ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення

ЛАБОРАТОРНА РОБОТА №5

з дисципліни «Веб-технології»

.(назва дисципліни)

на тему: «Створення Backend. Report сервіс»

ЛРІПЗс.2401075.05.07.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 репортів (Report сервіс) для забезпечення модерації контенту користувачів. 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 Report сервісу.

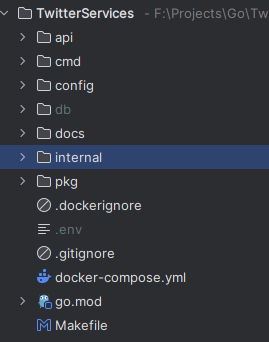


Рисунок 1 — Структура проекту

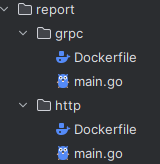


Рисунок 2 — Структура папки post в cmd

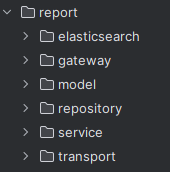


Рисунок 3 — Структура папки post в internal

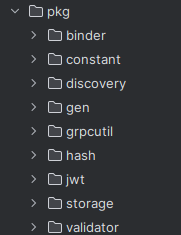


Рисунок 4 — Структура папки pkg

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, Report

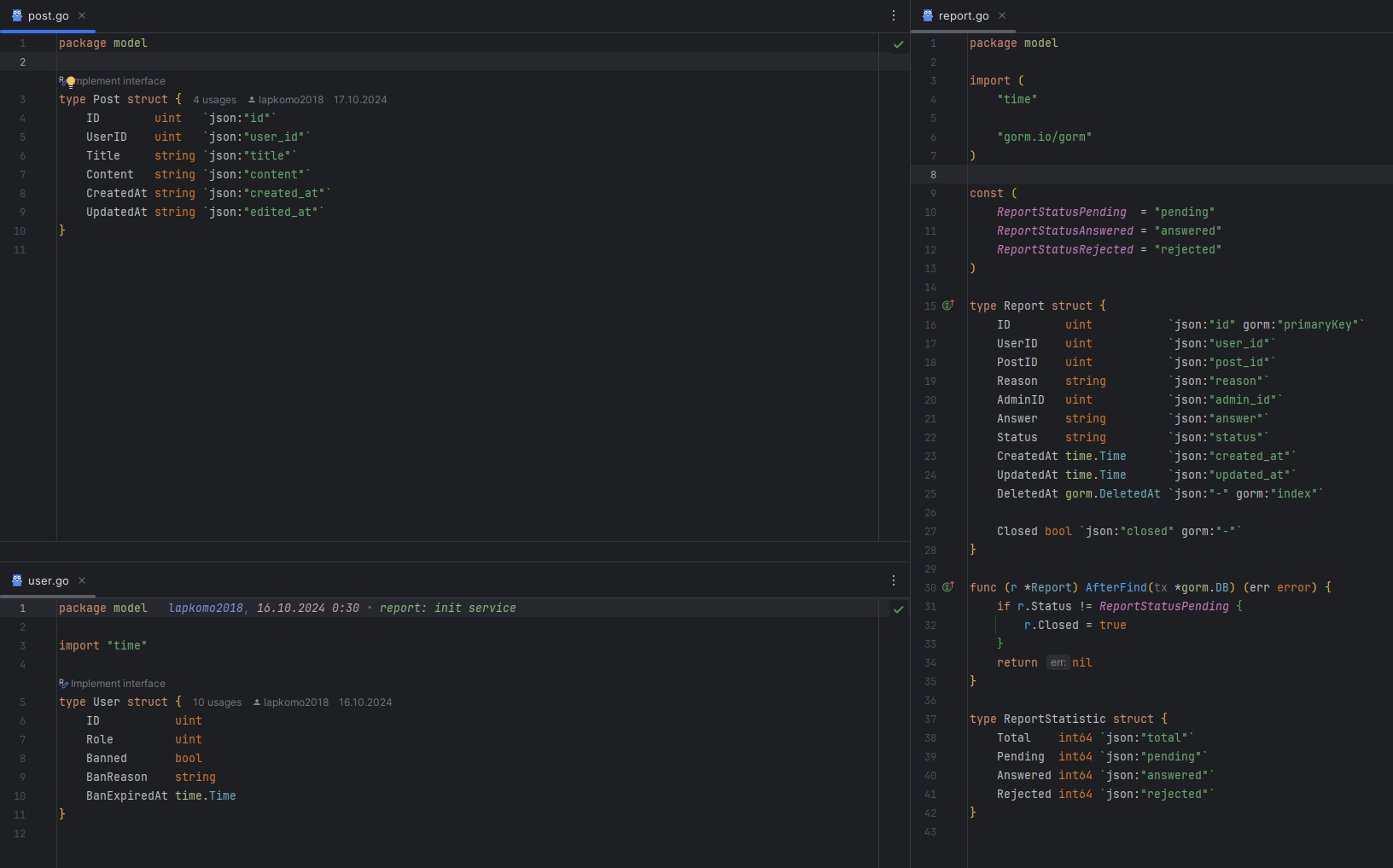
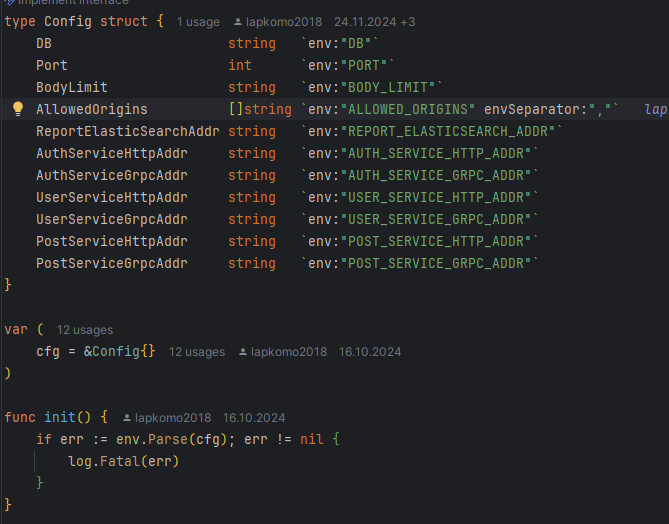


