.Post, error)  FindByUser(userID uint, skip, limit int) ([]\*model.Post, error)  Search(query string, skip, limit int) ([]\*model.Post, error)  }   LikeService interface {  LikePost(postID, userID uint) error  UnlikePost(postID, userID uint) error  }   CommentService interface {  Find(id uint) (\*model.Comment, error)  FindByPost(postID uint, skip, limit int) ([]\*model.Comment, error)  CommentPost(postID, userID uint, content string) error  Edit(id, userID uint, content string) error  Delete(id uint) error  Search(query string, skip, limit int) ([]\*model.Comment, error)  }   AuthGateway interface {  Authenticate(ctx context.Context, auth string) (uint, error)  }   UserGateway interface {  UserInfo(ctx context.Context, userID uint) (\*model.User, error)  }   FileStorage interface {  UploadFile(file io.ReadSeeker, fileName string) (string, error)  }   Handler struct {  ps PostService  ls LikeService  cs CommentService  ag AuthGateway  ug UserGateway  fs FileStorage  } )  func New(ps PostService, ls LikeService, cs CommentService, ag AuthGateway, ug UserGateway, fs FileStorage) \*Handler {  return &Handler{  ps: ps,  ls: ls,  cs: cs,  ag: ag,  ug: ug,  fs: fs,  } }  func (h \*Handler) Init(api \*echo.Group) {  log.Println("Initializing V1 api")  v1 := api.Group("/v1")  {  h.initPostApi(v1)  h.initCommentsApi(v1)  } } |

Код v1/middleware.go:

|  |
| --- |
| package v1  import (  "fmt"  "net/http"  "strconv"   "github.com/SocialNetworkY/Backend/internal/post/model"  "github.com/SocialNetworkY/Backend/pkg/constant"  "github.com/labstack/echo/v4" )  const (  *postIDParam* = "post\_id"  *userIDParam* = "user\_id"  *commentIDParam* = "comment\_id"   *postLocals* = "post"  *userLocals* = "user"  *requesterLocals* = "requester"  *commentLocals* = "comment"   *skipQuery* = "skip"  *limitQuery* = "limit"  *queryQuery* = "query"   *defaultSkip* = 0  *defaultLimit* = 10 )  func (h \*Handler) authenticationMiddleware(next echo.HandlerFunc) echo.HandlerFunc {  return func(c echo.Context) error {  authHeader := c.Request().Header.Get(constant.*HTTPAuthorizationHeader*)  if authHeader == "" {  return echo.NewHTTPError(http.*StatusUnauthorized*, "Missing Authorization header")  }   requesterID, err := h.ag.Authenticate(c.Request().Context(), authHeader)  if err != nil {  return echo.NewHTTPError(http.*StatusUnauthorized*, err.Error())  }   requester, err := h.ug.UserInfo(c.Request().Context(), requesterID)  if err != nil {  return echo.NewHTTPError(http.*StatusUnauthorized*, err.Error())  }   c.Set(*requesterLocals*, requester)   return next(c)  } }  func (h \*Handler) setUserByIDMiddleware(next echo.HandlerFunc) echo.HandlerFunc {  return func(c echo.Context) error {  userID, err := getUintParam(c, *userIDParam*)  if err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   user, err := h.ug.UserInfo(c.Request().Context(), userID)  if err != nil {  return echo.NewHTTPError(http.*StatusUnauthorized*, err.Error())  }   c.Set(*userLocals*, user)   return next(c)  } }  func (h \*Handler) banMiddleware(next echo.HandlerFunc) echo.HandlerFunc {  return func(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "failed to get requester")  }   if requester.Banned {  return echo.NewHTTPError(http.*StatusForbidden*, fmt.Sprintf("You are banned: %s.\nUntil: %s", requester.BanReason, requester.BanExpiredAt))  }   return next(c)  } }  func (h \*Handler) setPostByIDMiddleware(next echo.HandlerFunc) echo.HandlerFunc {  return func(c echo.Context) error {  postID, err := getUintParam(c, *postIDParam*)  if err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   post, err := h.ps.Find(postID)  if err != nil {  return echo.NewHTTPError(http.*StatusNotFound*, err.Error())  }   c.Set(*postLocals*, post)   return next(c)  } }  func (h \*Handler) setCommentByIDMiddleware(next echo.HandlerFunc) echo.HandlerFunc {  return func(c echo.Context) error {  commentID, err := getUintParam(c, *commentIDParam*)  if err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   comment, err := h.cs.Find(commentID)  if err != nil {  return echo.NewHTTPError(http.*StatusNotFound*, err.Error())  }   c.Set(*commentLocals*, comment)   return next(c)  } }  func getUintParam(c echo.Context, key string) (uint, error) {  param := c.Param(key)  if param == "" {  return 0, echo.NewHTTPError(http.*StatusBadRequest*, "missing parameter "+key)  }   id, err := strconv.ParseUint(param, 10, 64)  if err != nil {  return 0, echo.NewHTTPError(http.*StatusBadRequest*, "invalid parameter "+key)  }   return uint(id), nil }  func skipLimitQuery(c echo.Context) (int, int) {  skip := *defaultSkip* if s, err := strconv.Atoi(c.QueryParam(*skipQuery*)); err == nil {  skip = s  }  limit := *defaultLimit* if l, err := strconv.Atoi(c.QueryParam(*limitQuery*)); err == nil {  limit = l  }   return skip, limit } |

