## Репозиторий github.com/pedrecho/vkr-activity

### .github/workflows/activity-chart.yaml

```
name: Publish activity Helm Chart

on:
  push:
    tags:
      - 'chart-*'

jobs:
  helm:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Helm
        uses: azure/setup-helm@v3

      - name: Log in to GHCR
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ secrets.GHCR_USERNAME }}
          password: ${{ secrets.GHCR_TOKEN }}

      - name: Extract Chart Version
        id: chart
        run: |
          echo "VERSION=$(grep '^version:' activity-chart/Chart.yaml | awk
'{print $2}')" >> $GITHUB_OUTPUT

      - name: Package Helm Chart
        run: helm package activity-chart

      - name: Push Helm Chart to GHCR
        run: |
          helm push activity-chart-${{ steps.chart.outputs.VERSION }}.tgz
oci://ghcr.io/${{ secrets.GHCR_USERNAME }}
```

### .github/workflows/app.yaml

```
name: Build & Push activity-app to GHCR

on:
  push:
    branches: [master]
  workflow_dispatch:

jobs:
  docker:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v3
```

```yaml
      - name: Log in to GHCR
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ secrets.GHCR_USERNAME }}
          password: ${{ secrets.GHCR_TOKEN }}

      - name: Build and Push Docker Image
        uses: docker/build-push-action@v5
        with:
          context: .
          file: docker/activity-app/Dockerfile
          push: true
          tags: ghcr.io/${{ secrets.GHCR_USERNAME }}/activity-app:latest
          build-args: |
            GITHUB_TOKEN=${{ secrets.GOPRIVATE_PAT }}
```

## .github/workflows/migrations.yaml

```yaml
name: Build & Push activity-migrations to GHCR

on:
  push:
    branches: [master]
    paths:
      - 'migrations/**'
      - 'docker/activity-migrations/**'
  workflow_dispatch:

jobs:
  docker:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Log in to GHCR
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ secrets.GHCR_USERNAME }}
          password: ${{ secrets.GHCR_TOKEN }}

      - name: Build and Push Docker Image
        uses: docker/build-push-action@v5
        with:
          context: .
          file: docker/activity-migrations/Dockerfile
          push: true
          tags: ghcr.io/${{ secrets.GHCR_USERNAME }}/activity-
migrations:latest
```

## Makefile

```makefile
# ==== Docker image ====
IMAGE_NAME := ghcr.io/pedrecho/activity-app
DOCKER_TAG := latest

docker-build:
	docker build -t $(IMAGE_NAME):$(DOCKER_TAG) .
```

```
docker-push:
	docker push $(IMAGE_NAME):$(DOCKER_TAG)


# ==== Helm chart ====
CHART_NAME := activity-chart
CHART_VERSION := $(shell grep "^version:" $(CHART_NAME)/Chart.yaml | awk
"{print $$2}")
RELEASE_NAME := activity

helm-tag:
	git tag chart-$(CHART_VERSION)
	@echo "Created local tag: chart-$(CHART_VERSION)"
	@echo "To push it: git push origin chart-$(CHART_VERSION)"

helm-install:
	helm upgrade --install $(RELEASE_NAME) $(CHART_NAME) \
		-f ./activity-chart/values.yaml \
		-f ./activity-chart/values.secret.yaml

helm-uninstall:
	helm uninstall $(RELEASE_NAME) || true

helm-clean:
	kubectl delete all -l app=$(RELEASE_NAME) || true
```

## activity-chart/Chart.yaml

```
apiVersion: v2
name: activity-chart
description: A Helm chart for deploying activity-app with PostgreSQL
version: 0.1.2
```

## activity-chart/files/config.yaml

```
server:
  port: {{ .Values.activityApp.containerPort }}

logger:
  level: debug

nats:
  connection:
    host: {{ .Values.nats.connection.host }}
    port: {{ .Values.nats.connection.port }}
    ssl: {{ .Values.nats.connection.ssl }}
    durable_name: {{ .Values.nats.connection.durableName }}
  topics:

postgres:
  host: {{ .Values.database.host }}
  port: {{ .Values.database.port }}
  user: {{ .Values.database.user }}
  password: {{ .Values.database.password }}
  dbname: {{ .Values.database.name }}
  ssl: false
```

