ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення

ЛАБОРАТОРНА РОБОТА №3

з дисципліни «Веб-технології»

.(назва дисципліни)

на тему: «Створення Backend. User сервіс»

ЛРІПЗс.2401075.03.07.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 користувачів (User сервіс) для забезпечення безпечного доступу користувачів до системи. Реалізувати CRUD функціонал та бани, repository layer, service layer та transport layer з rest та grpc для коректної взаємодії з іншими мікросервісами.

**ХІД РОБОТИ**

**Завдання №1.** Підготовка проекту

Ініціалізація Go-проекту за допомогою go mod init.

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

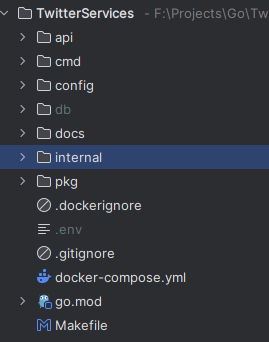


Рисунок 1 — Структура проекту

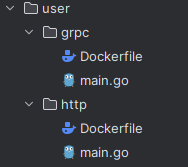


Рисунок 2 — Структура папки cmd

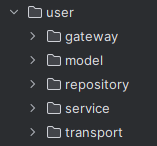


Рисунок 3 — Структура папки internal

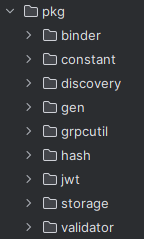


Рисунок 4 — Структура папки pkg

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, Ban

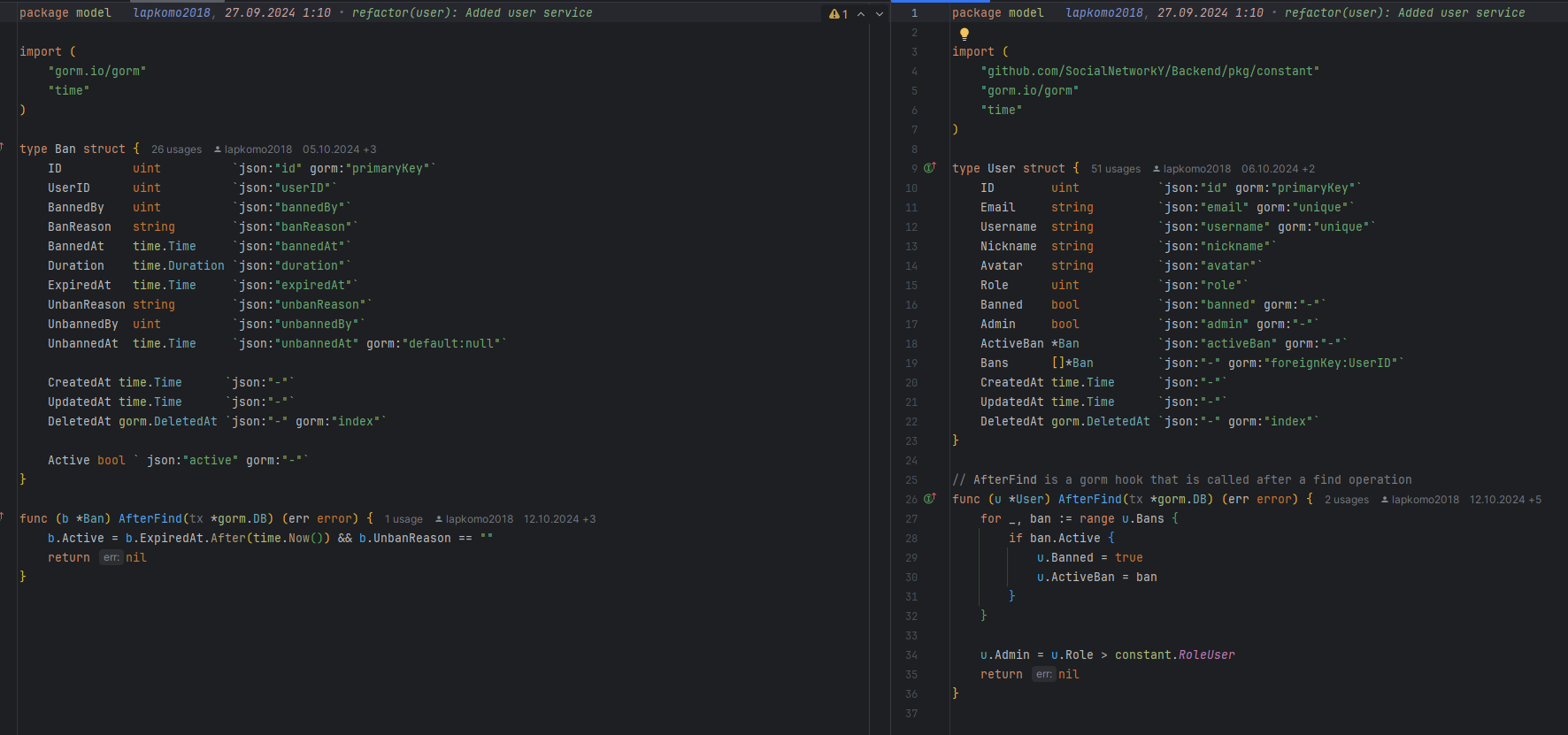
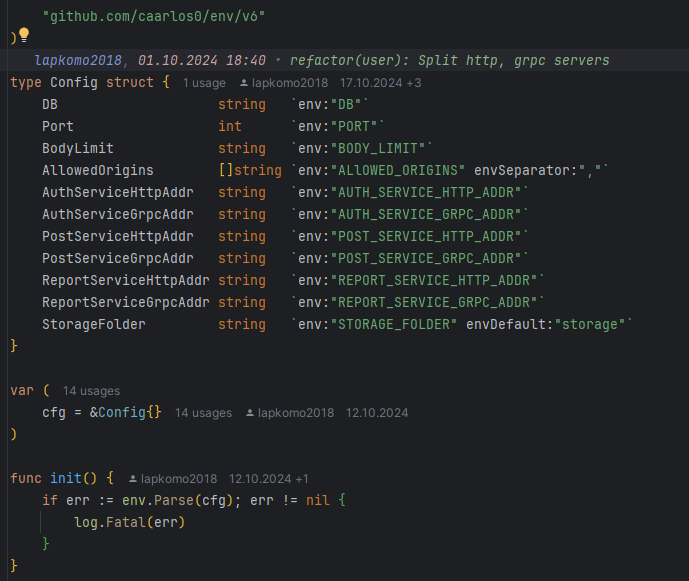


