ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення

ЛАБОРАТОРНА РОБОТА №1

з дисципліни «Веб-технології»

.(назва дисципліни)

на тему: «Створення Backend. Auth сервіс»

ЛРІПЗс.2401075.02.07.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 аутентифікації (Auth сервіс) для забезпечення безпечного доступу користувачів до системи. Реалізувати функціонал генерації та верифікації JWT токенів, 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 Auth сервісу.

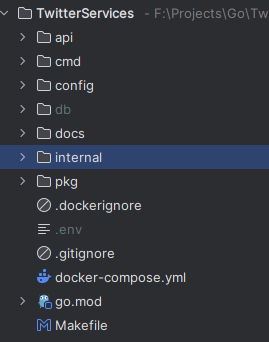


Рисунок 1 — Структура проекту

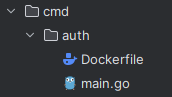


Рисунок 2 — Структура папки cmd

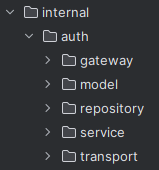


Рисунок 3 — Структура папки internal

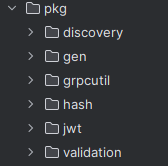


Рисунок 4 — Структура папки pkg

Додавання необхідних залежностей для роботи з базою даних, аутентифікації та транспортування даних через REST і gRPC (наприклад, github.com/golang-jwt/jwt, gorm.io/gorm, та інші).

**Завдання №2.** Реалізація repository layer

Створення моделей структур ActivationToken, RefreshToken, User

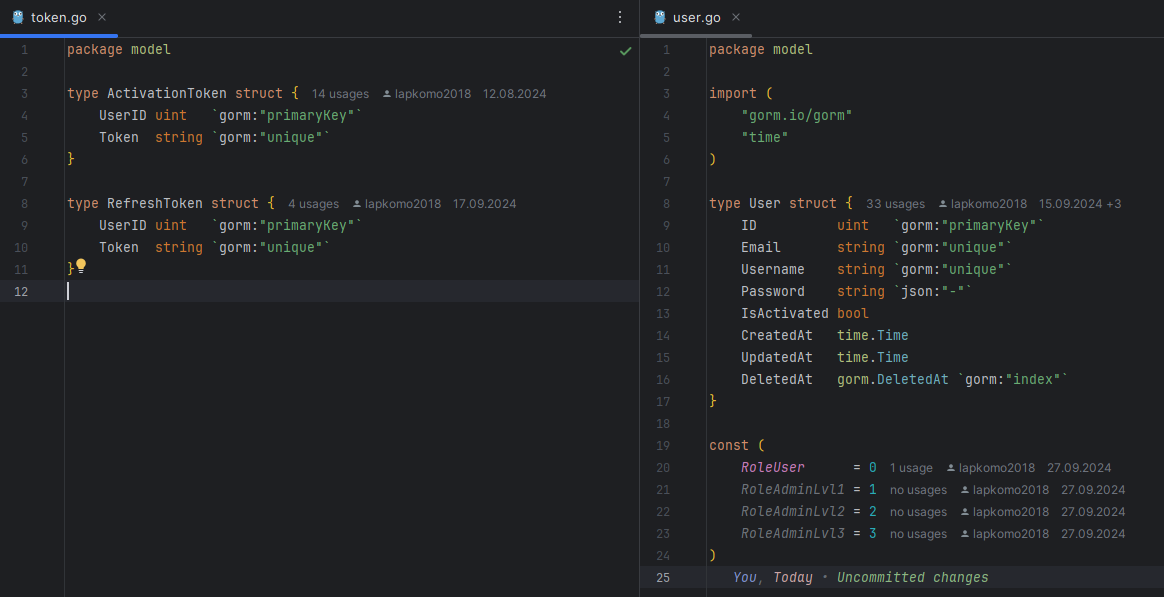


Рисунок 5 — Моделі структур ActivationToken, RefreshToken, User

Код env.go для парсингу env:

|  |
| --- |
| package model  import (  "github.com/joho/godotenv"  "log"  "os"  "strconv" )  type Env struct {  DB string  DiscoveryAddr string  ExternalRestPort int  ExternalGrpcPort int  HashSalt string  JWTSecret string  JWTRefreshSecret string }  const (  *envDB* = "DB"  *envDiscoveryAddr* = "DISCOVERY\_ADDR"  *envExternalRestPort* = "EXTERNAL\_REST\_PORT"  *envExternalGrpcPort* = "EXTERNAL\_GRPC\_PORT"  *envHashSalt* = "HASH\_SALT"  *envJWTSecret* = "JWT\_SECRET"  *envJWTRefreshSecret* = "JWT\_REFRESH" )  func ParseEnv() (\*Env, error) {  var err error  err = godotenv.Load()  if err != nil {  log.Printf("Error loading .env file, proceeding without it: %v", err)  }   restPort, err := strconv.Atoi(os.Getenv(*envExternalRestPort*))  if err != nil {  return nil, err  }   grpcPort, err := strconv.Atoi(os.Getenv(*envExternalGrpcPort*))  if err != nil {  return nil, err  }   return &Env{  DB: os.Getenv(*envDB*),  DiscoveryAddr: os.Getenv(*envDiscoveryAddr*),  ExternalRestPort: restPort,  ExternalGrpcPort: grpcPort,  HashSalt: os.Getenv(*envHashSalt*),  JWTSecret: os.Getenv(*envJWTSecret*),  JWTRefreshSecret: os.Getenv(*envJWTRefreshSecret*),  }, nil } |