Рисунок 5 — Моделі структур User, Post, Report

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 report.go:

|  |
| --- |
| package repository  import (  "log"   "github.com/SocialNetworkY/Backend/internal/report/model"  "gorm.io/gorm" )  type (  ReportSearch interface {  Index(report \*model.Report) error  Delete(id uint) error  Search(query string, skip, limit int) ([]uint, error)  }   Report struct {  db \*gorm.DB  s ReportSearch  } )  func NewReport(db \*gorm.DB, s ReportSearch) \*Report {  return &Report{  db: db,  s: s,  } }  func (r \*Report) Add(report \*model.Report) error {  log.Printf("Adding report: %v\n", report)  if err := r.db.Create(report).Error; err != nil {  log.Printf("Error adding report: %v\n", err)  return err  }   if err := r.s.Index(report); err != nil {  log.Printf("Error indexing report: %v\n", err)  return err  }  log.Printf("Report added successfully: %v\n", report)  return nil }  func (r \*Report) Save(report \*model.Report) error {  log.Printf("Saving report: %v\n", report)  if err := r.db.Save(report).Error; err != nil {  log.Printf("Error saving report: %v\n", err)  return err  }  if err := r.s.Index(report); err != nil {  log.Printf("Error indexing report: %v\n", err)  return err  }  log.Printf("Report saved successfully: %v\n", report)  return nil }  func (r \*Report) Delete(id uint) error {  log.Printf("Deleting report: %d\n", id)  if err := r.db.Delete(&model.Report{ID: id}).Error; err != nil {  log.Printf("Error deleting report: %v\n", err)  return err  }  if err := r.s.Delete(id); err != nil {  log.Printf("Error deleting report from search index: %v\n", err)  return err  }  log.Printf("Report deleted successfully: %d\n", id)  return nil }  func (r \*Report) Get(id uint) (\*model.Report, error) {  log.Printf("Getting report with ID: %d\n", id)  var report model.Report  err := r.db.First(&report, id).Error  if err != nil {  log.Printf("Error getting report: %v\n", err)  return nil, err  }  log.Printf("Report found: %v\n", report)  return &report, nil }  func (r \*Report) GetByPostUser(postID, userID uint) (\*model.Report, error) {  log.Printf("Getting report for post ID: %d and user ID: %d\n", postID, userID)  var report model.Report  err := r.db.Where("post\_id = ? AND user\_id = ?", postID, userID).First(&report).Error  if err != nil {  log.Printf("Error getting report: %v\n", err)  return nil, err  }  log.Printf("Report found: %v\n", report)  return &report, nil }  func (r \*Report) GetSome(skip, limit int, status string) ([]\*model.Report, error) {  log.Printf("Getting some reports with skip: %d, limit: %d, and status: %s\n", skip, limit, status)  var reports []\*model.Report  if limit < 0 {  skip = 0  }  query := r.db.Offset(skip).Limit(limit)  if status != "" {  query = query.Where("status = ?", status)  }  err := query.Find(&reports).Error  if err != nil {  log.Printf("Error getting some reports: %v\n", err)  return nil, err  }  log.Printf("Reports found: %v\n", reports)  return reports, nil }  func (r \*Report) GetByPost(postID uint, skip, limit int, status string) ([]\*model.Report, error) {  log.Printf("Getting reports for post ID: %d with skip: %d, limit: %d, and status: %s\n", postID, skip, limit, status)  var reports []\*model.Report  if limit < 0 {  skip = 0  }  query := r.db.Where("post\_id = ?", postID).Offset(skip).Limit(limit)  if status != "" {  query = query.Where("status = ?", status)  }  err := query.Find(&reports).Error  if err != nil {  log.Printf("Error getting reports for post: %v\n", err)  return nil, err  }  log.Printf("Reports found: %v\n", reports)  return reports, nil }  func (r \*Report) GetByUser(userID uint, skip, limit int, status string) ([]\*model.Report, error) {  log.Printf("Getting reports for user ID: %d with skip: %d, limit: %d, and status: %s\n", userID, skip, limit, status)  var reports []\*model.Report  if limit < 0 {  skip = 0  }  query := r.db.Where("user\_id = ?", userID).Offset(skip).Limit(limit)  if status != "" {  query = query.Where("status = ?", status)  }  err := query.Find(&reports).Error  if err != nil {  log.Printf("Error getting reports for user: %v\n", err)  return nil, err  }  log.Printf("Reports found: %v\n", reports)  return reports, nil }  func (r \*Report) GetByAdmin(adminID uint, skip, limit int, status string) ([]\*model.Report, error) {  log.Printf("Getting reports for admin ID: %d with skip: %d, limit: %d, and status: %s\n", adminID, skip, limit, status)  var reports []\*model.Report  if limit < 0 {  skip = 0  }  query := r.db.Where("admin\_id", adminID).Offset(skip).Limit(limit)  if status != "" {  query = query.Where("status = ?", status)  }  err := query.Find(&reports).Error  if err != nil {  log.Printf("Error getting reports for admin: %v\n", err)  return nil, err  }  log.Printf("Reports found: %v\n", reports)  return reports, nil }  func (r \*Report) Search(query string, skip, limit int) ([]\*model.Report, error) {  log.Printf("Searching reports with query: %s, skip: %d and limit: %d\n", query, skip, limit)  ids, err := r.s.Search(query, skip, limit)  if err != nil {  log.Printf("Error searching reports: %v\n", err)  return nil, err  }   var reports []\*model.Report  if err := r.db.Where("id IN ?", ids).Find(&reports).Error; err != nil {  log.Printf("Error getting reports: %v\n", err)  return nil, err  }  log.Printf("Found reports: %v\n", reports)  return reports, nil }  func (r \*Report) Statistic() (\*model.ReportStatistic, error) {  log.Println("Getting report statistic")   var stat model.ReportStatistic  err := r.db.Model(&Report{}).  Select("COUNT(\*) AS total, "+  "SUM(CASE WHEN status = ? THEN 1 ELSE 0 END) AS pending, "+  "SUM(CASE WHEN status = ? THEN 1 ELSE 0 END) AS answered, "+  "SUM(CASE WHEN status = ? THEN 1 ELSE 0 END) AS rejected",  model.*ReportStatusPending*, model.*ReportStatusAnswered*, model.*ReportStatusRejected*).  Scan(&stat).Error   if err != nil {  log.Printf("Error getting report statistic: %v\n", err)  return nil, err  }   log.Printf("Report statistic found: %+v\n", stat)  return &stat, nil } |

Код repository.go:

|  |
| --- |
| package repository package repository  import (  "log"   "gorm.io/gorm"  "gorm.io/gorm/logger"   "github.com/SocialNetworkY/Backend/internal/report/model" )  type Repository struct {  Report \*Report }  func New(dialector gorm.Dialector, rs ReportSearch) (\*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.Report{}); err != nil {  return nil, err  }  log.Println("AutoMigrating completed")   return &Repository{  Report: NewReport(db, rs),  }, nil } |