Рисунок 5 — Моделі структур User, Ban

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).

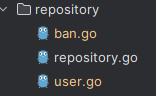


Рисунок 6 — Структура папки repository

Код user.go:

|  |
| --- |
| package repository  import (  "github.com/SocialNetworkY/Backend/internal/user/model"  "gorm.io/gorm" )  type UserRepository struct {  db \*gorm.DB }  func NewUserRepository(db \*gorm.DB) \*UserRepository {  return &UserRepository{  db: db,  } }  func (ur \*UserRepository) Add(user \*model.User) error {  if err := ur.db.Create(user).Error; err != nil {  return err  }  return nil }  func (ur \*UserRepository) Save(user \*model.User) error {  if err := ur.db.Save(user).Error; err != nil {  return err  }  return nil }  func (ur \*UserRepository) Delete(user \*model.User) error {  if err := ur.db.Delete(user).Error; err != nil {  return err  }  return nil }  func (ur \*UserRepository) ExistsByLogin(login string) (bool, error) {  exists, err := ur.ExistsByEmail(login)  if err != nil {  return false, err  }  if exists {  return true, nil  }   exists, err = ur.ExistsByUsername(login)  if err != nil {  return false, err  }  return exists, nil }  func (ur \*UserRepository) ExistsByEmail(email string) (bool, error) {  var count int64  err := ur.db.Model(&model.User{}).Where("email = ?", email).Count(&count).Error  if err != nil {  return false, err  }  return count > 0, nil }  func (ur \*UserRepository) ExistsByUsername(username string) (bool, error) {  var count int64  err := ur.db.Model(&model.User{}).Where("username = ?", username).Count(&count).Error  if err != nil {  return false, err  }  return count > 0, nil }  func (ur \*UserRepository) FindByLogin(login string) (\*model.User, error) {  user, err := ur.FindByEmail(login)  if err == nil {  return user, nil  }   user, err = ur.FindByUsername(login)  if err == nil {  return user, nil  }   return nil, err }  func (ur \*UserRepository) Find(id uint) (\*model.User, error) {  user := &model.User{}  if err := ur.db.Preload("Bans").Where("id = ?", id).First(user).Error; err != nil {  return nil, err  }  return user, nil }  func (ur \*UserRepository) FindSome(skip, limit int) ([]\*model.User, error) {  var users []\*model.User  if limit < 0 {  skip = -1  }  if err := ur.db.Preload("Bans").Offset(skip).Limit(limit).Find(&users).Error; err != nil {  return nil, err  }  return users, nil }  func (ur \*UserRepository) FindByUsername(username string) (\*model.User, error) {  user := &model.User{}  if err := ur.db.Preload("Bans").Where("username = ?", username).First(user).Error; err != nil {  return nil, err  }  return user, nil }  func (ur \*UserRepository) FindByEmail(email string) (\*model.User, error) {  user := &model.User{}  if err := ur.db.Preload("Bans").Where("email = ?", email).First(user).Error; err != nil {  return nil, err  }  return user, nil }  func (ur \*UserRepository) FindByNickname(nickname string, skip, limit int) ([]\*model.User, error) {  var users []\*model.User  if limit < 0 {  skip = -1  }  if err := ur.db.Preload("Bans").Where("nickname = ?", nickname).Offset(skip).Limit(limit).Find(&users).Error; err != nil {  return nil, err  }  return users, nil } |