Код v1/comments.go:

|  |
| --- |
| package v1  import (  "fmt"  "net/http"   "github.com/SocialNetworkY/Backend/internal/post/model"  "github.com/SocialNetworkY/Backend/pkg/constant"  "github.com/labstack/echo/v4" )  func (h \*Handler) initCommentsApi(api \*echo.Group) {  comments := api.Group("/comments")  {  comments.GET("/search", h.searchComments)   commentID := comments.Group(fmt.Sprintf("/:%s", *commentIDParam*), h.setCommentByIDMiddleware)  {  commentID.PUT("", h.changeComment, h.authenticationMiddleware)  commentID.DELETE("", h.deleteComment, h.authenticationMiddleware)  }   } }  func (h \*Handler) changeComment(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get requester from context")  }   comment, ok := c.Get(*commentLocals*).(\*model.Comment)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get comment from context")  }   if requester.ID != comment.UserID && requester.Role == constant.*RoleUser* {  return echo.NewHTTPError(http.*StatusForbidden*, "you are not allowed to edit this comment")  }   var req = struct {  Content string `json:"content" validate:"required,min=1,max=255"`  }{}   if err := c.Bind(&req); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   if err := c.Validate(req); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   if err := h.cs.Edit(comment.ID, requester.ID, req.Content); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   comment, err := h.cs.Find(comment.ID)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusOK*, comment) }  func (h \*Handler) deleteComment(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get requester from context")  }   comment, ok := c.Get(*commentLocals*).(\*model.Comment)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get comment from context")  }   if requester.ID != comment.UserID && requester.Role == constant.*RoleUser* {  return echo.NewHTTPError(http.*StatusForbidden*, "you are not allowed to delete this comment")  }   if err := h.cs.Delete(comment.ID); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.NoContent(http.*StatusNoContent*) }  func (h \*Handler) searchComments(c echo.Context) error {  skip, limit := skipLimitQuery(c)  query := c.QueryParam(*queryQuery*)  if query == "" {  return echo.NewHTTPError(http.*StatusBadRequest*, "query is required")  }   comments, err := h.cs.Search(query, skip, limit)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusOK*, comments) } |

Код v1/post.go:

|  |
| --- |
| package v1  import (  "fmt"  "mime/multipart"  "net/http"  "strings"  "time"   "github.com/SocialNetworkY/Backend/internal/post/model"  "github.com/SocialNetworkY/Backend/pkg/constant"  "github.com/labstack/echo/v4" )  const (  *imageMaxSize* = 50 \* 1024 \* 1024 // 50MB  *videoMaxSize* = 100 \* 1024 \* 1024 // 100MB )  func (h \*Handler) initPostApi(api \*echo.Group) {  posts := api.Group("/posts")  {  posts.GET("", h.getPosts)  posts.GET("/search", h.searchPosts)  posts.POST("", h.createPost, h.authenticationMiddleware, h.banMiddleware)  posts.GET(fmt.Sprintf("/users/:%s", *userIDParam*), h.getPostsByUserID, h.setUserByIDMiddleware)   postID := posts.Group(fmt.Sprintf("/:%s", *postIDParam*), h.setPostByIDMiddleware)  {  postID.GET("", h.getPost)  postID.PATCH("", h.updatePost, h.authenticationMiddleware, h.banMiddleware)  postID.DELETE("", h.deletePost, h.authenticationMiddleware, h.banMiddleware)  postID.GET("/like", h.likePost, h.authenticationMiddleware, h.banMiddleware)  postID.GET("/unlike", h.unlikePost, h.authenticationMiddleware, h.banMiddleware)   // Comments  comments := postID.Group("/comments")  {  comments.GET("", h.getPostComments)  comments.POST("", h.commentPost, h.authenticationMiddleware, h.banMiddleware)  }  }  } }  func (h \*Handler) createPost(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get requester")  }   req := &struct {  Title string `form:"title" validate:"required,min=1,max=255"`  Content string `form:"content" validate:"required,min=1,max=65535"`  Tags []string `form:"tags" validate:"omitempty,dive,min=1"`  Images []\*multipart.FileHeader `form:"images"`  Videos []\*multipart.FileHeader `form:"videos"`  }{}  if err := c.Bind(req); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   if err := c.Validate(req); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   // Process images  var imageUrls []string  for \_, imageHeader := range req.Images {  if !strings.HasPrefix(imageHeader.Header.Get("Content-Type"), "image/") {  return echo.NewHTTPError(http.*StatusBadRequest*, "file is not an image")  }   if imageHeader.Size > *imageMaxSize* {  return echo.NewHTTPError(http.*StatusBadRequest*, fmt.Sprintf("image file size is too large, max size is %d", *imageMaxSize*))  }   image, err := imageHeader.Open()  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   imageUrl, err := h.fs.UploadFile(image, fmt.Sprintf("%d\_%d\_%s", requester.ID, time.Now().Unix(), imageHeader.Filename))  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }  imageUrls = append(imageUrls, imageUrl)  }   // Process videos  var videoUrls []string  for \_, videoHeader := range req.Videos {  if !strings.HasPrefix(videoHeader.Header.Get("Content-Type"), "video/") {  return echo.NewHTTPError(http.*StatusBadRequest*, "file is not a video")  }   if videoHeader.Size > *videoMaxSize* {  return echo.NewHTTPError(http.*StatusBadRequest*, fmt.Sprintf("video file size is too large, max size is %d", *videoMaxSize*))  }   video, err := videoHeader.Open()  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   videoUrl, err := h.fs.UploadFile(video, fmt.Sprintf("%d\_%d\_%s", requester.ID, time.Now().Unix(), videoHeader.Filename))  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }  videoUrls = append(videoUrls, videoUrl)  }   tags := make([]\*model.Tag, len(req.Tags))  for i, tag := range req.Tags {  tags[i] = &model.Tag{  Name: tag,  }  }   post := &model.Post{  UserID: requester.ID,  Title: req.Title,  Content: req.Content,  Tags: tags,  ImageURLs: imageUrls,  VideoURLs: videoUrls,  PostedAt: time.Now(),  }   if err := h.ps.Create(post); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusCreated*, post) }  func (h \*Handler) getPost(c echo.Context) error {  post, ok := c.Get(*postLocals*).(\*model.Post)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get post")  }   return c.JSON(http.*StatusOK*, post) }  func (h \*Handler) updatePost(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get requester")  }   post, ok := c.Get(*postLocals*).(\*model.Post)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get post")  }   if requester.ID != post.UserID && requester.Role < constant.*RoleAdminLvl1* {  return echo.NewHTTPError(http.*StatusForbidden*, "you are not allowed to update this post")  }   req := &struct {  Title string `form:"title" validate:"omitempty,min=1,max=255"`  Content string `form:"content" validate:"omitempty,min=1,max=65535"`  Tags []string `form:"tags" validate:"omitempty,dive,min=1"`  ImageUrls []string `form:"image\_urls" validate:"omitempty,dive,url"`  VideoUrls []string `form:"video\_urls" validate:"omitempty,dive,url"`  Images []\*multipart.FileHeader `form:"images"`  Videos []\*multipart.FileHeader `form:"videos"`  }{}  if err := c.Bind(req); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   if err := c.Validate(req); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   if req.Title != "" {  post.Title = req.Title  }   if req.Content != "" {  post.Content = req.Content  }   if len(req.Tags) > 0 {  tags := make([]\*model.Tag, len(req.Tags))  for i, tag := range req.Tags {  tags[i] = &model.Tag{  Name: tag,  }  }  post.Tags = tags  }   if len(req.ImageUrls) > 0 {  post.ImageURLs = req.ImageUrls  }   for \_, imageHeader := range req.Images {  if imageHeader.Size > *imageMaxSize* {  return echo.NewHTTPError(http.*StatusBadRequest*, fmt.Sprintf("image file size is too large, max size is %d", *imageMaxSize*))  }   image, err := imageHeader.Open()  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   imageUrl, err := h.fs.UploadFile(image, fmt.Sprintf("%d\_%d\_%s", requester.ID, time.Now().Unix(), imageHeader.Filename))  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }  post.ImageURLs = append(post.ImageURLs, imageUrl)  }   if len(req.VideoUrls) > 0 {  post.VideoURLs = req.VideoUrls  }   for \_, videoHeader := range req.Videos {  if videoHeader.Size > *videoMaxSize* {  return echo.NewHTTPError(http.*StatusBadRequest*, fmt.Sprintf("video file size is too large, max size is %d", *videoMaxSize*))  }   video, err := videoHeader.Open()  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   videoUrl, err := h.fs.UploadFile(video, fmt.Sprintf("%d\_%d\_%s", requester.ID, time.Now().Unix(), videoHeader.Filename))  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }  post.VideoURLs = append(post.VideoURLs, videoUrl)  }   post.EditedBy = requester.ID   if err := h.ps.Update(post); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusOK*, post) }  func (h \*Handler) deletePost(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get requester")  }   post, ok := c.Get(*postLocals*).(\*model.Post)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get post")  }   if requester.ID != post.UserID && requester.Role < constant.*RoleAdminLvl1* {  return echo.NewHTTPError(http.*StatusForbidden*, "you are not allowed to delete this post")  }   if err := h.ps.Delete(post.ID); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.NoContent(http.*StatusNoContent*) }  func (h \*Handler) getPosts(c echo.Context) error {  skip, limit := skipLimitQuery(c)   posts, err := h.ps.FindSome(skip, limit)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusOK*, posts) }  func (h \*Handler) getPostsByUserID(c echo.Context) error {  user, ok := c.Get(*userLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get user")  }   skip, limit := skipLimitQuery(c)   posts, err := h.ps.FindByUser(user.ID, skip, limit)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusOK*, posts) }  func (h \*Handler) likePost(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get requester")  }   post, ok := c.Get(*postLocals*).(\*model.Post)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get post")  }   if err := h.ls.LikePost(post.ID, requester.ID); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.NoContent(http.*StatusNoContent*) }  func (h \*Handler) unlikePost(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get requester")  }   post, ok := c.Get(*postLocals*).(\*model.Post)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get post")  }   if err := h.ls.UnlikePost(post.ID, requester.ID); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.NoContent(http.*StatusNoContent*) }  func (h \*Handler) getPostComments(c echo.Context) error {  post, ok := c.Get(*postLocals*).(\*model.Post)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get post")  }   skip, limit := skipLimitQuery(c)   comments, err := h.cs.FindByPost(post.ID, skip, limit)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusOK*, comments) }  func (h \*Handler) commentPost(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get requester")  }   if requester.Banned {  return echo.NewHTTPError(http.*StatusForbidden*, "you are banned")  }   post, ok := c.Get(*postLocals*).(\*model.Post)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "failed to get post")  }   req := &struct {  Content string `json:"content" validate:"required,min=1,max=255"`  }{}  if err := c.Bind(req); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   if err := c.Validate(req); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   if err := h.cs.CommentPost(post.ID, requester.ID, req.Content); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.NoContent(http.*StatusNoContent*) }  func (h \*Handler) searchPosts(c echo.Context) error {  skip, limit := skipLimitQuery(c)   query := c.QueryParam(*queryQuery*)  if query == "" {  return echo.NewHTTPError(http.*StatusBadRequest*, "query is required")  }   posts, err := h.ps.Search(query, skip, limit)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusOK*, posts) } |