**Завдання №3.** Реалізація service layer

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

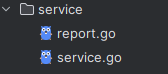


Рисунок 7 — Структура папки service

Код report.go:

|  |
| --- |
| package service  import (  "context"  "errors"  "log"   "github.com/SocialNetworkY/Backend/internal/report/model" )  type (  ReportRepo interface {  Add(report \*model.Report) error  Save(report \*model.Report) error  Delete(id uint) error  Get(id uint) (\*model.Report, error)  GetByPostUser(postID, userID uint) (\*model.Report, error)  GetSome(skip, limit int, status string) ([]\*model.Report, error)  GetByPost(postID uint, skip, limit int, status string) ([]\*model.Report, error)  GetByUser(userID uint, skip, limit int, status string) ([]\*model.Report, error)  GetByAdmin(adminID uint, skip, limit int, status string) ([]\*model.Report, error)  Search(query string, skip, limit int) ([]\*model.Report, error)  Statistic() (\*model.ReportStatistic, error)  }   PostGateway interface {  PostInfo(ctx context.Context, postID uint) (\*model.Post, error)  }   Report struct {  repo ReportRepo  pg PostGateway  } )  func NewReport(repo ReportRepo, pg PostGateway) \*Report {  return &Report{  repo: repo,  pg: pg,  } }  func (r \*Report) Create(userID, postID uint, reason string) (\*model.Report, error) {  log.Printf("Creating report for user ID: %d and post ID: %d with reason: %s\n", userID, postID, reason)  \_, err := r.pg.PostInfo(context.Background(), postID)  if err != nil {  log.Printf("Error fetching post info: %v\n", err)  return nil, errors.New("post not found")  }   \_, err = r.repo.GetByPostUser(postID, userID)  if err == nil {  log.Printf("Report already exists for post ID: %d and user ID: %d\n", postID, userID)  return nil, errors.New("report already exists")  }   report := &model.Report{  UserID: userID,  PostID: postID,  Reason: reason,  Status: model.*ReportStatusPending*,  }   if err := r.repo.Add(report); err != nil {  log.Printf("Error adding report: %v\n", err)  return nil, err  }   log.Printf("Report created successfully: %v\n", report)  return report, nil }  func (r \*Report) Answer(reportID, adminID uint, answer string) (\*model.Report, error) {  log.Printf("Answering report ID: %d by admin ID: %d with answer: %s\n", reportID, adminID, answer)  report, err := r.repo.Get(reportID)  if err != nil {  log.Printf("Error getting report: %v\n", err)  return nil, err  }   report.AdminID = adminID  report.Answer = answer  report.Status = model.*ReportStatusAnswered* report.Closed = true   if err := r.repo.Save(report); err != nil {  log.Printf("Error saving report: %v\n", err)  return nil, err  }   log.Printf("Report answered successfully: %v\n", report)  return report, nil }  func (r \*Report) Reject(reportID, adminID uint, answer string) (\*model.Report, error) {  log.Printf("Rejecting report ID: %d by admin ID: %d with answer: %s\n", reportID, adminID, answer)  report, err := r.repo.Get(reportID)  if err != nil {  log.Printf("Error getting report: %v\n", err)  return nil, err  }   report.AdminID = adminID  report.Answer = answer  report.Status = model.