Код ban.go:

|  |
| --- |
| package mysql package repository  import (  "github.com/SocialNetworkY/Backend/internal/user/model"  "gorm.io/gorm" )  type BanRepository struct {  db \*gorm.DB }  func NewBanRepository(db \*gorm.DB) \*BanRepository {  return &BanRepository{  db: db,  } }  func (br \*BanRepository) Add(ban \*model.Ban) error {  if err := br.db.Create(ban).Error; err != nil {  return err  }  return nil }  func (br \*BanRepository) Save(ban \*model.Ban) error {  if err := br.db.Save(ban).Error; err != nil {  return err  }  return nil }  func (br \*BanRepository) Delete(id uint) error {  if err := br.db.Delete(&model.Ban{}, id).Error; err != nil {  return err  }  return nil }  func (br \*BanRepository) Find(id uint) (\*model.Ban, error) {  ban := &model.Ban{}  if err := br.db.First(ban, id).Error; err != nil {  return nil, err  }  return ban, nil }  func (br \*BanRepository) FindSome(skip, limit int) ([]\*model.Ban, error) {  var bans []\*model.Ban  if limit < 0 {  skip = -1  }  if err := br.db.Offset(skip).Limit(limit).Find(&bans).Error; err != nil {  return nil, err  }  return bans, nil }  func (br \*BanRepository) FindByUser(userID uint, skip, limit int) ([]\*model.Ban, error) {  var bans []\*model.Ban  if limit < 0 {  skip = -1  }  if err := br.db.Offset(skip).Where("user\_id = ?", userID).Limit(limit).Find(&bans).Error; err != nil {  return nil, err  }  return bans, nil }  func (br \*BanRepository) FindByAdmin(adminID uint, skip, limit int) ([]\*model.Ban, error) {  var bans []\*model.Ban  if limit < 0 {  skip = -1  }  if err := br.db.Offset(skip).Where("admin\_id = ?", adminID).Limit(limit).Find(&bans).Error; err != nil {  return nil, err  }  return bans, nil } |

Код repository.go:

|  |
| --- |
| package repository  import (  "log"   "gorm.io/gorm"  "gorm.io/gorm/logger"   "github.com/SocialNetworkY/Backend/internal/user/model" )  type Repository struct {  User \*UserRepository  Ban \*BanRepository }  func New(dialector gorm.Dialector) (\*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.User{}, &model.Ban{}); err != nil {  return nil, err  }  log.Println("AutoMigrating completed")   return &Repository{  User: NewUserRepository(db),  Ban: NewBanRepository(db),  }, nil } |

**Завдання №3.** Реалізація service layer

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

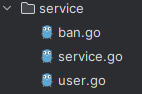


Рисунок 7 — Структура папки service

Код ban.go:

|  |
| --- |
| package service  import (  "context"  "github.com/SocialNetworkY/Backend/internal/user/model"  "time" )  type (  BanRepo interface {  Add(ban \*model.Ban) error  Save(ban \*model.Ban) error  Delete(id uint) error  Find(id uint) (\*model.Ban, error)  FindByUser(userID uint, skip, limit int) ([]\*model.Ban, error)  FindByAdmin(adminID uint, skip, limit int) ([]\*model.Ban, error)  }   BanService struct {  repo BanRepo  pg PostGateway  } )  func NewBanService(r BanRepo, pg PostGateway) \*BanService {  return &BanService{  repo: r,  pg: pg,  } }  // Ban bans a user with time and reason func (bs \*BanService) Ban(userID, adminID uint, reason string, duration time.Duration) error {  ban := &model.Ban{  UserID: userID,  BannedBy: adminID,  BanReason: reason,  BannedAt: time.Now(),  Duration: duration,  ExpiredAt: time.Now().Add(duration),  }   if err := bs.pg.DeleteUserPosts(context.Background(), userID); err != nil {  return err  }   if err := bs.pg.DeleteUserComments(context.Background(), userID); err != nil {  return err  }   if err := bs.pg.DeleteUserLikes(context.Background(), userID); err != nil {  return err  }   return bs.repo.Add(ban) }  // Unban unbans a user with reason func (bs \*BanService) Unban(banID, adminID uint, reason string) error {  ban, err := bs.repo.Find(banID)  if err != nil {  return err  }   ban.UnbanReason = reason  ban.UnbannedBy = adminID  ban.UnbannedAt = time.Now()  return bs.repo.Save(ban) }  // Find returns a ban by id func (bs \*BanService) Find(id uint) (\*model.Ban, error) {  return bs.repo.Find(id) }  // FindByUser returns all bans for a user func (bs \*BanService) FindByUser(userID uint, skip, limit int) ([]\*model.Ban, error) {  return bs.repo.FindByUser(userID, skip, limit) } |

Код user.go:

|  |
| --- |
| package service  import (  "context"  "errors"  "fmt"  "github.com/SocialNetworkY/Backend/internal/user/model"  "io" )  type (  UserRepo interface {  Add(user \*model.User) error  Save(user \*model.User) error  Delete(user \*model.User) error  ExistsByLogin(login string) (bool, error)  ExistsByEmail(email string) (bool, error)  ExistsByUsername(username string) (bool, error)  FindByLogin(login string) (\*model.User, error)  Find(id uint) (\*model.User, error)  FindSome(skip, limit int) ([]\*model.User, error)  FindByUsername(username string) (\*model.User, error)  FindByEmail(email string) (\*model.User, error)  FindByNickname(nickname string, skip, limit int) ([]\*model.User, error)  }   AuthGateway interface {  UpdateUsernameEmail(ctx context.Context, id uint, username, email string) error  DeleteUser(ctx context.Context, id uint) error  }   PostGateway interface {  DeleteUserPosts(ctx context.Context, id uint) error  DeleteUserComments(ctx context.Context, id uint) error  DeleteUserLikes(ctx context.Context, id uint) error  }   ReportGateway interface {  DeleteUserReports(ctx context.Context, userID uint) error  }   ImageStorage interface {  UploadImage(file io.ReadSeeker, filename string) (string, error)  }   UserService struct {  repo UserRepo  ag AuthGateway  pg PostGateway  rg ReportGateway  is ImageStorage  } )  var (  ErrUserUsernameTaken = errors.New("username is already taken")  ErrUserEmailTaken = errors.New("email is already taken") )  func NewUserService(repo UserRepo, is ImageStorage, ag AuthGateway, pg PostGateway, rg ReportGateway) \*UserService {  return &UserService{  repo: repo,  ag: ag,  pg: pg,  rg: rg,  is: is,  } }  func (us \*UserService) Create(id, role uint, username, email string) (\*model.User, error) {  exists, err := us.repo.ExistsByUsername(username)  switch {  case err != nil:  return nil, err  case exists:  return nil, ErrUserUsernameTaken  }   exists, err = us.repo.ExistsByEmail(email)  switch {  case err != nil:  return nil, err  case exists:  return nil, ErrUserEmailTaken  }   user := &model.User{  ID: id,  Email: email,  Username: username,  Nickname: username,  Role: role,  }   if err := us.repo.Add(user); err != nil {  return nil, err  }   return user, nil }  func (us \*UserService) Find(id uint) (\*model.User, error) {  return us.repo.Find(id) }  func (us \*UserService) FindByUsername(username string) (\*model.User, error) {  return us.repo.FindByUsername(username) }  func (us \*UserService) FindByEmail(email string) (\*model.User, error) {  return us.repo.FindByEmail(email) }  func (us \*UserService) FindByNickname(nickname string, skip, limit int) ([]\*model.User, error) {  return us.repo.FindByNickname(nickname, skip, limit) }  func (us \*UserService) ChangeEmail(id uint, email string) error {  user, err := us.repo.Find(id)  if err != nil {  return err  }   exists, err := us.repo.ExistsByEmail(email)  switch {  case err != nil:  return err  case exists:  return ErrUserEmailTaken  }   user.Email = email  if err := us.repo.Save(user); err != nil {  return err  }   // Grpc call to update email in auth service  if err := us.ag.UpdateUsernameEmail(context.Background(), id, "", email); err != nil {  return err  }   return nil }  func (us \*UserService) ChangeUsername(id uint, username string) error {  user, err := us.repo.Find(id)  if err != nil {  return err  }   exists, err := us.repo.ExistsByUsername(username)  switch {  case err != nil:  return err  case exists:  return ErrUserUsernameTaken  }   user.Username = username  if err := us.repo.Save(user); err != nil {  return err  }   // Grpc call to update username in auth service  if err := us.ag.UpdateUsernameEmail(context.Background(), id, username, ""); err != nil {  return err  }   return nil }  func (us \*UserService) ChangeNickname(id uint, nickname string) error {  user, err := us.repo.Find(id)  if err != nil {  return err  }   user.Nickname = nickname  return us.repo.Save(user) }  func (us \*UserService) ChangeAvatar(id uint, file io.ReadSeeker) error {  user, err := us.repo.Find(id)  if err != nil {  return err  }   newAvatarURL, err := us.is.UploadImage(file, fmt.Sprintf("%d\_avatar", user.ID))  if err != nil {  return err  }   user.Avatar = newAvatarURL  return us.repo.Save(user) }  func (us \*UserService) ChangeRole(id, role uint) error {  user, err := us.repo.Find(id)  if err != nil {  return err  }   user.Role = role  return us.repo.Save(user) }  func (us \*UserService) Delete(id uint) error {  user, err := us.repo.Find(id)  if err != nil {  return err  }   if err := us.pg.DeleteUserPosts(context.Background(), id); err != nil {  return err  }   if err := us.pg.DeleteUserComments(context.Background(), id); err != nil {  return err  }   if err := us.pg.DeleteUserLikes(context.Background(), id); err != nil {  return err  }   if err := us.rg.DeleteUserReports(context.Background(), id); err != nil {  return err  }   if err := us.ag.DeleteUser(context.Background(), id); err != nil {  return err  }   return us.repo.Delete(user) } |

Код service.go:

|  |
| --- |
| package service  type Service struct {  User \*UserService  Ban \*BanService }  func New(userRepo UserRepo, banRepo BanRepo, is ImageStorage, ag AuthGateway, pg PostGateway, rg ReportGateway) \*Service {  userService := NewUserService(userRepo, is, ag, pg, rg)  banService := NewBanService(banRepo, pg)   return &Service{  User: userService,  Ban: banService,  } }  } |

**Завдання №4.** Реалізація transport layer

Реалізація http сервера з підключенням API v1.