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

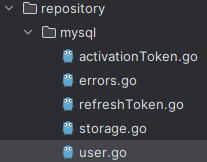


Рисунок 6 — Структура папки repository

Код user.go:

|  |
| --- |
| package mysql  import (  "github.com/lapkomo2018/goTwitterServices/internal/auth/model"  "gorm.io/gorm" )  type UserStorage struct {  db \*gorm.DB }  func NewUserStorage(db \*gorm.DB) \*UserStorage {  return &UserStorage{  db: db,  } }  func (us \*UserStorage) ExistsByLogin(login string) (bool, error) {  exists, err := us.ExistsByEmail(login)  if err != nil {  return false, err  }  if exists {  return true, nil  }   exists, err = us.ExistsByUsername(login)  if err != nil {  return false, err  }  return exists, nil }  func (us \*UserStorage) ExistsByEmail(email string) (bool, error) {  var count int64  err := us.db.Model(&model.User{}).Where("email = ?", email).Count(&count).Error  if err != nil {  return false, ErrDatabase  }  return count > 0, nil }  func (us \*UserStorage) ExistsByUsername(username string) (bool, error) {  var count int64  err := us.db.Model(&model.User{}).Where("username = ?", username).Count(&count).Error  if err != nil {  return false, ErrDatabase  }  return count > 0, nil }  func (us \*UserStorage) FindByLogin(login string) (\*model.User, error) {  user, err := us.FindByEmail(login)  if err == nil {  return user, nil  }   user, err = us.FindByUsername(login)  if err == nil {  return user, nil  }   return nil, err }  func (us \*UserStorage) Find(id uint) (\*model.User, error) {  user := &model.User{}  if err := us.db.Where("id = ?", id).First(user).Error; err != nil {  return nil, ErrUserNotFound  }  return user, nil }  func (us \*UserStorage) FindByUsername(username string) (\*model.User, error) {  user := &model.User{}  if err := us.db.Where("username = ?", username).First(user).Error; err != nil {  return nil, ErrUserNotFound  }  return user, nil }  func (us \*UserStorage) FindByEmail(email string) (\*model.User, error) {  user := &model.User{}  if err := us.db.Where("email = ?", email).First(user).Error; err != nil {  return nil, ErrUserNotFound  }  return user, nil }  func (us \*UserStorage) Add(user \*model.User) error {  if err := us.db.Create(user).Error; err != nil {  return ErrUserCreate  }  return nil }  func (us \*UserStorage) Save(user \*model.User) error {  if err := us.db.Save(user).Error; err != nil {  return ErrUserSave  }  return nil }  func (us \*UserStorage) Delete(user \*model.User) error {  if err := us.db.Delete(user).Error; err != nil {  return ErrUserDelete  }  return nil } |

Код activationToken.go:

|  |
| --- |
| package mysql  import (  "errors"  "github.com/lapkomo2018/goTwitterServices/internal/auth/model"  "gorm.io/gorm" )  type ActivationTokenStorage struct {  db \*gorm.DB }  func NewActivationTokenStorage(db \*gorm.DB) \*ActivationTokenStorage {  return &ActivationTokenStorage{  db: db,  } }  func (ats \*ActivationTokenStorage) Set(userID uint, activationToken string) error {  var existingToken model.ActivationToken  err := ats.db.Where("user\_id = ?", userID).First(&existingToken).Error  if err != nil && !errors.Is(err, gorm.ErrRecordNotFound) {  return ErrActivationTokenSet  }   if errors.Is(err, gorm.ErrRecordNotFound) {  newToken := model.ActivationToken{  UserID: userID,  Token: activationToken,  }   if err := ats.db.Create(&newToken).Error; err != nil {  return ErrActivationTokenCreate  }  return nil  }   existingToken.Token = activationToken  if err := ats.db.Save(&existingToken).Error; err != nil {  return ErrActivationTokenSave  }  return nil }  func (ats \*ActivationTokenStorage) Get(userID uint) (\*model.ActivationToken, error) {  existingToken := &model.ActivationToken{}  err := ats.db.Where("user\_id = ?", userID).First(existingToken).Error  if err != nil {  return nil, err  }  return existingToken, nil }  func (ats \*ActivationTokenStorage) GetByToken(activationToken string) (\*model.ActivationToken, error) {  existingToken := &model.ActivationToken{}  err := ats.db.Where("token = ?", activationToken).First(existingToken).Error  if err != nil {  return nil, ErrActivationTokenNotFound  }  return existingToken, nil }  func (ats \*ActivationTokenStorage) Delete(userID uint) error {  err := ats.db.Where("user\_id = ?", userID).Delete(&model.ActivationToken{}).Error  if err != nil {  return ErrActivationTokenNotFound  }  return nil } |

Код refreshToken.go:

|  |
| --- |
| package mysql  import (  "errors"  "github.com/lapkomo2018/goTwitterServices/internal/auth/model"  "gorm.io/gorm" )  type (  RefreshTokenStorage struct {  db \*gorm.DB  } )  func NewRefreshTokenStorage(db \*gorm.DB) \*RefreshTokenStorage {  return &RefreshTokenStorage{  db: db,  } }  func (us \*RefreshTokenStorage) Set(userID uint, refreshToken string) error {  var existingToken model.RefreshToken  err := us.db.Where("user\_id = ?", userID).First(&existingToken).Error  if err != nil && !errors.Is(err, gorm.ErrRecordNotFound) {  return ErrRefreshTokenSet  }   if errors.Is(err, gorm.ErrRecordNotFound) {  newToken := model.RefreshToken{  UserID: userID,  Token: refreshToken,  }   if err := us.db.Create(&newToken).Error; err != nil {  return ErrRefreshTokenCreate  }  return nil  }   existingToken.Token = refreshToken  if err := us.db.Save(&existingToken).Error; err != nil {  return ErrRefreshTokenSave  }  return nil }  func (us \*RefreshTokenStorage) Get(userID uint) (string, error) {  existingToken := &model.RefreshToken{}  err := us.db.Where("user\_id = ?", userID).First(existingToken).Error  if err != nil {  return "", ErrRefreshTokenNotFound  }  return existingToken.Token, nil } |

Код errors.go:

|  |
| --- |
| package mysql  import (  "errors" )  var (  ErrDatabase = errors.New("database error")  ErrBDConnect = errors.New("error connecting to the database")  ErrBDPing = errors.New("error pinging the database")   ErrActivationTokenSet = errors.New("error setting activation token")  ErrActivationTokenCreate = errors.New("error creating activation token")  ErrActivationTokenSave = errors.New("error saving activation token")  ErrActivationTokenNotFound = errors.New("activation token not found")   ErrRefreshTokenSet = errors.New("error setting refresh token")  ErrRefreshTokenCreate = errors.New("error creating refresh token")  ErrRefreshTokenSave = errors.New("error saving refresh token")  ErrRefreshTokenNotFound = errors.New("refresh token not found")   ErrUserNotFound = errors.New("user not found")  ErrUserExists = errors.New("user already exists")  ErrUserCreate = errors.New("error creating user")  ErrUserSave = errors.New("error saving user")  ErrUserDelete = errors.New("error deleting user") ) |