*ReportStatusRejected* report.Closed = true   if err := r.repo.Save(report); err != nil {  log.Printf("Error saving report: %v\n", err)  return nil, err  }   log.Printf("Report rejected successfully: %v\n", report)  return report, nil }  func (r \*Report) Delete(reportID uint) error {  log.Printf("Deleting report ID: %d\n", reportID)  if err := r.repo.Delete(reportID); err != nil {  log.Printf("Error deleting report: %v\n", err)  return err  }  log.Printf("Report deleted successfully: %d\n", reportID)  return nil }  func (r \*Report) DeleteByUser(userID uint) error {  log.Printf("Deleting reports for user ID: %d\n", userID)  reports, err := r.repo.GetByUser(userID, 0, -1, "")  if err != nil {  log.Printf("Error getting reports for user: %v\n", err)  return err  }   for \_, report := range reports {  if err := r.Delete(report.ID); err != nil {  log.Printf("Error deleting report ID: %d\n", report.ID)  return err  }  }   log.Printf("Reports deleted successfully for user ID: %d\n", userID)  return nil }  func (r \*Report) DeleteByPost(postID uint) error {  log.Printf("Deleting reports for post ID: %d\n", postID)  reports, err := r.repo.GetByPost(postID, 0, -1, "")  if err != nil {  log.Printf("Error getting reports for post: %v\n", err)  return err  }   for \_, report := range reports {  if err := r.Delete(report.ID); err != nil {  log.Printf("Error deleting report ID: %d\n", report.ID)  return err  }  }   log.Printf("Reports deleted successfully for post ID: %d\n", postID)  return nil }  func (r \*Report) Get(reportID uint) (\*model.Report, error) {  log.Printf("Getting report ID: %d\n", reportID)  report, err := r.repo.Get(reportID)  if err != nil {  log.Printf("Error getting report: %v\n", err)  return nil, err  }  log.Printf("Report found: %v\n", report)  return report, nil }  func (r \*Report) GetSome(skip, limit int, status string) ([]\*model.Report, error) {  log.Printf("Getting some reports with skip: %d, limit: %d, and status: %s\n", skip, limit, status)  reports, err := r.repo.GetSome(skip, limit, status)  if err != nil {  log.Printf("Error getting some reports: %v\n", err)  return nil, err  }  log.Printf("Reports found: %v\n", reports)  return reports, nil }  func (r \*Report) GetByPost(postID uint, skip, limit int, status string) ([]\*model.Report, error) {  log.Printf("Getting reports for post ID: %d with skip: %d, limit: %d, and status: %s\n", postID, skip, limit, status)  reports, err := r.repo.GetByPost(postID, skip, limit, status)  if err != nil {  log.Printf("Error getting reports for post: %v\n", err)  return nil, err  }  log.Printf("Reports found: %v\n", reports)  return reports, nil }  func (r \*Report) GetByUser(userID uint, skip, limit int, status string) ([]\*model.Report, error) {  log.Printf("Getting reports for user ID: %d with skip: %d, limit: %d, and status: %s\n", userID, skip, limit, status)  reports, err := r.repo.GetByUser(userID, skip, limit, status)  if err != nil {  log.Printf("Error getting reports for user: %v\n", err)  return nil, err  }  log.Printf("Reports found: %v\n", reports)  return reports, nil }  func (r \*Report) GetByAdmin(adminID uint, skip, limit int, status string) ([]\*model.Report, error) {  log.Printf("Getting reports for admin ID: %d with skip: %d, limit: %d, and status: %s\n", adminID, skip, limit, status)  reports, err := r.repo.GetByAdmin(adminID, skip, limit, status)  if err != nil {  log.Printf("Error getting reports for admin: %v\n", err)  return nil, err  }  log.Printf("Reports found: %v\n", reports)  return reports, nil }  func (r \*Report) Search(query string, skip, limit int) ([]\*model.Report, error) {  log.Printf("Searching reports with query: %s, skip: %d and limit: %d\n", query, skip, limit)  reports, err := r.repo.Search(query, skip, limit)  if err != nil {  log.Printf("Error searching reports: %v\n", err)  return nil, err  }  log.Printf("Reports found: %v\n", reports)  return reports, nil }  func (r \*Report) Statistic() (\*model.ReportStatistic, error) {  return r.repo.Statistic() } |