Створення Proto файлу для grpc.

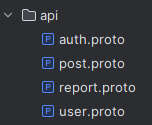


Рисунок 9 — Структура api папки з .proto файлами

Код post.proto:

|  |
| --- |
| syntax = "proto3"; option go\_package = "/gen";  service PostService {  rpc DeleteUserPosts(DeleteUserPostsRequest) returns (DeleteUserPostsResponse);  rpc DeleteUserComments(DeleteUserCommentsRequest) returns (DeleteUserCommentsResponse);  rpc DeleteUserLikes(DeleteUserLikesRequest) returns (DeleteUserLikesResponse); }  message DeleteUserPostsRequest {  uint64 user\_id = 1; }  message DeleteUserPostsResponse {  bool success = 1; }  message DeleteUserCommentsRequest {  uint64 user\_id = 1; }  message DeleteUserCommentsResponse {  bool success = 1; }  message DeleteUserLikesRequest {  uint64 user\_id = 1; }  message DeleteUserLikesResponse {  bool success = 1; } |

Сгенерував grpc файли за допомогоб команди proto в Makefile:

|  |
| --- |
| proto:  @if exist .\pkg\gen (rd /s /q .\pkg\gen)  @mkdir .\pkg\gen  protoc --go\_out=pkg/gen --go-grpc\_out=pkg/gen --go\_opt=paths=source\_relative --go-grpc\_opt=paths=source\_relative api/user.proto  @move /Y .\pkg\gen\api\user.pb.go .\pkg\gen\user.pb.go @move /Y .\pkg\gen\api\user\_grpc.pb.go .\pkg\gen\user\_grpc.pb.go  @rd /s /q .\pkg\gen\api |

Реалізація Grpc сервера з підключенням API v1.

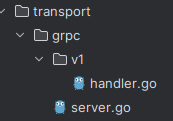


Рисунок 10 — Структура grpc в transport layer

Код server.go:

|  |
| --- |
| package grpc  import (  "context"  "fmt"  "github.com/SocialNetworkY/Backend/internal/post/transport/grpc/v1"  "github.com/SocialNetworkY/Backend/pkg/gen"  "google.golang.org/grpc"  "google.golang.org/grpc/reflection"  "log"  "net"  "time" )  type Server struct {  addr string  grpcServer \*grpc.Server }  func New(port int) \*Server {  log.Printf("Creating grpc server with port: %d", port)  grpcServ := grpc.NewServer(  grpc.UnaryInterceptor(UnaryServerInterceptor()),  )  reflection.Register(grpcServ)   return &Server{  addr: fmt.Sprintf(":%d", port),  grpcServer: grpcServ,  } }  func (s \*Server) Init(ps v1.PostService, cs v1.CommentService, ls v1.LikeService) \*Server {  handler := v1.New(ps, cs, ls)  gen.RegisterPostServiceServer(s.grpcServer, handler)  return s }  func (s \*Server) Run() error {  lis, err := net.Listen("tcp", s.addr)  if err != nil {  return err  }   log.Printf("Grpc server listening at %v", lis.Addr())  return s.grpcServer.Serve(lis) }  // UnaryServerInterceptor for logging func UnaryServerInterceptor() grpc.UnaryServerInterceptor {  return func(  ctx context.Context,  req interface{},  info \*grpc.UnaryServerInfo,  handler grpc.UnaryHandler,  ) (interface{}, error) {  start := time.Now()  h, err := handler(ctx, req)  end := time.Now()   log.Printf("Request - Method:%s\tDuration:%s\tError:%v\n",  info.FullMethod,  end.Sub(start),  err)   return h, err  } } |