Код storage.go:

|  |
| --- |
| package mysql  import (  "github.com/lapkomo2018/goTwitterServices/internal/auth/model"  "gorm.io/driver/mysql"  "gorm.io/gorm"  "gorm.io/gorm/logger"  "log" )  type Storage struct {  User \*UserStorage  RefreshToken \*RefreshTokenStorage  ActivationToken \*ActivationTokenStorage }  func New(dsn string) (\*Storage, error) {  log.Println("Connecting mysql...")  db, err := gorm.Open(mysql.Open(dsn), &gorm.Config{  Logger: logger.Default.LogMode(logger.*Silent*),  })  if err != nil {  return nil, err  }  log.Println("Connected mysql")   log.Println("Starting AutoMigrating...")  if err := db.AutoMigrate(&model.User{}, &model.ActivationToken{}, &model.RefreshToken{}); err != nil {  return nil, err  }  log.Println("AutoMigrating completed")   return &Storage{  User: NewUserStorage(db),  RefreshToken: NewRefreshTokenStorage(db),  ActivationToken: NewActivationTokenStorage(db),  }, nil } |

**Завдання №3.** Реалізація service layer

Реалізація функції генерації JWT токену після успішної аутентифікації користувача, налаштування секретного ключа та часу дії токену.

Код pkg/jwt/manager.go:

|  |
| --- |
| package jwt  import (  "github.com/golang-jwt/jwt/v4"  "time" )  type (  Config struct {  TokenDuration time.Duration  RefreshDuration time.Duration  }   Manager struct {  secretKey string  tokenDuration time.Duration  refreshSecretKey string  refreshDuration time.Duration  } )  func NewManager(cfg Config, secret, refreshSecret string) \*Manager {  return &Manager{  secretKey: secret,  tokenDuration: cfg.TokenDuration,  refreshSecretKey: refreshSecret,  refreshDuration: cfg.RefreshDuration,  } }  type UserClaims struct {  jwt.RegisteredClaims  UserID uint `json:"user\_id"` }  func (manager \*Manager) Generate(userID uint) (accessToken string, refreshToken string, err error) {  // Generate access token  accessToken, err = manager.generateToken(userID, manager.secretKey, manager.tokenDuration)  if err != nil {  return "", "", ErrGenerateAccessToken  }   // Generate refresh token  refreshToken, err = manager.generateToken(userID, manager.refreshSecretKey, manager.refreshDuration)  if err != nil {  return "", "", ErrGenerateRefreshToken  }   return accessToken, refreshToken, nil }  func (manager \*Manager) generateToken(userID uint, secretKey string, duration time.Duration) (string, error) {  claims := &UserClaims{  RegisteredClaims: jwt.RegisteredClaims{  ExpiresAt: jwt.NewNumericDate(time.Now().Add(duration)),  },  UserID: userID,  }   token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)  return token.SignedString([]byte(secretKey)) }  func (manager \*Manager) Verify(accessToken string) (userID uint, err error) {  return manager.verifyToken(accessToken, manager.secretKey) }  func (manager \*Manager) VerifyRefreshToken(refreshToken string) (userID uint, err error) {  return manager.verifyToken(refreshToken, manager.refreshSecretKey) }  func (manager \*Manager) verifyToken(tokenString, secretKey string) (userID uint, err error) {  token, err := jwt.ParseWithClaims(  tokenString,  &UserClaims{},  func(token \*jwt.Token) (interface{}, error) {  if \_, ok := token.Method.(\*jwt.SigningMethodHMAC); !ok {  return nil, ErrUnexpectedSigningMethod  }  return []byte(secretKey), nil  },  )  if err != nil {  return 0, err  }   claims, ok := token.Claims.(\*UserClaims)  if !ok || !token.Valid {  return 0, ErrInvalidToken  }   return claims.UserID, nil } |

Реалізація логіки хешування паролів перед зберіганням у базі даних.

Код pkg/hash/hash.go:

|  |
| --- |
| package hash  import (  "crypto/sha1"  "encoding/hex" )  type (  SHA1Hasher struct {  salt string  } )  func NewSHA1Hasher(salt string) \*SHA1Hasher {  return &SHA1Hasher{  salt: salt,  } }  func (h \*SHA1Hasher) Hash(password string) string {  hash := sha1.New()  hash.Write([]byte(password))   return hex.EncodeToString(hash.Sum([]byte(h.salt))) }  func (h \*SHA1Hasher) Verify(hash string, password string) bool {  return h.Hash(password) == hash } |

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

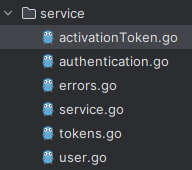


Рисунок 7 — Структура папки service

Код activationToken.go:

|  |
| --- |
| package service  import (  "crypto/rand"  "encoding/hex"  "github.com/lapkomo2018/goTwitterServices/internal/auth/model" )  type (  ActivationTokenStorage interface {  Set(userID uint, activationToken string) error  Get(userID uint) (\*model.ActivationToken, error)  GetByToken(activationToken string) (\*model.ActivationToken, error)  Delete(userID uint) error  } )  type ActivationTokenService struct {  storage ActivationTokenStorage }  func NewActivationTokenService(storage ActivationTokenStorage) \*ActivationTokenService {  return &ActivationTokenService{  storage: storage,  } }  func (ts \*ActivationTokenService) Generate(userID uint) (string, error) {  // generate activation token  tokenBytes := make([]byte, 32)  if \_, err := rand.Read(tokenBytes); err != nil {  return "", ErrActivationTokenGeneration  }  activationToken := hex.EncodeToString(tokenBytes)   if err := ts.storage.Set(userID, activationToken); err != nil {  return "", err  }   return activationToken, nil }  func (ts \*ActivationTokenService) Get(userID uint) (\*model.ActivationToken, error) {  return ts.storage.Get(userID) }  func (ts \*ActivationTokenService) GetByToken(activationToken string) (\*model.ActivationToken, error) {  return ts.storage.GetByToken(activationToken) }  func (ts \*ActivationTokenService) Delete(userID uint) error {  return ts.storage.Delete(userID) } |