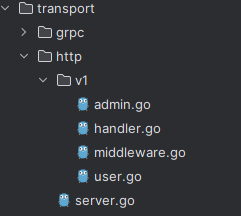
****

Рисунок 8 — Структура http в transport layer

Код кастомного біндера для сервера який підтримує парсинг структур з файлами в form-data binder.go:

|  |
| --- |
| package binder  import (  "fmt"  "github.com/labstack/echo/v4"  "mime/multipart"  "reflect"  "strings" )  type EchoCustomBinder struct {  binder echo.DefaultBinder }  func NewEchoCustomBinder() \*EchoCustomBinder {  return &EchoCustomBinder{} }  func (cb \*EchoCustomBinder) Bind(i interface{}, c echo.Context) error {  if err := cb.binder.Bind(i, c); err != nil {  return err  }   contentType := c.Request().Header.Get(echo.*HeaderContentType*)   if !strings.HasPrefix(contentType, echo.*MIMEApplicationForm*) && !strings.HasPrefix(contentType, echo.*MIMEMultipartForm*) {  return nil  }   form, err := c.MultipartForm()  if err != nil {  return err  }   return bindFiles(i, form.File) }  func bindFiles(i interface{}, files map[string][]\*multipart.FileHeader) error {  iValue := reflect.Indirect(reflect.ValueOf(i))  if iValue.Kind() != reflect.*Struct* {  err := fmt.Errorf("input is not struct pointer, indirect type is %s", iValue.Type().String())  return err  }   iType := iValue.Type()  for i := 0; i < iType.NumField(); i++ {  fValue := iValue.Field(i)  if !fValue.CanSet() {  continue  }   fType := iType.Field(i)   switch fType.Type {  case reflect.TypeOf((\*multipart.FileHeader)(nil)):  file := getFiles(files, fType.Tag.Get("form"), fType.Name)  if len(file) > 0 {  fValue.Set(reflect.ValueOf(file[0]))  }  case reflect.TypeOf(([]\*multipart.FileHeader)(nil)):  file := getFiles(files, fType.Tag.Get("form"), fType.Name)  if len(file) > 0 {  fValue.Set(reflect.ValueOf(file))  }  }  }  return nil }  func getFiles(files map[string][]\*multipart.FileHeader, names ...string) []\*multipart.FileHeader {  for \_, name := range names {  file, ok := files[name]  if ok {  return file  }  }  return nil } |

Код кастомного валідатора для сервера validator.go:

|  |
| --- |
| package validator  import (  "github.com/dlclark/regexp2"  "github.com/go-playground/validator"  "github.com/labstack/echo/v4"  "net/http" )  type EchoCustomValidator struct {  v \*validator.Validate }  var (  usernameRegex = regexp2.MustCompile("^[a-zA-Z0-9]{3,20}$", 0)  passwordRegex = regexp2.MustCompile("^(?=.\*[A-Za-z])(?=.\*\\d)[A-Za-z\\d]{8,}$", 0) )  func NewEchoCustomValidator() \*EchoCustomValidator {  v := validator.New()  v.RegisterValidation("username", func(fl validator.FieldLevel) bool {  b, \_ := usernameRegex.MatchString(fl.Field().String())  return b  })  v.RegisterValidation("password", func(fl validator.FieldLevel) bool {  b, \_ := passwordRegex.MatchString(fl.Field().String())  return b  })   return &EchoCustomValidator{  v: v,  } }  func (cv \*EchoCustomValidator) Validate(i interface{}) error {  if err := cv.v.Struct(i); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }  return nil } |

Код server.go:

|  |
| --- |
| package http  import (  "fmt"  "github.com/SocialNetworkY/Backend/internal/user/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(us v1.UserService, bs v1.BanService, ag v1.AuthGateway) \*Server {  log.Println("Initializing server...")  log.Println("Initializing api...")  handlerV1 := v1.New(us, bs, ag)  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"  "github.com/SocialNetworkY/Backend/internal/user/model"  "github.com/labstack/echo/v4"  "io"  "log"  "time" )  type (  UserService interface {  Find(id uint) (\*model.User, error)  FindByUsername(username string) (\*model.User, error)  FindByEmail(email string) (\*model.User, error)  FindByNickname(nickname string, skip, limit int) ([]\*model.User, error)  ChangeEmail(id uint, email string) error  ChangeUsername(id uint, username string) error  ChangeNickname(id uint, nickname string) error  ChangeAvatar(id uint, file io.ReadSeeker) error  Delete(id uint) error  }   BanService interface {  Ban(userID, adminID uint, reason string, duration time.Duration) error  Unban(banID, adminID uint, reason string) error  Find(id uint) (\*model.Ban, error)  FindByUser(userID uint, skip, limit int) ([]\*model.Ban, error)  }   AuthGateway interface {  Authenticate(ctx context.Context, auth string) (uint, error)  }   Handler struct {  us UserService  bs BanService  ag AuthGateway  } )  func New(us UserService, bs BanService, ag AuthGateway) \*Handler {  return &Handler{  us: us,  ag: ag,  bs: bs,  } }  func (h \*Handler) Init(api \*echo.Group) {  log.Println("Initializing V1 api")  v1 := api.Group("/v1")  {  h.initUserApi(v1)  h.initAdminApi(v1)  } } |