Код service.go:

|  |
| --- |
| package service  type Service struct {  Report \*Report }  func New(reportRepo ReportRepo, pg PostGateway) \*Service {  r := NewReport(reportRepo, pg)   return &Service{  Report: r,  } } |

**Завдання №4.** Реалізація transport layer

Реалізація http сервера з підключенням API v1.

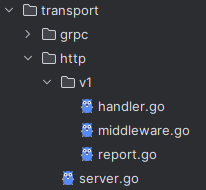
****

Рисунок 8 — Структура http в transport layer

Код server.go:

|  |
| --- |
| package http  import (  "fmt"  "log"  "net/http"   "github.com/SocialNetworkY/Backend/internal/report/transport/http/v1"   "github.com/SocialNetworkY/Backend/pkg/binder"  "github.com/SocialNetworkY/Backend/pkg/validator"   "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,  AllowCredentials: true,  }  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),  } }  func (s \*Server) Init(rs v1.ReportService, ag v1.AuthGateway, ug v1.UserGateway) \*Server {  log.Println("Initializing server...")  log.Println("Initializing api...")  handlerV1 := v1.New(rs, ag, ug)  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"  "log"   "github.com/SocialNetworkY/Backend/internal/report/model"  "github.com/labstack/echo/v4" )  type (  ReportService interface {  Create(userID, postID uint, reason string) (\*model.Report, error)  Answer(reportID, adminID uint, answer string) (\*model.Report, error)  Reject(reportID, adminID uint, answer string) (\*model.Report, error)  Get(reportID uint) (\*model.Report, error)  GetSome(skip, limit int, status string) ([]\*model.Report, error)  GetByPost(postID uint, skip, limit int, status string) ([]\*model.Report, error)  GetByUser(userID uint, skip, limit int, status string) ([]\*model.Report, error)  GetByAdmin(adminID uint, skip, limit int, status string) ([]\*model.Report, error)  Delete(reportID uint) error  Search(query string, skip, limit int) ([]\*model.Report, error)  Statistic() (\*model.ReportStatistic, error)  }   AuthGateway interface {  Authenticate(ctx context.Context, auth string) (uint, error)  }   UserGateway interface {  UserInfo(ctx context.Context, userID uint) (\*model.User, error)  }   Handler struct {  rs ReportService  ag AuthGateway  ug UserGateway  } )  func New(rs ReportService, ag AuthGateway, ug UserGateway) \*Handler {  return &Handler{  rs: rs,  ag: ag,  ug: ug,  } }  func (h \*Handler) Init(api \*echo.Group) {  log.Println("Initializing V1 api")  v1 := api.Group("/v1")  {  h.initReportApi(v1)  } } |