Код authentication.go:

|  |
| --- |
| package service  import (  "github.com/lapkomo2018/goTwitterServices/internal/auth/model"  "strings" )  type (  authSchemeHandler func(token string) (\*model.User, error)   AuthenticationUserService interface {  Find(id uint) (\*model.User, error)  }  AuthenticationTokenService interface {  Verify(accessToken string) (userID uint, err error)  }   AuthenticationService struct {  userService AuthenticationUserService  tokenService AuthenticationTokenService  authHandlers map[string]authSchemeHandler  } )  func NewAuthenticationService(us AuthenticationUserService, ts AuthenticationTokenService) \*AuthenticationService {  authService := &AuthenticationService{  userService: us,  tokenService: ts,  }   authService.authHandlers = map[string]authSchemeHandler{  "bearer": authService.bearerHandler,  }   return authService }  func (as \*AuthenticationService) Auth(auth string) (\*model.User, error) {  authHeaderParts := strings.Split(auth, " ")  if len(authHeaderParts) != 2 {  return nil, ErrAuthenticationInvalidAuthString  }   scheme := strings.ToLower(authHeaderParts[0])  tokenString := authHeaderParts[1]   handler := as.authHandlers[scheme]   return handler(tokenString) }  func (as \*AuthenticationService) bearerHandler(token string) (\*model.User, error) {  userID, err := as.tokenService.Verify(token)  if err != nil {  return nil, err  }   user, err := as.userService.Find(userID)  if err != nil {  return nil, err  }   if !user.IsActivated {  return nil, ErrUserNotActivated  }   return user, nil } |

Код errors.go:

|  |
| --- |
| package service  import "errors"  var (  ErrActivationTokenGeneration = errors.New("error generating activation token")   ErrAuthenticationInvalidAuthString = errors.New("invalid auth string")   ErrInvalidRefreshToken = errors.New("invalid refresh token")   ErrUserNotActivated = errors.New("user not activated")  ErrUserUsernameTaken = errors.New("username already taken")  ErrUserEmailTaken = errors.New("email already taken")  ErrUserInvalidPassword = errors.New("invalid password") ) |

Код service.go:

|  |
| --- |
| package service  type Service struct {  User \*UserService  Tokens \*TokensService  Authentication \*AuthenticationService  ActivationToken \*ActivationTokenService }  func New(userStorage UserStorage, refreshTokenStorage TokensRefreshTokenStorage, activationTokenStorage ActivationTokenStorage, tokenManager TokensManager, hasher Hasher, ug UserGateway) \*Service {  activationTokenService := NewActivationTokenService(activationTokenStorage)  tokensService := NewTokensService(refreshTokenStorage, tokenManager)  userService := NewUserService(userStorage, tokensService, activationTokenService, hasher, ug)  authenticationService := NewAuthenticationService(userService, tokensService)   return &Service{  User: userService,  Tokens: tokensService,  Authentication: authenticationService,  ActivationToken: activationTokenService,  } } |

Код tokens.go:

|  |
| --- |
| package service  type (  TokensRefreshTokenStorage interface {  Set(userID uint, refreshToken string) error  Get(userID uint) (string, error)  }   TokensManager interface {  Generate(userID uint) (accessToken string, refreshToken string, err error)  Verify(accessToken string) (userID uint, err error)  VerifyRefreshToken(refreshToken string) (userID uint, err error)  } )  type TokensService struct {  storage TokensRefreshTokenStorage  tokenManager TokensManager }  func NewTokensService(refreshTokenStorage TokensRefreshTokenStorage, tokenManager TokensManager) \*TokensService {  return &TokensService{  storage: refreshTokenStorage,  tokenManager: tokenManager,  } }  func (ts \*TokensService) Generate(userID uint) (string, string, error) {  accessToken, refreshToken, err := ts.tokenManager.Generate(userID)  if err != nil {  return "", "", err  }   if err := ts.storage.Set(userID, refreshToken); err != nil {  return "", "", err  }   return accessToken, refreshToken, nil }  func (ts \*TokensService) Verify(accessToken string) (userID uint, err error) {  return ts.tokenManager.Verify(accessToken) }  func (ts \*TokensService) VerifyRefreshToken(refreshToken string) (userID uint, err error) {  userID, err = ts.tokenManager.VerifyRefreshToken(refreshToken)  if err != nil {  return 0, err  }  existingToken, err := ts.storage.Get(userID)  if err != nil {  return 0, err  }   if existingToken != refreshToken {  return 0, ErrInvalidRefreshToken  }   return userID, nil } |

Код user.go:

|  |
| --- |
| package service  import (  "context"  "fmt"  "github.com/lapkomo2018/goTwitterServices/internal/auth/model" )  type (  Hasher interface {  Hash(password string) string  Verify(hash string, password string) bool  }   UserStorage interface {  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)  FindByUsername(username string) (\*model.User, error)  FindByEmail(email string) (\*model.User, error)  Add(user \*model.User) error  Save(user \*model.User) error  Delete(user \*model.User) error  }   UserTokenService interface {  Generate(userID uint) (string, string, error)  }   UserActivationTokenService interface {  Generate(userID uint) (string, error)  Get(userID uint) (\*model.ActivationToken, error)  GetByToken(activationToken string) (\*model.ActivationToken, error)  Delete(userID uint) error  }   UserGateway interface {  CreateUser(ctx context.Context, auth string, userID, role uint, username, email string) error  GetUserRole(ctx context.Context, auth string, userID uint) (uint, error)  }   UserService struct {  storage UserStorage  tokenService UserTokenService  activationTokenService UserActivationTokenService  hasher Hasher  userGateway UserGateway  } )  func NewUserService(userStorage UserStorage, tokenService UserTokenService, activationTokenService UserActivationTokenService, hasher Hasher, ug UserGateway) \*UserService {  return &UserService{  storage: userStorage,  tokenService: tokenService,  activationTokenService: activationTokenService,  hasher: hasher,  userGateway: ug,  } }  func (us \*UserService) Register(username, email, password string) (activationToken string, err error) {  exists, err := us.storage.ExistsByUsername(username)  switch {  case err != nil:  return "", err  case exists:  return "", ErrUserUsernameTaken  }   exists, err = us.storage.ExistsByEmail(email)  switch {  case err != nil:  return "", err  case exists:  return "", ErrUserEmailTaken  }   hashedPassword := us.hasher.Hash(password)   user := &model.User{  Email: email,  Username: username,  Password: hashedPassword,  }   if err := us.storage.Add(user); err != nil {  return "", err  }   activationToken, err = us.activationTokenService.Generate(user.ID)  if err != nil {  return "", err  }   return activationToken, nil }  func (us \*UserService) Login(login, password string) (string, string, error) {  user, err := us.storage.FindByLogin(login)  if err != nil {  return "", "", err  }   if !us.hasher.Verify(user.Password, password) {  return "", "", ErrUserInvalidPassword  }   if !user.IsActivated {  return "", "", ErrUserNotActivated  }   return us.tokenService.Generate(user.ID) }  func (us \*UserService) Activate(activationToken string) (accessToken, refreshToken string, err error) {  token, err := us.activationTokenService.GetByToken(activationToken)  if err != nil {  return "", "", err  }   user, err := us.storage.Find(token.UserID)  if err != nil {  return "", "", err  }   user.IsActivated = true  if err := us.storage.Save(user); err != nil {  return "", "", err  }   accessToken, refreshToken, err = us.tokenService.Generate(user.ID)  if err != nil {  return "", "", err  }   if err := us.userGateway.CreateUser(context.Background(), fmt.Sprintf("Bearer %s", accessToken), user.ID, model.*RoleUser*, user.Username, user.Email); err != nil {  return "", "", err  }   if err := us.activationTokenService.Delete(token.UserID); err != nil {  return "", "", err  }   return accessToken, refreshToken, nil }  func (us \*UserService) Find(id uint) (\*model.User, error) {  return us.storage.Find(id) }  func (us \*UserService) ChangeEmail(id uint, email string) error {  user, err := us.storage.Find(id)  if err != nil {  return err  }   exists, err := us.storage.ExistsByEmail(email)  switch {  case err != nil:  return err  case exists:  return ErrUserEmailTaken  }   user.Email = email  return us.storage.Save(user) }  func (us \*UserService) ChangeUsername(id uint, username string) error {  user, err := us.storage.Find(id)  if err != nil {  return err  }   exists, err := us.storage.ExistsByUsername(username)  switch {  case err != nil:  return err  case exists:  return ErrUserUsernameTaken  }   user.Username = username  return us.storage.Save(user) }  func (us \*UserService) ChangePassword(id uint, password string) error {  user, err := us.storage.Find(id)  if err != nil {  return err  }   hashedPassword := us.hasher.Hash(password)  user.Password = hashedPassword  return us.storage.Save(user) }  func (us \*UserService) Delete(id uint) error {  user, err := us.storage.Find(id)  if err != nil {  return err  }   return us.storage.Delete(user) } |