Код v1/handler.go:

|  |
| --- |
| package v1  import (  "context"   "github.com/SocialNetworkY/Backend/pkg/gen" )  type (  PostService interface {  DeleteByUser(userID uint) error  }   CommentService interface {  DeleteByUser(userID uint) error  }   LikeService interface {  DeleteByUser(userID uint) error  }   Handler struct {  gen.PostServiceServer  ps PostService  cs CommentService  ls LikeService  } )  func New(ps PostService, cs CommentService, ls LikeService) \*Handler {  return &Handler{  ps: ps,  cs: cs,  ls: ls,  } }  func (h \*Handler) DeleteUserPosts(ctx context.Context, r \*gen.DeleteUserPostsRequest) (\*gen.DeleteUserPostsResponse, error) {  userID := uint(r.GetUserId())  if err := h.ps.DeleteByUser(userID); err != nil {  return nil, err  }   return &gen.DeleteUserPostsResponse{  Success: true,  }, nil }  func (h \*Handler) DeleteUserComments(ctx context.Context, r \*gen.DeleteUserCommentsRequest) (\*gen.DeleteUserCommentsResponse, error) {  userID := uint(r.GetUserId())  if err := h.cs.DeleteByUser(userID); err != nil {  return nil, err  }   return &gen.DeleteUserCommentsResponse{  Success: true,  }, nil }  func (h \*Handler) DeleteUserLikes(ctx context.Context, r \*gen.DeleteUserLikesRequest) (\*gen.DeleteUserLikesResponse, error) {  userID := uint(r.GetUserId())  if err := h.ls.DeleteByUser(userID); err != nil {  return nil, err  }   return &gen.DeleteUserLikesResponse{  Success: true,  }, nil } |

**Завдання №5.** Створення main.go, Dockerfile файлів для запуску, та docker-compose.yml для запуску сервісу з бд

Реалізація User main.go файлів для запуску http, grpc серверів.

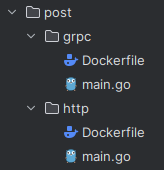


Рисунок 12 — Структура cmd/post папки

Код grpc/Dockerfile:

|  |
| --- |
| FROM golang:latest AS *builder* WORKDIR /app  COPY go.mod ./ COPY go.sum ./  RUN go mod download  COPY .. .  RUN CGO\_ENABLED=0 go build -o /server cmd/post/grpc/main.go  FROM alpine:latest COPY --from=*builder* server . CMD ["/server"] |

Код grpc/main.go:

|  |
| --- |
| package main  import (  "log"   "github.com/SocialNetworkY/Backend/internal/post/elasticsearch"  "github.com/SocialNetworkY/Backend/internal/post/gateway/report"  "github.com/SocialNetworkY/Backend/internal/post/repository"  "gorm.io/driver/mysql"   "github.com/SocialNetworkY/Backend/internal/post/service"  "github.com/SocialNetworkY/Backend/internal/post/transport/grpc"   "github.com/caarlos0/env/v6" )  type Config struct {  DB string `env:"DB"`  Port int `env:"PORT"`  PostElasticSearchAddr string `env:"POST\_ELASTICSEARCH\_ADDR"`  TagElasticSearchAddr string `env:"TAG\_ELASTICSEARCH\_ADDR"`  CommentElasticSearchAddr string `env:"COMMENT\_ELASTICSEARCH\_ADDR"`  ReportServiceHttpAddr string `env:"REPORT\_SERVICE\_HTTP\_ADDR"`  ReportServiceGrpcAddr string `env:"REPORT\_SERVICE\_GRPC\_ADDR"` }  var (  cfg = &Config{} )  func init() {  if err := env.Parse(cfg); err != nil {  log.Fatal(err)  } }  func main() {  postSearch, err := elasticsearch.NewPost(cfg.PostElasticSearchAddr)  if err != nil {  log.Fatalf("Post Elasticsearch err: %v", err)  }   tagSearch, err := elasticsearch.NewTag(cfg.TagElasticSearchAddr)  if err != nil {  log.Fatalf("Tag Elasticsearch err: %v", err)  }   commentSearch, err := elasticsearch.NewComment(cfg.CommentElasticSearchAddr)  if err != nil {  log.Fatalf("Comment Elasticsearch err: %v", err)  }   repos, err := repository.New(mysql.Open(cfg.DB), postSearch, commentSearch, tagSearch)  if err != nil {  log.Fatal(err)  }   reportGateway := report.New(cfg.ReportServiceHttpAddr, cfg.ReportServiceGrpcAddr)  services := service.New(repos.Post, repos.Tag, repos.Like, repos.Comment, reportGateway)   if err := grpc.New(cfg.Port).Init(services.Post, services.Comment, services.Like).Run(); err != nil {  log.Fatalf("Grpc server err: %v", err)  } } |

Код http/Dockerfile:

|  |
| --- |
| FROM golang:latest AS *builder* WORKDIR /apFROM golang:latest AS builder  WORKDIR /app  COPY go.mod ./  COPY go.sum ./  RUN go mod download  COPY . .  RUN CGO\_ENABLED=0 go build -o /server cmd/post/http/main.go  FROM alpine:latest  COPY --from=builder server .  CMD ["/server"]p  COPY go.mod ./ COPY go.sum ./  RUN go mod download  COPY . .  RUN CGO\_ENABLED=0 go build -o /server cmd/post/http/main.go  FROM alpine:latest COPY --from=*builder* server . CMD ["/server"] |