Код v1/middleware.go:

|  |
| --- |
| package v1  import (  "fmt"  "net/http"  "strconv"   "github.com/SocialNetworkY/Backend/internal/report/model"  "github.com/SocialNetworkY/Backend/pkg/constant"  "github.com/labstack/echo/v4" )  const (  *reportIDParam* = "report\_id"   *reportLocals* = "report"  *requesterLocals* = "requester"   *statusQuery* = "status"  *queryQuery* = "query"  *skipQuery* = "skip"  *limitQuery* = "limit"   *defaultSkip* = 0  *defaultLimit* = 10 )  func (h \*Handler) authMiddleware(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) 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) setReportByIDMiddleware(next echo.HandlerFunc) echo.HandlerFunc {  return func(c echo.Context) error {  reportID, err := getUintParam(c, *reportIDParam*)  if err != nil {  return err  }   report, err := h.rs.Get(reportID)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   c.Set(*reportLocals*, report)   return next(c)  } }  func (h \*Handler) checkAccessMiddleware(next echo.HandlerFunc) echo.HandlerFunc {  return func(c echo.Context) error {  report, ok := c.Get(*reportLocals*).(\*model.Report)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "failed to get report")  }   requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "failed to get requester")  }   if report.UserID != requester.ID && requester.Role < constant.*RoleAdminLvl1* {  return echo.NewHTTPError(http.*StatusForbidden*, "You don't have access to this report")   }   return next(c)  } }  func (h \*Handler) adminMiddleware(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.Role < constant.*RoleAdminLvl1* {  return echo.NewHTTPError(http.*StatusForbidden*, "You don't have access to this report")  }   return next(c)  } }  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 }  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 } |

Код v1/report.go:

|  |
| --- |
| package v1  import (  "fmt"  "net/http"   "github.com/SocialNetworkY/Backend/internal/report/model"  "github.com/labstack/echo/v4" )  func (h \*Handler) initReportApi(api \*echo.Group) {  reports := api.Group("/reports", h.authMiddleware, h.banMiddleware)  {  reports.POST("", h.postReport)  reports.GET("/search", h.search, h.adminMiddleware)  reports.GET("/stats", h.getStats, h.adminMiddleware)  reports.GET("", h.getReports, h.adminMiddleware)   report := reports.Group(fmt.Sprintf("/:%s", *reportIDParam*), h.setReportByIDMiddleware, h.checkAccessMiddleware)  {  report.GET("", h.get)  report.DELETE("", h.delete, h.adminMiddleware)  report.POST("/answer", h.answer, h.adminMiddleware)  report.POST("/reject", h.reject, h.adminMiddleware)  }  } }  func (h \*Handler) postReport(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "failed to get requester")  }   req := &struct {  PostID uint `form:"post\_id" validate:"required"`  Reason string `form:"reason" 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())  }   report, err := h.rs.Create(requester.ID, req.PostID, req.Reason)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusCreated*, report) }  func (h \*Handler) get(c echo.Context) error {  report, ok := c.Get(*reportLocals*).(\*model.Report)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "failed to get report")  }   return c.JSON(http.*StatusOK*, report) }  func (h \*Handler) delete(c echo.Context) error {  report, ok := c.Get(*reportLocals*).(\*model.Report)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "failed to get report")  }   if err := h.rs.Delete(report.ID); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.NoContent(http.*StatusNoContent*) }  func (h \*Handler) answer(c echo.Context) error {  report, ok := c.Get(*reportLocals*).(\*model.Report)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "failed to get report")  }   if report.Closed {  return echo.NewHTTPError(http.*StatusForbidden*, "report is closed")  }   requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "failed to get requester")  }   req := &struct {  Answer string `form:"answer" 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())  }   report, err := h.rs.Answer(report.ID, requester.ID, req.Answer)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusOK*, report) }  func (h \*Handler) reject(c echo.Context) error {  report, ok := c.Get(*reportLocals*).(\*model.Report)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "failed to get report")  }   if report.Closed {  return echo.NewHTTPError(http.*StatusForbidden*, "report is closed")  }   requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "failed to get requester")  }   req := &struct {  Answer string `form:"answer" 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())  }   report, err := h.rs.Reject(report.ID, requester.ID, req.Answer)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusOK*, report) }  func (h \*Handler) search(c echo.Context) error {  skip, limit := skipLimitQuery(c)   query := c.QueryParam(*queryQuery*)  if query == "" {  return echo.NewHTTPError(http.*StatusBadRequest*, "query is required")  }   reports, err := h.rs.Search(query, skip, limit)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusOK*, reports) }  func (h \*Handler) getStats(c echo.Context) error {  stats, err := h.rs.Statistic()  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }  return c.JSON(http.*StatusOK*, stats) }  func (h \*Handler) getReports(c echo.Context) error {  skip, limit := skipLimitQuery(c)   reports, err := h.rs.GetSome(skip, limit, c.QueryParam(*statusQuery*))  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusOK*, reports) } |