**Завдання №4.** Реалізація transport layer

Реалізація Rest сервера з підключенням API v1.

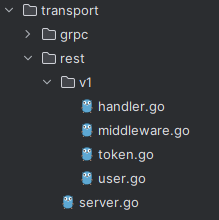
****

Рисунок 8 — Структура rest в transport layer

Код server.go:

|  |
| --- |
| package rest  import (  "fmt"  "github.com/labstack/echo/v4"  "github.com/labstack/echo/v4/middleware"  "github.com/lapkomo2018/goTwitterServices/internal/auth/transport/rest/v1"  echoSwagger "github.com/swaggo/echo-swagger"  "log"  "net/http"  "strconv"  "time"   \_ "github.com/lapkomo2018/goTwitterServices/docs/auth" )  type (  Config struct {  Port int  BodyLimit int  AllowedOrigins []string  }   Server struct {  echo \*echo.Echo  addr string  } )  func New(cfg Config) \*Server {  log.Printf("Creating rest server with port: %d", cfg.Port)   e := echo.New()   e.Use(middleware.BodyLimit(strconv.Itoa(cfg.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: cfg.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", cfg.Port),  } }  func (s \*Server) Init(userService v1.UserService, tokenService v1.TokenService, authenticationService v1.AuthenticationService, validator v1.Validator, refreshTokenDuration time.Duration) \*Server {  log.Println("Initializing server...")  s.echo.GET("/swagger/\*", echoSwagger.WrapHandler)  s.echo.GET("/swagger", func(c echo.Context) error {  return c.Redirect(http.*StatusMovedPermanently*, "/swagger/index.html")  })   log.Println("Initializing api...")  handlerV1 := v1.New(userService, tokenService, authenticationService, validator, refreshTokenDuration)  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 (  "github.com/labstack/echo/v4"  "github.com/lapkomo2018/goTwitterServices/internal/auth/model"  "log"  "time" )  type (  UserService interface {  Register(username, email, password string) (string, error)  Login(login, password string) (string, string, error)  Activate(token string) (string, string, error)  Find(id uint) (\*model.User, error)  ChangePassword(id uint, password string) error  }   TokenService interface {  Generate(userID uint) (string, string, error)  Verify(accessToken string) (userID uint, err error)  VerifyRefreshToken(refreshToken string) (userID uint, err error)  }   AuthenticationService interface {  Auth(auth string) (\*model.User, error)  }   Validator interface {  Login(login string) error  Email(email string) error  Username(username string) error  Password(password string) error  }   Handler struct {  userService UserService  tokenService TokenService  authenticationService AuthenticationService  validator Validator  RefreshTokenDuration time.Duration  } )  const (  *userLocals* = "user"  *refreshTokenCookieName* = "refresh\_token" )  func New(userService UserService, tokenService TokenService, authenticationService AuthenticationService, validator Validator, refreshTokenDuration time.Duration) \*Handler {  return &Handler{  userService: userService,  tokenService: tokenService,  authenticationService: authenticationService,  validator: validator,  RefreshTokenDuration: refreshTokenDuration,  } }  func (h \*Handler) Init(api \*echo.Group) {  log.Println("Initializing V1 api")  v1 := api.Group("/v1")  {  h.initUserApi(v1)  h.initTokenApi(v1)  } } |

Код v1/middleware.go:

|  |
| --- |
| package v1  import (  "github.com/labstack/echo/v4"  "net/http" )  const (  *authorizationHeader* = "Authorization"  *authenticationHeader* = "Authentication" )  func (h \*Handler) authenticationMiddleware(next echo.HandlerFunc) echo.HandlerFunc {  return func(c echo.Context) error {  authHeader := c.Request().Header.Get(*authorizationHeader*)  if authHeader == "" {  return echo.NewHTTPError(http.*StatusUnauthorized*, "Missing Authorization header")  }   user, err := h.authenticationService.Auth(authHeader)  if err != nil {  return echo.NewHTTPError(http.*StatusUnauthorized*, err.Error())  }   c.Set(*userLocals*, user)   return next(c)  } } |