Код v1/middleware.go:

|  |
| --- |
| package v1  import (  "github.com/SocialNetworkY/Backend/internal/user/model"  "github.com/SocialNetworkY/Backend/pkg/constant"  "github.com/labstack/echo/v4"  "net/http"  "strconv" )  const (  *userLocals* = "user"  *requesterLocals* = "requester"  *paramUserID* = "userID"  *paramUsername* = "username" )  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")  }   userID, err := h.ag.Authenticate(c.Request().Context(), authHeader)  if err != nil {  return echo.NewHTTPError(http.*StatusUnauthorized*, err.Error())  }   requester, err := h.us.Find(userID)  if err != nil {  return echo.NewHTTPError(http.*StatusUnauthorized*, err.Error())  }   c.Set(*requesterLocals*, requester)   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*, "User not found")  }   if !requester.Admin {  return echo.NewHTTPError(http.*StatusForbidden*, "User is not admin")  }   return next(c)  } }  func (h \*Handler) setUserByIDFromParam(next echo.HandlerFunc) echo.HandlerFunc {  return func(c echo.Context) error {  userID, err := getUintParam(c, *paramUserID*)  if err != nil {  return err  }   user, err := h.us.Find(userID)  if err != nil {  return echo.NewHTTPError(http.*StatusNotFound*, "user not found")  }   c.Set(*userLocals*, user)   return next(c)  } }  func (h \*Handler) setUserByUsernameFromParam(next echo.HandlerFunc) echo.HandlerFunc {  return func(c echo.Context) error {  username := c.Param(*paramUsername*)  user, err := h.us.FindByUsername(username)  if err != nil {  return echo.NewHTTPError(http.*StatusNotFound*, "user not found")  }   c.Set(*userLocals*, user)   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 } |

Код v1/admin.go:

|  |
| --- |
| package v1  import (  "fmt"  "github.com/SocialNetworkY/Backend/internal/user/model"  "github.com/labstack/echo/v4"  "net/http"  "time" )  func (h \*Handler) initAdminApi(group \*echo.Group) {  initUserEndpoints := func(user \*echo.Group) {  user.POST("/ban", h.banUser)  user.GET("/bans", h.getUserBans)  }   admin := group.Group("/admin", h.authenticationMiddleware, h.adminMiddleware)  {  users := admin.Group("/users")  {  initUserEndpoints(users.Group(fmt.Sprintf("/:%s", *paramUserID*), h.setUserByIDFromParam))  initUserEndpoints(users.Group(fmt.Sprintf("/@:%s", *paramUsername*), h.setUserByUsernameFromParam))  }   admin.POST("/unban", h.unbanByBanID)  } }  func (h \*Handler) banUser(c echo.Context) error {  var req struct {  Reason string `json:"reason"`  Duration string `json:"duration"`  }  if err := c.Bind(&req); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, "invalid request")  }   if req.Reason == "" {  return echo.NewHTTPError(http.*StatusBadRequest*, "invalid reason")  }   duration, err := time.ParseDuration(req.Duration)  if err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, "invalid duration")  }   if duration == 0 {  return echo.NewHTTPError(http.*StatusBadRequest*, "invalid duration")  }   requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "internal error")  }   user, ok := c.Get(*userLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "internal error")  }   if user.Role > requester.Role {  return echo.NewHTTPError(http.*StatusForbidden*, "forbidden")  }   if user.Banned {  return echo.NewHTTPError(http.*StatusBadRequest*, "user already banned")  }   if err := h.bs.Ban(user.ID, requester.ID, req.Reason, duration); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.NoContent(http.*StatusOK*) }  func (h \*Handler) unbanByBanID(c echo.Context) error {  var req struct {  BanID uint `json:"banID"`  Reason string `json:"reason"`  }  if err := c.Bind(&req); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, "invalid request")  }   requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "internal error")  }   ban, err := h.bs.Find(req.BanID)  if err != nil {  return echo.NewHTTPError(http.*StatusNotFound*, "ban not found")  }   if ban.UserID == requester.ID {  return echo.NewHTTPError(http.*StatusForbidden*, "forbidden")  }   if !ban.Active {  return echo.NewHTTPError(http.*StatusBadRequest*, "ban already inactive")  }   bannedBy, err := h.us.Find(ban.BannedBy)  if err != nil {  return echo.NewHTTPError(http.*StatusNotFound*, "admin who banned not found")  }   if bannedBy.Role > requester.Role {  return echo.NewHTTPError(http.*StatusForbidden*, "forbidden")  }   if err := h.bs.Unban(ban.ID, requester.ID, req.Reason); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.NoContent(http.*StatusOK*) }  func (h \*Handler) getUserBans(c echo.Context) error {  user, ok := c.Get(*userLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "internal error")  }   return c.JSON(http.*StatusOK*, struct {  Bans []\*model.Ban `json:"bans"`  }{  Bans: user.Bans,  }) } |