Створення Proto файлу для grpc.

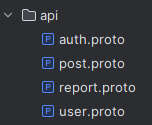


Рисунок 9 — Структура api папки з .proto файлами

Код report.proto:

|  |
| --- |
| syntax = "proto3"; option go\_package = "/gen";  service ReportService {  rpc DeleteUserReports(DeleteUserReportsRequest) returns (DeleteUserReportsResponse);  rpc DeletePostReports(DeletePostReportsRequest) returns (DeletePostReportsResponse); }  message DeleteUserReportsRequest {  uint64 user\_id = 1; }  message DeleteUserReportsResponse {  bool success = 1; }  message DeletePostReportsRequest {  uint64 post\_id = 1; }  message DeletePostReportsResponse {  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"  v1 "github.com/SocialNetworkY/Backend/internal/report/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(rs v1.ReportService) \*Server {  handler := v1.New(rs)  gen.RegisterReportServiceServer(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 (  ReportService interface {  DeleteByUser(userID uint) error  DeleteByPost(postID uint) error  }   Handler struct {  gen.ReportServiceServer  rs ReportService  } )  func New(rs ReportService) \*Handler {  return &Handler{  rs: rs,  } }  func (h \*Handler) DeleteUserReports(ctx context.Context, r \*gen.DeleteUserReportsRequest) (\*gen.DeleteUserReportsResponse, error) {  userID := uint(r.GetUserId())  if err := h.rs.DeleteByUser(userID); err != nil {  return nil, err  }   return &gen.DeleteUserReportsResponse{  Success: true,  }, nil }  func (h \*Handler) DeletePostReports(ctx context.Context, r \*gen.DeletePostReportsRequest) (\*gen.DeletePostReportsResponse, error) {  postID := uint(r.GetPostId())  if err := h.rs.DeleteByPost(postID); err != nil {  return nil, err  }   return &gen.DeletePostReportsResponse{  Success: true,  }, nil } |

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

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

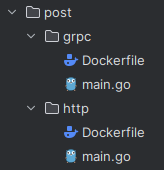


Рисунок 12 — Структура cmd/report папки

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/report/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/report/elasticsearch"  "github.com/SocialNetworkY/Backend/internal/report/gateway/post"   "github.com/SocialNetworkY/Backend/internal/report/repository"  "github.com/SocialNetworkY/Backend/internal/report/service"  "github.com/SocialNetworkY/Backend/internal/report/transport/grpc"   "github.com/caarlos0/env/v6"  "gorm.io/driver/mysql" )  type Config struct {  DB string `env:"DB"`  Port int `env:"PORT"`  ReportElasticSearchAddr string `env:"REPORT\_ELASTICSEARCH\_ADDR"`  PostServiceHttpAddr string `env:"POST\_SERVICE\_HTTP\_ADDR"`  PostServiceGrpcAddr string `env:"POST\_SERVICE\_GRPC\_ADDR"` }  var (  cfg = &Config{} )  func init() {  if err := env.Parse(cfg); err != nil {  log.Fatal(err)  } }  func main() {  reportSearch, err := elasticsearch.NewReport(cfg.ReportElasticSearchAddr)  if err != nil {  log.Fatal(err)  }   repos, err := repository.New(mysql.Open(cfg.DB), reportSearch)  if err != nil {  log.Fatal(err)  }   postGateway := post.New(cfg.PostServiceHttpAddr, cfg.PostServiceGrpcAddr)  services := service.New(repos.Report, postGateway)   if err := grpc.New(cfg.Port).Init(services.Report).Run(); err != nil {  log.Fatalf("Grpc server err: %v", err)  } } |