Код v1/token.go:

|  |
| --- |
| package v1  import (  "github.com/labstack/echo/v4"  "net/http"  "time" )  func (h \*Handler) initTokenApi(api \*echo.Group) {  api.POST("/refresh", h.refreshToken) }  // @Summary Refresh token // @Description Refresh jwt token // @Tags Token // @Accept json // @Produce json // @Param refresh\_token header string true "Refresh Token" // @Success 200 {object} accessTokenResp // @Header 200 {string} Set-Cookie "Refresh Token" // @Failure default {object} echo.HTTPError // @Router /refresh [post] func (h \*Handler) refreshToken(c echo.Context) error {  refreshTokenCookie, err := c.Cookie(*refreshTokenCookieName*)  if err != nil {  return echo.NewHTTPError(http.*StatusUnauthorized*, err.Error())  }   userID, err := h.tokenService.VerifyRefreshToken(refreshTokenCookie.Value)  if err != nil {  return echo.NewHTTPError(http.*StatusUnauthorized*, err.Error())  }   accessToken, refreshToken, err := h.tokenService.Generate(userID)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return h.setAndReturnTokens(c, accessToken, refreshToken) }  type (  accessTokenResp struct {  Token string `json:"token"`  } )  // func set refresh token cookie and return access token func (h \*Handler) setAndReturnTokens(c echo.Context, accessToken, refreshToken string) error {  c.SetCookie(&http.Cookie{  Name: *refreshTokenCookieName*,  Value: refreshToken,  Expires: time.Now().Add(h.RefreshTokenDuration),  HttpOnly: true,  SameSite: http.*SameSiteStrictMode*,  Path: "/",  })   return c.JSON(http.*StatusOK*, accessTokenResp{  Token: accessToken,  }) } |

Код v1/user.go:

|  |
| --- |
| package v1  import (  "github.com/labstack/echo/v4"  "github.com/lapkomo2018/goTwitterServices/internal/auth/model"  "net/http"  "time" )  func (h \*Handler) initUserApi(api \*echo.Group) {  api.POST("/login", h.userLogin)  api.POST("/register", h.userRegister)  api.GET("/activate/:token", h.userActivate)  api.GET("/authenticate", h.userAuthenticate, h.authenticationMiddleware)  api.GET("/info", h.userInfo, h.authenticationMiddleware)  api.PATCH("/change-password", h.userChangePassword, h.authenticationMiddleware) }  // @Summary User login // @Description login bruh // @Tags User // @Accept json // @Produce json // @Param input body userLoginReq true "user credentials" // @Success 200 {object} accessTokenResp // @Header 200 {string} Set-Cookie "Refresh Token" // @Failure default {object} echo.HTTPError // @Router /login [post] func (h \*Handler) userLogin(c echo.Context) error {  var body userLoginReq  if err := c.Bind(&body); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   if err := h.validator.Login(body.Login); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }  if err := h.validator.Password(body.Password); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   accessToken, refreshToken, err := h.userService.Login(body.Login, body.Password)  if err != nil {  if err.Error() == "user not found" || err.Error() == "invalid password" {  return echo.NewHTTPError(http.*StatusUnauthorized*, err.Error())  }  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return h.setAndReturnTokens(c, accessToken, refreshToken) }  type (  userLoginReq struct {  Login string `json:"login" binding:"required"`  Password string `json:"password" binding:"required"`  } )  // @Summary User register // @Description reg bruh // @Tags User // @Accept json // @Produce json // @Param input body userRegisterReq true "user credentials" // @Success 200 {object} userRegisterResp // @Failure default {object} echo.HTTPError // @Router /register [post] func (h \*Handler) userRegister(c echo.Context) error {  var body userRegisterReq  if err := c.Bind(&body); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   if err := h.validator.Email(body.Email); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }  if err := h.validator.Username(body.Username); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }  if err := h.validator.Password(body.Password); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   activationToken, err := h.userService.Register(body.Username, body.Email, body.Password)  if err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.JSON(http.*StatusOK*, userRegisterResp{  ActivationToken: activationToken,  }) }  type (  userRegisterReq struct {  Email string `json:"email" binding:"required"`  Username string `json:"username" binding:"required"`  Password string `json:"password" binding:"required"`  }  userRegisterResp struct {  ActivationToken string `json:"activation\_token"`  } )  // @Summary Activate // @Description Activate user // @Tags User // @Produce json // @Param token path string true "activation token" // @Success 200 {object} accessTokenResp // @Header 200 {string} Set-Cookie "Refresh Token" // @Failure default {object} echo.HTTPError // @Router /activate/{token} [get] func (h \*Handler) userActivate(c echo.Context) error {  activationToken := c.Param("token")   accessToken, refreshToken, err := h.userService.Activate(activationToken)  if err != nil {  return echo.NewHTTPError(http.*StatusUnauthorized*, err.Error())  }   return h.setAndReturnTokens(c, accessToken, refreshToken) }  // @Summary Authenticate // @Description Check user access token // @Tags User // @Produce json // @Security ApiKeyAuth // @Success 200 {object} userAuthenticateResp // @Failure default {object} echo.HTTPError // @Router /authenticate [get] func (h \*Handler) userAuthenticate(c echo.Context) error {  user, ok := c.Get(*userLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "invalid user")  }   return c.JSON(http.*StatusOK*, userAuthenticateResp{  UserID: user.ID,  }) }  type (  userAuthenticateResp struct {  UserID uint `json:"user\_id"`  } )  // @Summary Info // @Description Get users info // @Tags User // @Produce json // @Security ApiKeyAuth // @Success 200 {object} userInfoResp // @Failure default {object} echo.HTTPError // @Router /info [get] func (h \*Handler) userInfo(c echo.Context) error {  user, ok := c.Get(*userLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "invalid user")  }   return c.JSON(http.*StatusOK*, userInfoResp{  ID: user.ID,  Email: user.Email,  Username: user.Username,  CreatedAt: user.CreatedAt,  UpdatedAt: user.UpdatedAt,  }) }  type (  userInfoResp struct {  ID uint `json:"id"`  Email string `json:"email"`  Username string `json:"username"`  CreatedAt time.Time `json:"created\_at"`  UpdatedAt time.Time `json:"updated\_at"`  } )  // @Summary Change Password // @Description Change user password // @Tags User // @Accept json // @Produce json // @Security ApiKeyAuth // @Param input body userChangePasswordReq true "new password" // @Success 204 // @Failure default {object} echo.HTTPError // @Router /change/password [patch] func (h \*Handler) userChangePassword(c echo.Context) error {  user, ok := c.Get(*userLocals*).(\*model.User)  if !ok {  return echo.NewHTTPError(http.*StatusInternalServerError*, "invalid user")  }   var body userChangePasswordReq  if err := c.Bind(&body); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   if err := h.validator.Password(body.Password); err != nil {  return echo.NewHTTPError(http.*StatusBadRequest*, err.Error())  }   if err := h.userService.ChangePassword(user.ID, body.Password); err != nil {  return echo.NewHTTPError(http.*StatusInternalServerError*, err.Error())  }   return c.NoContent(http.*StatusNoContent*) }  type (  userChangePasswordReq struct {  Password string `json:"password" binding:"required"`  } ) |

Створення Proto файлу для grpc.