Код http/main.go:

|  |
| --- |
| package main  import (  "fmt"  "log"   "github.com/SocialNetworkY/Backend/internal/post/elasticsearch"  "github.com/SocialNetworkY/Backend/internal/post/gateway/report"  "github.com/SocialNetworkY/Backend/internal/post/gateway/user"  "github.com/SocialNetworkY/Backend/pkg/storage"  "gorm.io/driver/mysql"   "github.com/SocialNetworkY/Backend/internal/post/gateway/auth"  "github.com/SocialNetworkY/Backend/internal/post/repository"  "github.com/SocialNetworkY/Backend/internal/post/service"  "github.com/SocialNetworkY/Backend/internal/post/transport/http"   "github.com/caarlos0/env/v6" )  type Config struct {  DB string `env:"DB"`  Port int `env:"PORT"`  BodyLimit string `env:"BODY\_LIMIT"`  AllowedOrigins []string `env:"ALlOWED\_ORIGINS" envSeparator:","`  PostElasticSearchAddr string `env:"POST\_ELASTICSEARCH\_ADDR"`  TagElasticSearchAddr string `env:"TAG\_ELASTICSEARCH\_ADDR"`  CommentElasticSearchAddr string `env:"COMMENT\_ELASTICSEARCH\_ADDR"`  AuthServiceHttpAddr string `env:"AUTH\_SERVICE\_HTTP\_ADDR"`  AuthServiceGrpcAddr string `env:"AUTH\_SERVICE\_GRPC\_ADDR"`  UserServiceHttpAddr string `env:"USER\_SERVICE\_HTTP\_ADDR"`  UserServiceGrpcAddr string `env:"USER\_SERVICE\_GRPC\_ADDR"`  ReportServiceHttpAddr string `env:"REPORT\_SERVICE\_HTTP\_ADDR"`  ReportServiceGrpcAddr string `env:"REPORT\_SERVICE\_GRPC\_ADDR"`  StorageFolder string `env:"STORAGE\_FOLDER" envDefault:"storage"` }  var (  cfg = &Config{} )  func init() {  if err := env.Parse(cfg); err != nil {  log.Fatal(err)  } }  func main() {  postSearch, err := elasticsearch.NewPost(cfg.PostElasticSearchAddr)  if err != nil {  log.Fatalf("Post Elasticsearch err: %v", err)  }   tagSearch, err := elasticsearch.NewTag(cfg.TagElasticSearchAddr)  if err != nil {  log.Fatalf("Tag Elasticsearch err: %v", err)  }   commentSearch, err := elasticsearch.NewComment(cfg.CommentElasticSearchAddr)  if err != nil {  log.Fatalf("Comment Elasticsearch err: %v", err)  }   repos, err := repository.New(mysql.Open(cfg.DB), postSearch, commentSearch, tagSearch)  if err != nil {  log.Fatal(err)  }   fileStorage, err := storage.NewLocalStorage(cfg.StorageFolder, fmt.Sprintf("http://localhost:%d/%s", cfg.Port, "storage"))  if err != nil {  log.Fatalf("Image storage err: %v", err)  }   authGateway := auth.New(cfg.AuthServiceHttpAddr, cfg.AuthServiceGrpcAddr)  postGateway := user.New(cfg.UserServiceHttpAddr, cfg.UserServiceGrpcAddr)  reportGateway := report.New(cfg.ReportServiceHttpAddr, cfg.ReportServiceGrpcAddr)  services := service.New(repos.Post, repos.Tag, repos.Like, repos.Comment, reportGateway)   if err := http.New(cfg.BodyLimit, cfg.AllowedOrigins, cfg.Port).Init(services.Post, services.Like, services.Comment, authGateway, postGateway, fileStorage).AddStaticFolder("storage", cfg.StorageFolder).Run(); err != nil {  log.Fatalf("Http server err: %v", err)  } } |

Реалізація docker-compose.yml файла для запуску http, grpc сервісів з бд та nginx.

Код docker-compose.yml:

|  |
| --- |
| services:  nginx:  image: nginx:latest  container\_name: nginx  ports:  - "80:80"  volumes:  - ./nginx.conf:/etc/nginx/nginx.conf  networks:  - services  depends\_on:  - auth-http-service  - user-http-service  - post-http-service   elastic:  image: docker.elastic.co/elasticsearch/elasticsearch:8.15.3  container\_name: elastic  environment:  discovery.type: single-node  xpack.security.enabled: false  xpack.security.http.ssl.enabled: false  networks:  - services  volumes:  - elastic:/usr/share/elasticsearch/data   auth-http-service:  build:  context: .  dockerfile: cmd/auth/http/Dockerfile  container\_name: auth-http-service  environment:  DB: myuser:strongpass@tcp(auth-db:3306)/mydb?parseTime=true  PORT: 8080  BODY\_LIMIT: 20MB  ALLOWED\_ORIGINS: "\*"  HASH\_SALT: "hashSalt"  JWT\_SECRET: "accessPass"  JWT\_DURATION: "1h"  JWT\_REFRESH\_SECRET: "refreshPass"  JWT\_REFRESH\_DURATION: "168h"  USER\_SERVICE\_HTTP\_ADDR: http://nginx:80/user  USER\_SERVICE\_GRPC\_ADDR: user-grpc-service:8083  networks:  - services  - auth  depends\_on:  auth-db:  condition: service\_healthy   auth-grpc-service:  build:  context: .  dockerfile: cmd/auth/grpc/Dockerfile  container\_name: auth-grpc-service  environment:  DB: myuser:strongpass@tcp(auth-db:3306)/mydb?parseTime=true  PORT: 8081  HASH\_SALT: "hashSalt"  JWT\_SECRET: "accessPass"  JWT\_DURATION: "1h"  JWT\_REFRESH\_SECRET: "refreshPass"  JWT\_REFRESH\_DURATION: "168h"  USER\_SERVICE\_HTTP\_ADDR: http://nginx:80/user  USER\_SERVICE\_GRPC\_ADDR: user-grpc-service:8083  networks:  - services  - auth  depends\_on:  auth-db:  condition: service\_healthy   auth-db:  image: mysql:latest  container\_name: auth-db  environment:  MYSQL\_DATABASE: mydb  MYSQL\_USER: myuser  MYSQL\_PASSWORD: strongpass  MYSQL\_ROOT\_PASSWORD: verystrongpass  ports:  - "5677:3306"  volumes:  - auth\_db\_data:/var/lib/mysql  networks:  - auth  healthcheck:  test: [ "CMD", "mysqladmin", "ping", "-h", "localhost" ]  timeout: 20s  retries: 10   user-http-service:  build:  context: .  dockerfile: cmd/user/http/Dockerfile  container\_name: user-http-service  environment:  DB: myuser:strongpass@tcp(user-db:3306)/mydb?parseTime=true  PORT: 8082  BODY\_LIMIT: 50MB  ALLOWED\_ORIGINS: "\*"  STORAGE\_FOLDER: ./storage  USER\_ELASTICSEARCH\_ADDR: http://nginx:80/elastic  BAN\_ELASTICSEARCH\_ADDR: http://nginx:80/elastic  AUTH\_SERVICE\_HTTP\_ADDR: http://nginx:80/auth  AUTH\_SERVICE\_GRPC\_ADDR: auth-grpc-service:8081  POST\_SERVICE\_HTTP\_ADDR: http://nginx:80/post  POST\_SERVICE\_GRPC\_ADDR: post-grpc-service:8085  REPORT\_SERVICE\_HTTP\_ADDR: http://nginx:80/report  REPORT\_SERVICE\_GRPC\_ADDR: report-grpc-service:8087  volumes:  - user\_storage:/storage  networks:  - services  - user  depends\_on:  user-db:  condition: service\_healthy   user-grpc-service:  build:  context: .  dockerfile: cmd/user/grpc/Dockerfile  container\_name: user-grpc-service  environment:  DB: myuser:strongpass@tcp(user-db:3306)/mydb?parseTime=true  PORT: 8083  STORAGE\_FOLDER: ./storage  USER\_ELASTICSEARCH\_ADDR: http://nginx:80/elastic  BAN\_ELASTICSEARCH\_ADDR: http://nginx:80/elastic  AUTH\_SERVICE\_HTTP\_ADDR: http://nginx:80/auth  AUTH\_SERVICE\_GRPC\_ADDR: auth-grpc-service:8081  POST\_SERVICE\_HTTP\_ADDR: http://nginx:80/post  POST\_SERVICE\_GRPC\_ADDR: post-grpc-service:8085  REPORT\_SERVICE\_HTTP\_ADDR: http://nginx:80/report  REPORT\_SERVICE\_GRPC\_ADDR: report-grpc-service:8087  volumes:  - user\_storage:/storage  networks:  - services  - user  depends\_on:  user-db:  condition: service\_healthy   user-db:  image: mysql:latest  container\_name: user-db  environment:  MYSQL\_DATABASE: mydb  MYSQL\_USER: myuser  MYSQL\_PASSWORD: strongpass  MYSQL\_ROOT\_PASSWORD: verystrongpass  ports:  - "5678:3306"  volumes:  - user\_db\_data:/var/lib/mysql  networks:  - user  healthcheck:  test: [ "CMD", "mysqladmin", "ping", "-h", "localhost" ]  timeout: 20s  retries: 10   post-http-service:  build:  context: .  