Код http/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/report/http/main.go  FROM alpine:latest COPY --from=*builder* server . CMD ["/server"] |

Код http/main.go:

|  |
| --- |
| package main  import (  "log"   "github.com/SocialNetworkY/Backend/internal/report/elasticsearch"  "github.com/SocialNetworkY/Backend/internal/report/gateway/post"  "github.com/SocialNetworkY/Backend/internal/report/transport/http"   "github.com/SocialNetworkY/Backend/internal/report/gateway/auth"  "github.com/SocialNetworkY/Backend/internal/report/gateway/user"  "github.com/SocialNetworkY/Backend/internal/report/repository"  "github.com/SocialNetworkY/Backend/internal/report/service"  "github.com/caarlos0/env/v6"  "gorm.io/driver/mysql" )  type Config struct {  DB string `env:"DB"`  Port int `env:"PORT"`  BodyLimit string `env:"BODY\_LIMIT"`  AllowedOrigins []string `env:"ALLOWED\_ORIGINS" envSeparator:","`  ReportElasticSearchAddr string `env:"REPORT\_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"`  PostServiceHttpAddr string `env:"POST\_SERVICE\_HTTP\_ADDR"`  PostServiceGrpcAddr string `env:"POST\_SERVICE\_GRPC\_ADDR"` }  var (  cfg = &Config{} )  func init() {  if err := env.Parse(cfg); err != nil {  log.Fatal(err)  } }  func main() {  reportSearch, err := elasticsearch.NewReport(cfg.ReportElasticSearchAddr)  if err != nil {  log.Fatal(err)  }   repos, err := repository.New(mysql.Open(cfg.DB), reportSearch)  if err != nil {  log.Fatal(err)  }   authGateway := auth.New(cfg.AuthServiceHttpAddr, cfg.AuthServiceGrpcAddr)  userGateway := user.New(cfg.UserServiceHttpAddr, cfg.UserServiceGrpcAddr)  postGateway := post.New(cfg.PostServiceHttpAddr, cfg.PostServiceGrpcAddr)  services := service.New(repos.Report, postGateway)   if err := http.New(cfg.BodyLimit, cfg.AllowedOrigins, cfg.Port).Init(services.Report, authGateway, userGateway).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  - report-http-service   elastic:  image: docker.elastic.co/elasticsearch/elasticsearch:8.15.3  container\_name: elastic  environment:  discovery.type: single-node  cluster.routing.allocation.disk.watermark.low: 10gb  cluster.routing.allocation.disk.watermark.high: 5gb  cluster.routing.allocation.disk.watermark.flood\_stage: 1gb  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: "http://localhost:3000"  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  ports:  - 8080:8080  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: "http://localhost:3000"  STORAGE\_FOLDER: ./storage  USER\_ELASTICSEARCH\_ADDR: http://elastic:9200  BAN\_ELASTICSEARCH\_ADDR: http://elastic:9200  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  ports:  - "8082:8082"  depends\_on:  user-db:  condition: service\_healthy  elastic:  condition: service\_started   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://elastic:9200  BAN\_ELASTICSEARCH\_ADDR: http://elastic:9200  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  elastic:  condition: service\_started   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: "http://localhost:3000"  STORAGE\_FOLDER: ./storage  POST\_ELASTICSEARCH\_ADDR: http://elastic:9200  COMMENT\_ELASTICSEARCH\_ADDR: http://elastic:9200  TAG\_ELASTICSEARCH\_ADDR: http://elastic:9200  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  ports:  - "8084:8084"  depends\_on:  post-db:  condition: service\_healthy  elastic:  condition: service\_started   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://elastic:9200  COMMENT\_ELASTICSEARCH\_ADDR: http://elastic:9200  TAG\_ELASTICSEARCH\_ADDR: http://elastic:9200  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  elastic:  condition: service\_started   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: "http://localhost:3000"  REPORT\_ELASTICSEARCH\_ADDR: http://elastic:9200  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  ports:  - "8086:8086"  depends\_on:  report-db:  condition: service\_healthy  elastic:  condition: service\_started   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://elastic:9200  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  elastic:  condition: service\_started   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 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 /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;  }   } } |