****

Рисунок 9 — Структура api папки з .proto файлами

Код auth.proto:

|  |
| --- |
| syntax = "proto3"; option go\_package = "/gen";  service AuthService {  rpc CheckAuth(CheckAuthRequest) returns (CheckAuthResponse);  rpc UpdateUsernameEmail(UpdateUsernameEmailRequest) returns (UpdateUsernameEmailResponse);  rpc DeleteUser(DeleteUserRequest) returns (DeleteUserResponse); }  message CheckAuthRequest { }  message CheckAuthResponse {  uint64 user\_id = 1; }  message UpdateUsernameEmailRequest {  uint64 user\_id = 1;  string username = 2;  string email = 3; }  message UpdateUsernameEmailResponse {  bool success = 1; }  message DeleteUserRequest {  uint64 user\_id = 1; }  message DeleteUserResponse {  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/auth.proto  @move /Y .\pkg\gen\api\auth.pb.go .\pkg\gen\auth.pb.go  @move /Y .\pkg\gen\api\auth\_grpc.pb.go .\pkg\gen\auth\_grpc.pb.go  @rd /s /q .\pkg\gen\api |

Реалізація Grpc сервера з підключенням API v1.

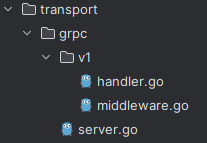
****

Рисунок 10 — Структура grpc в transport layer

Код server.go:

|  |
| --- |
| package grpc  import (  "fmt"  "github.com/lapkomo2018/goTwitterServices/internal/auth/transport/grpc/v1"  "github.com/lapkomo2018/goTwitterServices/pkg/gen"  "google.golang.org/grpc"  "google.golang.org/grpc/reflection"  "log"  "net" )  type (  Config struct {  Port int  }   Server struct {  addr string  grpcServer \*grpc.Server  } )  func New(cfg Config) \*Server {  log.Printf("Creating grpc server with port: %d", cfg.Port)  grpcServ := grpc.NewServer()  reflection.Register(grpcServ)   return &Server{  addr: fmt.Sprintf(":%d", cfg.Port),  grpcServer: grpcServ,  } }  func (s \*Server) Init(as v1.AuthenticationService, us v1.UserService) \*Server {  handler := v1.New(as, us)  gen.RegisterAuthServiceServer(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) } |

Код v1/handler.go:

|  |
| --- |
| package v1  import (  "context"  "github.com/lapkomo2018/goTwitterServices/internal/auth/model"  "github.com/lapkomo2018/goTwitterServices/pkg/gen" )  type (  AuthenticationService interface {  Auth(auth string) (\*model.User, error)  }   UserService interface {  ChangeEmail(id uint, email string) error  ChangeUsername(id uint, username string) error  Delete(id uint) error  }   Handler struct {  gen.UnimplementedAuthServiceServer  as AuthenticationService  us UserService  } )  func New(as AuthenticationService, us UserService) \*Handler {  return &Handler{  as: as,  us: us,  } }  func (h \*Handler) CheckAuth(ctx context.Context, r \*gen.CheckAuthRequest) (\*gen.CheckAuthResponse, error) {  user, err := h.getUserFromMetadata(ctx)  if err != nil {  return nil, err  }   return &gen.CheckAuthResponse{UserId: uint64(user.ID)}, nil }  // *TODO: Fix this (Need update credentials for user from request id)* /\*func (h \*Handler) UpdateUsernameEmail(ctx context.Context, r \*gen.UpdateUsernameEmailRequest) (\*gen.UpdateUsernameEmailResponse, error) {  user, err := h.getUserFromMetadata(ctx)  if err != nil {  return &gen.UpdateUsernameEmailResponse{Success: false}, err  }   if r.Username != "" {  err = h.us.ChangeUsername(user.ID, r.Username)  if err != nil {  return &gen.UpdateUsernameEmailResponse{Success: false}, err  }  }   if r.Email != "" {  err = h.us.ChangeEmail(user.ID, r.Email)  if err != nil {  return &gen.UpdateUsernameEmailResponse{Success: false}, err  }  }   return &gen.UpdateUsernameEmailResponse{Success: true}, nil }\*/  // *TODO: Fix this (Need delete user from request id not from token)* /\*func (h \*Handler) DeleteUser(ctx context.Context, r \*gen.DeleteUserRequest) (\*gen.DeleteUserResponse, error) {  user, err := h.getUserFromMetadata(ctx)  if err != nil {  return &gen.DeleteUserResponse{Success: false}, err  }   err = h.us.Delete(user.ID)  if err != nil {  return &gen.DeleteUserResponse{Success: false}, err  }   return &gen.DeleteUserResponse{Success: true}, nil }\*/ |

Код v1/middleware.go:

|  |
| --- |
| package v1  import (  "context"  "github.com/lapkomo2018/goTwitterServices/internal/auth/model"  "google.golang.org/grpc/metadata"  "google.golang.org/grpc/status"  "net/http" )  const (  *authorizationMetadata* = "authorization" )  func (h \*Handler) getUserFromMetadata(ctx context.Context) (\*model.User, error) {  md, ok := metadata.FromIncomingContext(ctx)  if !ok {  return nil, status.Errorf(http.*StatusUnauthorized*, "missing metadata")  }   authHeader, ok := md[*authorizationMetadata*]  if !ok || len(authHeader) == 0 {  return nil, status.Errorf(http.*StatusUnauthorized*, "missing Authorization header")  }   token := authHeader[0]  return h.as.Auth(token) } |

**Завдання №5.** Створення config.go для парсингу конфігурації сервера

Реалізація config системи.