## activity-chart/templates/activity-app-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: activity-app
spec:
  replicas: {{ .Values.activityApp.replicaCount }}
  selector:
    matchLabels:
      app: activity-app
  template:
    metadata:
      labels:
        app: activity-app
    spec:
      serviceAccountName: {{ .Values.serviceAccount.name }}
      containers:
        - name: activity-app
          image: {{ .Values.activityApp.image }}
          imagePullPolicy: {{ .Values.activityApp.imagePullPolicy }}
          ports:
            - containerPort: {{ .Values.activityApp.containerPort }}
          command: ["./activity-app"]
          args: ["--config", "{{ .Values.activityApp.configPath }}"]
          volumeMounts:
            - name: config-volume
              mountPath: {{ .Values.activityApp.configPath }}
              subPath: config.yaml
      volumes:
        - name: config-volume
          configMap:
            name: activity-app-config
```

## activity-chart/templates/activity-app-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: activity-app
spec:
  selector:
    app: activity-app
  ports:
    - port: {{ .Values.activityApp.externalPort }}
      targetPort: {{ .Values.activityApp.containerPort }}
  type: ClusterIP
```

## activity-chart/templates/configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: activity-app-config
  labels:
    app: activity-app

data:
  config.yaml: |-
{{ tpl (.Files.Get "files/config.yaml") . | indent 4 }}
```

## activity-chart/templates/migration-job.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: activity-db-migrate
  labels:
    app: activity-db
  annotations:
    "helm.sh/hook": post-install,post-upgrade
    # "helm.sh/hook-delete-policy": hook-succeeded
spec:
  template:
    spec:
      serviceAccountName: {{ .Values.serviceAccount.name }}
      restartPolicy: OnFailure
      containers:
        - name: migrate
          image: "{{ .Values.migrations.image.repository }}:{{
.Values.migrations.image.tag }}"
          imagePullPolicy: {{ .Values.migrations.imagePullPolicy }}
          args:
            - "-source=file:///migrations"
            - "-database=$(DB_URL)"
            - "up"
          env:
            - name: DB_URL
              value: "postgres://{{ .Values.database.user }}:{{
.Values.database.password }}@{{ .Values.database.host }}:{{
.Values.database.port }}/{{ .Values.database.name }}?sslmode=disable"
```

## activity-chart/templates/postgres-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: activity-db
spec:
  replicas: {{ .Values.database.replicaCount }}
  selector:
    matchLabels:
      app: activity-db
  template:
    metadata:
      labels:
        app: activity-db
    spec:
      serviceAccountName: {{ .Values.serviceAccount.name }}
      containers:
        - name: postgres
          image: {{ .Values.database.image }}
          ports:
            - containerPort: {{ .Values.database.containerPort }}
          env:
            - name: POSTGRES_PASSWORD
              value: "{{ .Values.database.password }}"
            - name: POSTGRES_DB
              value: "{{ .Values.database.name }}"
            - name: POSTGRES_USER
              value: "{{ .Values.database.user }}"
            - name: PGDATA
              value: /var/lib/postgresql/data/pgdata/data
          volumeMounts:
            - mountPath: /var/lib/postgresql/data/pgdata
              name: pgdata
      volumes:
        - name: pgdata
          persistentVolumeClaim:
            claimName: activity-db-pvc
```

## activity-chart/templates/postgres-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: activity-db-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: {{ .Values.database.storage }}
```

## activity-chart/templates/postgres-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: activity-db
spec:
  selector:
    app: activity-db
  ports:
    - port: {{ .Values.database.containerPort }}
```

## activity-chart/values.secret.example.yaml

```
database:
  password: "<db-password>"
```

## activity-chart/values.yaml

```
activityApp:
  image: ghcr.io/pedrecho/activity-app:latest
  imagePullPolicy: Always
  replicaCount: 1
  host: activity-app
  externalPort: 13804
  containerPort: 50051
  configPath: /app/config.yaml

database:
  image: ghcr.io/pedrecho/postgres:15
  replicaCount: 1
  containerPort: 5432
  host: activity-db
  port: 5432
  user: postgres
  name: activity
  storage: 1Gi

nats:
  connection:
    host: "nats"
    port: 4222
    ssl: false
    durableName: "activity-service"
  topics:

serviceAccount:
  name: vkr-ghcr-access

migrations:
  image:
    repository: ghcr.io/pedrecho/activity-migrations
    tag: latest
  imagePullPolicy: Always
```

## cmd/main.go

```
package main

import (
    "flag"
    "fmt"
    "github.com/pedrecho/vkr-activity/internal/app"
)

var (
    configPath = flag.String("config", "", "config path")
)

func main() {
    flag.Parse()

    app, err := app.New(*configPath)
    if err != nil {
       panic(fmt.Errorf("init app: %w", err))
    }

    if err = app.Run(); err != nil {
       panic(err)
    }
}
```

## docker/activity-app/Dockerfile