Код v1/user.go:

|  |
| --- |
| package v1  import (  "fmt"  "github.com/SocialNetworkY/Backend/internal/user/model"  "github.com/SocialNetworkY/Backend/pkg/constant"  "github.com/labstack/echo/v4"  "mime/multipart"  "net/http" )  func (h \*Handler) initUserApi(group \*echo.Group) {  initUserEndpoints := func(user \*echo.Group) {  user.GET("", h.getUser)  user.GET("/full", h.getFullUser, h.authenticationMiddleware)  user.PATCH("", h.patchUser, h.authenticationMiddleware)  user.DELETE("", h.deleteUser, h.authenticationMiddleware)  }   users := group.Group("/users")  {  initUserEndpoints(users.Group(fmt.Sprintf("/@:%s", *paramUsername*), h.setUserByUsernameFromParam))  initUserEndpoints(users.Group(fmt.Sprintf("/:%s", *paramUserID*), h.setUserByIDFromParam))  } }  func (h \*Handler) getUser(c echo.Context) error {  user, ok := c.Get(*userLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "internal error")  }   return c.JSON(http.*StatusOK*, struct {  UserID uint `json:"user\_id"`  Username string `json:"username"`  Nickname string `json:"nickname"`  Avatar string `json:"avatar"`  }{  UserID: user.ID,  Username: user.Username,  Nickname: user.Nickname,  Avatar: user.Avatar,  }) }  func (h \*Handler) getFullUser(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "user not found")  }   user, ok := c.Get(*userLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "internal error")  }   if requester.ID != user.ID && !requester.Admin {  return echo.NewHTTPError(http.*StatusForbidden*, "you don't have permission to view this user")  }   return c.JSON(http.*StatusOK*, user) }  func (h \*Handler) patchUser(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "user not found")  }   authHeader := c.Request().Header.Get(constant.*HTTPAuthorizationHeader*)  if authHeader == "" {  return echo.NewHTTPError(http.*StatusUnauthorized*, "missing Authorization header")  }   user, ok := c.Get(*userLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "internal error")  }   var requestBody struct {  Nickname string `form:"nickname"`  Username string `form:"username"`  Email string `form:"email"`  Avatar \*multipart.FileHeader `form:"avatar"`  }  if err := c.Bind(&requestBody); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, "invalid request body")  }   if requester.ID != user.ID && requester.Role < constant.*RoleAdminLvl1* {  return echo.NewHTTPError(http.*StatusForbidden*, "you don't have permission to update this user")  }   if requestBody.Nickname != "" {  if err := h.us.ChangeNickname(user.ID, requestBody.Nickname); err != nil {  return err  }  }  if requestBody.Username != "" {  if err := h.us.ChangeUsername(user.ID, requestBody.Username); err != nil {  return err  }  }  if requestBody.Email != "" {  if err := h.us.ChangeEmail(user.ID, requestBody.Email); err != nil {  return err  }  }   if requestBody.Avatar != nil {  src, err := requestBody.Avatar.Open()  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }  defer src.Close()   if err := h.us.ChangeAvatar(user.ID, src); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }  }   return c.NoContent(http.*StatusOK*) }  func (h \*Handler) deleteUser(c echo.Context) error {  requester, ok := c.Get(*requesterLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusUnauthorized*, "user not found")  }   authHeader := c.Request().Header.Get(constant.*HTTPAuthorizationHeader*)  if authHeader == "" {  return echo.NewHTTPError(http.*StatusUnauthorized*, "missing Authorization header")  }   user, ok := c.Get(*userLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "internal error")  }   if requester.ID != user.ID && requester.Role < constant.*RoleAdminLvl3* {  return echo.NewHTTPError(http.*StatusForbidden*, "you don't have permission to delete this user")  }   if err := h.us.Delete(user.ID); err != nil {  return err  }   return c.NoContent(http.*StatusOK*) } |

Створення Proto файлу для grpc.

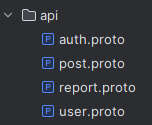
****

Рисунок 9 — Структура api папки з .proto файлами

Код user.proto:

|  |
| --- |
| syntax = "proto3"; option go\_package = "/gen";  service UserService {  rpc CreateUser(CreateUserRequest) returns (CreateUserResponse);  rpc UserInfo(UserInfoRequest) returns (UserInfoResponse); }  message CreateUserRequest {  uint64 user\_id = 1;  string username = 2;  string email = 3;  uint64 role = 4; }  message CreateUserResponse {  uint64 user\_id = 1; }  message UserInfoRequest {  uint64 user\_id = 1; }  message UserInfoResponse {  uint64 user\_id = 1;  uint64 role = 4;  bool banned = 5;  string banReason = 6;  string banExpiredAt = 7; } |

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.

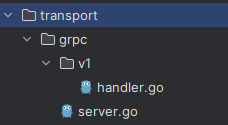
****

Рисунок 10 — Структура grpc в transport layer

Код server.go:

|  |
| --- |
| package grpc  import (  "context"  "fmt"  "github.com/SocialNetworkY/Backend/internal/user/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(us v1.UserService, ag v1.AuthGateway) \*Server {  handler := v1.New(us, ag)  gen.RegisterUserServiceServer(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"  "errors"  "github.com/SocialNetworkY/Backend/internal/user/model"  "github.com/SocialNetworkY/Backend/pkg/gen"  "time" )  type (  UserService interface {  Create(id, role uint, username, email string) (\*model.User, error)  Find(id uint) (\*model.User, error)  }   AuthGateway interface {  Authenticate(ctx context.Context, auth string) (uint, error)  }   Handler struct {  gen.UnimplementedUserServiceServer  us UserService  ag AuthGateway  } )  func New(us UserService, ag AuthGateway) \*Handler {  return &Handler{  us: us,  ag: ag,  } }  func (h \*Handler) CreateUser(ctx context.Context, r \*gen.CreateUserRequest) (\*gen.CreateUserResponse, error) {  user, err := h.us.Create(uint(r.GetUserId()), uint(r.GetRole()), r.GetUsername(), r.GetEmail())  if err != nil {  return nil, err  }   return &gen.CreateUserResponse{  UserId: uint64(user.ID),  }, nil }  func (h \*Handler) UserInfo(ctx context.Context, r \*gen.UserInfoRequest) (\*gen.UserInfoResponse, error) {  userID := uint(r.GetUserId())  if userID == 0 {  return nil, errors.New("invalid user id")  }   user, err := h.us.Find(userID)  if err != nil {  return nil, err  }   banReason := ""  banExpiredAt := ""  if user.Banned {  banReason = user.ActiveBan.BanReason  banExpiredAt = user.ActiveBan.ExpiredAt.Format(time.*RFC3339*)  }   return &gen.UserInfoResponse{  UserId: uint64(user.ID),  Role: uint64(user.Role),  Banned: user.Banned,  BanReason: banReason,  BanExpiredAt: banExpiredAt,  }, 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.

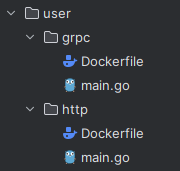
****

Рисунок 12 — Структура cmd/user 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/user/grpc/main.go  FROM alpine:latest COPY --from=*builder* server . CMD ["/server"] |

Код grpc/main.go:

|  |
| --- |
| package main  import (  "fmt"  "github.com/SocialNetworkY/Backend/internal/user/gateway/post"  "github.com/SocialNetworkY/Backend/internal/user/gateway/report"  "github.com/SocialNetworkY/Backend/internal/user/repository"  "github.com/SocialNetworkY/Backend/pkg/storage"  "gorm.io/driver/mysql"  "log"   "github.com/SocialNetworkY/Backend/internal/user/gateway/auth"  "github.com/SocialNetworkY/Backend/internal/user/service"  "github.com/SocialNetworkY/Backend/internal/user/transport/grpc"   "github.com/caarlos0/env/v6" )  type Config struct {  DB string `env:"DB"`  Port int `env:"PORT"`  AuthServiceHttpAddr string `env:"AUTH\_SERVICE\_HTTP\_ADDR"`  AuthServiceGrpcAddr string `env:"AUTH\_SERVICE\_GRPC\_ADDR"`  PostServiceHttpAddr string `env:"POST\_SERVICE\_HTTP\_ADDR"`  PostServiceGrpcAddr string `env:"POST\_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() {  repos, err := repository.New(mysql.Open(cfg.DB))  if err != nil {  log.Fatal(err)  }   imageStorage, err := storage.NewLocalStorage(cfg.StorageFolder, fmt.Sprintf("http://localhost:%d/%s", cfg.Port, "storage"))  if err != nil {  log.Fatal(err)  }   authGateway := auth.New(cfg.AuthServiceHttpAddr, cfg.AuthServiceGrpcAddr)  postGateway := post.New(cfg.PostServiceHttpAddr, cfg.PostServiceGrpcAddr)  reportGateway := report.New(cfg.ReportServiceHttpAddr, cfg.ReportServiceGrpcAddr)  services := service.New(repos.User, repos.Ban, imageStorage, authGateway, postGateway, reportGateway)   if err := grpc.New(cfg.Port).Init(services.User, authGateway).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/user/http/main.go  FROM alpine:latest COPY --from=*builder* server . CMD ["/server"] |

Код http/main.go:

|  |
| --- |
| package main  import (  "fmt"  "github.com/SocialNetworkY/Backend/internal/user/gateway/post"  "github.com/SocialNetworkY/Backend/internal/user/gateway/report"  "github.com/SocialNetworkY/Backend/internal/user/repository"  "github.com/SocialNetworkY/Backend/pkg/storage"  "gorm.io/driver/mysql"  "log"   "github.com/SocialNetworkY/Backend/internal/user/gateway/auth"  "github.com/SocialNetworkY/Backend/internal/user/service"  "github.com/SocialNetworkY/Backend/internal/user/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:","`  AuthServiceHttpAddr string `env:"AUTH\_SERVICE\_HTTP\_ADDR"`  AuthServiceGrpcAddr string `env:"AUTH\_SERVICE\_GRPC\_ADDR"`  PostServiceHttpAddr string `env:"POST\_SERVICE\_HTTP\_ADDR"`  PostServiceGrpcAddr string `env:"POST\_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() {  repos, err := repository.New(mysql.Open(cfg.DB))  if err != nil {  log.Fatal(err)  }   imageStorage, 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 := post.New(cfg.PostServiceHttpAddr, cfg.PostServiceGrpcAddr)  reportGateway := report.New(cfg.ReportServiceHttpAddr, cfg.ReportServiceGrpcAddr)  services := service.New(repos.User, repos.Ban, imageStorage, authGateway, postGateway, reportGateway)   if err := http.New(cfg.BodyLimit, cfg.AllowedOrigins, cfg.Port).Init(services.User, services.Ban, authGateway).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   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  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  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  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  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: "\*"  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  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:  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;  }   } } |