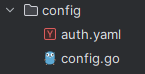
****

Рисунок 11 — Структура config папки

Код auth.yaml:

|  |
| --- |
| service:  name: "AuthService"  restServer:  port: 8080  bodyLimit: 25165824  allowedOrigins:  - "\*"  grpcServer:  port: 8081  jwt:  tokenDuration: 5m  refreshDuration: 168h *# 7 days* validator:  emailRegex: "^[a-z0-9.\_%+\\-]+@[a-z0-9.\\-]+\\.[a-z]{2,4}$"  usernameRegex: "^[a-zA-Z0-9]{3,20}$"  passwordRegex: "^(?=.\*[A-Za-z])(?=.\*\\d)[A-Za-z\\d]{8,}$" |

Код config.go:

|  |
| --- |
| package config  import (  "log"   "github.com/spf13/viper" )  type Service struct {  Name string }  func LoadConfig[T any]() (\*T, error) {  log.Println("Loading config...")   viper.AddConfigPath("./config")  viper.SetConfigName("config")  viper.SetConfigType("yaml")  if err := viper.ReadInConfig(); err != nil {  return nil, err  }   var config T  if err := viper.Unmarshal(&config); err != nil {  return nil, err  }   log.Println("Loaded config")  return &config, nil } |

**Завдання №6.** Створення main.go, Dockerfile файлів для запуску, та docker-compose.yml для запуску сервісу з бд

Реалізація Auth main.go файла для запуску програми.

****

Рисунок 12 — Структура cmd/auth папки

Код 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/auth/main.go  FROM alpine:latest COPY --from=*builder* server . COPY --from=*builder* /app/config/auth.yaml /config/config.yaml CMD ["/server"] |

Код main.go:

|  |
| --- |
| package main  import (  "context"  "log"  "os"  "os/signal"  "syscall"  "time"   "github.com/lapkomo2018/goTwitterServices/config"  "github.com/lapkomo2018/goTwitterServices/pkg/discovery"  "github.com/lapkomo2018/goTwitterServices/pkg/discovery/consul"  "github.com/lapkomo2018/goTwitterServices/pkg/hash"  "github.com/lapkomo2018/goTwitterServices/pkg/jwt"  "github.com/lapkomo2018/goTwitterServices/pkg/validation"   userService "github.com/lapkomo2018/goTwitterServices/internal/auth/gateway/user/grpc"  "github.com/lapkomo2018/goTwitterServices/internal/auth/model"  "github.com/lapkomo2018/goTwitterServices/internal/auth/repository/mysql"  "github.com/lapkomo2018/goTwitterServices/internal/auth/service"  "github.com/lapkomo2018/goTwitterServices/internal/auth/transport/grpc"  "github.com/lapkomo2018/goTwitterServices/internal/auth/transport/rest" )  type Config struct {  Service config.Service  RestServer rest.Config  GrpcServer grpc.Config  JWT jwt.Config  Validator validation.Config }  const (  *TagRest* = "rest"  *TagGRPC* = "grpc" )  var (  cfg \*Config  env \*model.Env )  func init() {  var err error   env, err = model.ParseEnv()  if err != nil {  log.Fatal(err)  }   cfg, err = config.LoadConfig[Config]()  if err != nil {  log.Fatal(err)  } }  // @title Twitter Auth Service // @version 1.0 // @description Twitter Auth Service  // @securityDefinitions.apikey ApiKeyAuth // @in header // @name Authorization  // @host localhost:8080 // @BasePath /api/v1 func main() {  registry, err := consul.NewRegistry(env.DiscoveryAddr)  if err != nil {  log.Fatalf("could not create registry: %v", err)  }  ctx := context.Background()   // Register REST service  restInstanceID := discovery.GenerateInstanceID(cfg.Service.Name, env.ExternalRestPort)  if err := registry.Register(ctx, restInstanceID, cfg.Service.Name, env.ExternalRestPort, []string{*TagRest*}); err != nil {  log.Fatalf("could not register rest service: %v", err)  }  defer registry.Deregister(ctx, restInstanceID, cfg.Service.Name)   // Register gRPC service  grpcInstanceID := discovery.GenerateInstanceID(cfg.Service.Name, env.ExternalGrpcPort)  if err := registry.Register(ctx, grpcInstanceID, cfg.Service.Name, env.ExternalGrpcPort, []string{*TagGRPC*}); err != nil {  log.Fatalf("could not register grpc service: %v", err)  }  defer registry.Deregister(ctx, grpcInstanceID, cfg.Service.Name)   go func() {  for {  if err := registry.ReportHealthyState(restInstanceID, cfg.Service.Name); err != nil {  log.Printf("could not report rest healthy state: %v", err)  }  if err := registry.ReportHealthyState(grpcInstanceID, cfg.Service.Name); err != nil {  log.Printf("could not report grpc healthy state: %v", err)  }  time.Sleep(1 \* time.*Second*)  }  }()   storages, err := mysql.New(env.DB)  if err != nil {  log.Fatal(err)  }  validator, err := validation.NewValidator(cfg.Validator)  if err != nil {  log.Fatal(err)  }  hasher := hash.NewSHA1Hasher(env.HashSalt)  tokenManager := jwt.NewManager(cfg.JWT, env.JWTSecret, env.JWTRefreshSecret)  userGateway := userService.New(registry)  services := service.New(storages.User, storages.RefreshToken, storages.ActivationToken, tokenManager, hasher, userGateway)   go func() {  server := rest.New(cfg.RestServer).Init(services.User, services.Tokens, services.Authentication, validator, cfg.JWT.RefreshDuration)  if err := server.Run(); err != nil {  log.Fatalf("Rest server err: %v", err)  }  }()   go func() {  server := grpc.New(cfg.GrpcServer).Init(services.Authentication, services.User)  if err := server.Run(); err != nil {  log.Fatalf("Grpc server err: %v", err)  }  }()   quit := make(chan os.Signal, 1)  signal.Notify(quit, syscall.*SIGINT*, syscall.*SIGTERM*)  <-quit  log.Println("Shutting down servers...") } |

Реалізація docker-compose.yml файла для запуску сервісу з бд та consul.

Код docker-compose.yml:

|  |
| --- |
| services:  *# Consul service* consul:  image: consul:1.15.4  container\_name: consul  ports:  - "8500:8500"  networks:  - consul   *# Auth service* auth-service:  build:  context: .  dockerfile: cmd/auth/Dockerfile  container\_name: auth-service  environment:  DB: myuser:strongpass@tcp(auth-db:3306)/mydb?parseTime=true  DISCOVERY\_ADDR: consul:8500  EXTERNAL\_REST\_PORT: 8080  EXTERNAL\_GRPC\_PORT: 8081  HASH\_SALT: "hashSalt"  JWT\_SECRET: "accessPass"  JWT\_REFRESH: "refreshPass"  ports:  - "8080:8080" *# Rest* - "8081:8081" *# gRPC* networks:  - consul  - 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   volumes:  auth\_db\_data:  driver: local  driver\_opts:  type: none  o: bind  device: ./db/auth   networks:  consul:  auth: |