dockerfile: cmd/post/http/Dockerfile  container\_name: post-http-service  environment:  DB: myuser:strongpass@tcp(post-db:3306)/mydb?parseTime=true  PORT: 8084  BODY\_LIMIT: 500MB  ALLOWED\_ORIGINS: "\*"  STORAGE\_FOLDER: ./storage  POST\_ELASTICSEARCH\_ADDR: http://nginx:80/elastic  COMMENT\_ELASTICSEARCH\_ADDR: http://nginx:80/elastic  TAG\_ELASTICSEARCH\_ADDR: http://nginx:80/elastic  AUTH\_SERVICE\_HTTP\_ADDR: http://nginx:80/auth  AUTH\_SERVICE\_GRPC\_ADDR: auth-grpc-service:8081  USER\_SERVICE\_HTTP\_ADDR: http://nginx:80/user  USER\_SERVICE\_GRPC\_ADDR: user-grpc-service:8083  REPORT\_SERVICE\_HTTP\_ADDR: http://nginx:80/report  REPORT\_SERVICE\_GRPC\_ADDR: report-grpc-service:8087  volumes:  - post\_storage:/storage  networks:  - services  - post  depends\_on:  post-db:  condition: service\_healthy   post-grpc-service:  build:  context: .  dockerfile: cmd/post/grpc/Dockerfile  container\_name: post-grpc-service  environment:  DB: myuser:strongpass@tcp(post-db:3306)/mydb?parseTime=true  PORT: 8085  POST\_ELASTICSEARCH\_ADDR: http://nginx:80/elastic  COMMENT\_ELASTICSEARCH\_ADDR: http://nginx:80/elastic  TAG\_ELASTICSEARCH\_ADDR: http://nginx:80/elastic  REPORT\_SERVICE\_HTTP\_ADDR: http://nginx:80/report  REPORT\_SERVICE\_GRPC\_ADDR: report-grpc-service:8087  networks:  - services  - post  depends\_on:  post-db:  condition: service\_healthy   post-db:  image: mysql:latest  container\_name: post-db  environment:  MYSQL\_DATABASE: mydb  MYSQL\_USER: myuser  MYSQL\_PASSWORD: strongpass  MYSQL\_ROOT\_PASSWORD: verystrongpass  ports:  - "5679:3306"  volumes:  - post\_db\_data:/var/lib/mysql  networks:  - post  healthcheck:  test: [ "CMD", "mysqladmin", "ping", "-h", "localhost" ]  timeout: 20s  retries: 10   report-http-service:  build:  context: .  dockerfile: cmd/report/http/Dockerfile  container\_name: report-http-service  environment:  DB: myuser:strongpass@tcp(report-db:3306)/mydb?parseTime=true  PORT: 8086  BODY\_LIMIT: 20MB  ALLOWED\_ORIGINS: "\*"  REPORT\_ELASTICSEARCH\_ADDR: http://nginx:80/elastic  AUTH\_SERVICE\_HTTP\_ADDR: http://nginx:80/auth  AUTH\_SERVICE\_GRPC\_ADDR: auth-grpc-service:8081  USER\_SERVICE\_HTTP\_ADDR: http://nginx:80/user  USER\_SERVICE\_GRPC\_ADDR: user-grpc-service:8083  POST\_SERVICE\_HTTP\_ADDR: http://nginx:80/post  POST\_SERVICE\_GRPC\_ADDR: post-grpc-service:8085  networks:  - services  - report  depends\_on:  report-db:  condition: service\_healthy   report-grpc-service:  build:  context: .  dockerfile: cmd/report/grpc/Dockerfile  container\_name: report-grpc-service  environment:  DB: myuser:strongpass@tcp(report-db:3306)/mydb?parseTime=true  PORT: 8087  REPORT\_ELASTICSEARCH\_ADDR: http://nginx:80/elastic  POST\_SERVICE\_HTTP\_ADDR: http://nginx:80/post  POST\_SERVICE\_GRPC\_ADDR: post-grpc-service:8085  networks:  - services  - report  depends\_on:  report-db:  condition: service\_healthy   report-db:  image: mysql:latest  container\_name: report-db  environment:  MYSQL\_DATABASE: mydb  MYSQL\_USER: myuser  MYSQL\_PASSWORD: strongpass  MYSQL\_ROOT\_PASSWORD: verystrongpass  ports:  - "5680:3306"  volumes:  - report\_db\_data:/var/lib/mysql  networks:  - report  healthcheck:  test: [ "CMD", "mysqladmin", "ping", "-h", "localhost" ]  timeout: 20s  retries: 10  volumes:  elastic:  driver: local  driver\_opts:  type: none  o: bind  device: ./elastic  auth\_db\_data:  driver: local  driver\_opts:  type: none  o: bind  device: ./db/auth  user\_db\_data:  driver: local  driver\_opts:  type: none  o: bind  device: ./db/user  user\_storage:  driver: local  driver\_opts:  type: none  o: bind  device: ./storage/user  post\_db\_data:  driver: local  driver\_opts:  type: none  o: bind  device: ./db/post  post\_storage:  driver: local  driver\_opts:  type: none  o: bind  device: ./storage/post  report\_db\_data:  driver: local  driver\_opts:  type: none  o: bind  device: ./db/report  networks:  services:  auth:  user:  post:  report: |

Код nginx.conf:

|  |
| --- |
| events { }  http {  upstream elastic {  server elastic:9200;  }   upstream auth\_http {  server auth-http-service:8080;  }   upstream user\_http {  server user-http-service:8082;  }   upstream post\_http {  server post-http-service:8084;  }   upstream report\_http {  server report-http-service:8086;  }   server {  listen 80;   location /elastic/ {  rewrite ^/elastic/(.\*)$ /$1 break;  proxy\_pass http://elastic;  }   location /auth/ {  rewrite ^/auth/(.\*)$ /$1 break;  proxy\_pass http://auth\_http;  }   location /user/ {  rewrite ^/user/(.\*)$ /$1 break;  proxy\_pass http://user\_http;  }   location /post/ {  rewrite ^/post/(.\*)$ /$1 break;  proxy\_pass http://post\_http;  }   location /report/ {  rewrite ^/report/(.\*)$ /$1 break;  proxy\_pass http://report\_http;  }   } } |