```
# Стадия сборки
FROM golang:1.24 AS builder

ARG GITHUB_TOKEN
ENV GOPRIVATE=github.com/pedrecho
ENV CGO_ENABLED=0

RUN git config --global url."https://${GITHUB_TOKEN}:x-oauth-
basic@github.com/".insteadOf "https://github.com/"

WORKDIR /app
COPY ../../go.mod go.sum ./
RUN go mod download

COPY ../.. ./
# Сборка бинарника activity-app вместо profile-app
RUN go build -o activity-app ./cmd/main.go

# Финальный образ
FROM alpine:3.19
```

```
WORKDIR /app
COPY --from=builder /app/activity-app .
RUN chmod +x ./activity-app
CMD ["./activity-app"]
```

## docker/activity-migrations/Dockerfile

```
FROM migrate/migrate:v4.15.2

COPY ./migrations /migrations

ENTRYPOINT ["/migrate"]
```

## go.mod

```
module github.com/pedrecho/vkr-activity

go 1.24.0

require (
    github.com/golang-jwt/jwt/v5 v5.2.2
    github.com/google/uuid v1.6.0
    github.com/pedrecho/vkr-pkg v0.0.0-20250508155451-5484a53270fe
    google.golang.org/grpc v1.70.0
    gopkg.in/yaml.v3 v3.0.1
)

require (
    github.com/klauspost/compress v1.18.0 // indirect
    github.com/lib/pq v1.10.9 // indirect
    github.com/nats-io/nats.go v1.39.1 // indirect
    github.com/nats-io/nkeys v0.4.9 // indirect
    github.com/nats-io/nuid v1.0.1 // indirect
    go.uber.org/multierr v1.10.0 // indirect
    go.uber.org/zap v1.27.0 // indirect
    golang.org/x/crypto v0.36.0 // indirect
    golang.org/x/net v0.38.0 // indirect
    golang.org/x/sys v0.31.0 // indirect
    golang.org/x/text v0.23.0 // indirect
    google.golang.org/genproto/googleapis/rpc v0.0.0-20241202173237-
19429a94021a // indirect
    google.golang.org/protobuf v1.36.5 // indirect
    gopkg.in/natefinch/lumberjack.v2 v2.2.1 // indirect
)
```

## internal/app/app.go

```
package app

import (
    "fmt"
    "github.com/pedrecho/vkr-activity/internal/config"
    "github.com/pedrecho/vkr-activity/internal/grpctransport"
    "github.com/pedrecho/vkr-activity/internal/repository/postgres"
    "github.com/pedrecho/vkr-activity/internal/service"
    "github.com/pedrecho/vkr-pkg/logger"
    pb "github.com/pedrecho/vkr-pkg/pb/activity"
    "google.golang.org/grpc"
    "net"
)
```

```go
type App struct {
	cfg *config.Config
}

func New(configPath string) (*App, error) {
	cfg, err := config.Load(configPath)
	if err != nil {
		return nil, fmt.Errorf("config init: %w", err)
	}

	return &App{
		cfg: cfg,
	}, nil
}

func (a *App) Run() error {
	zapSLogger, err := logger.NewZapSLogger(a.cfg.Logger)
	if err != nil {
		return fmt.Errorf("zaplogger init: %w", err)
	}

	zapSLogger.Info("content-app started")

	postgresRep, err := postgres.New(a.cfg.Postgres)
	if err != nil {
		return fmt.Errorf("init postgres: %w", err)
	}

	activitySvc := service.New(postgresRep)

	grpcSrv := grpc.NewServer()
	activityTransport := grpctransport.NewServer(zapSLogger, activitySvc)
	pb.RegisterActivityServiceServer(grpcSrv, activityTransport)

	listener, err := net.Listen("tcp", fmt.Sprintf(":%d", a.cfg.Server.Port))
	if err != nil {
		return fmt.Errorf("listen: %w", err)
	}

	zapSLogger.Infof("starting gRPC server on %d", a.cfg.Server.Port)
	return grpcSrv.Serve(listener)
}
```

## internal/config/config.go

```go
package config

import (
	"fmt"
	"github.com/pedrecho/vkr-pkg/db"
	"github.com/pedrecho/vkr-pkg/logger"
	"github.com/pedrecho/vkr-pkg/messaging"
	"gopkg.in/yaml.v3"
	"os"
)

type Config struct {
	Server   ServerConfig      `yaml:"server"`
	Logger   logger.ZapConfig  `yaml:"logger"`
	Postgres db.PostgresConfig `yaml:"postgres"`
	//todo client
	Nats NatsConfig `yaml:"nats"`
}
```

```go
func Load(filename string) (*Config, error) {
	file, err := os.ReadFile(filename)
	if err != nil {
		return nil, fmt.Errorf("read config file: %w", err)
	}
	cfg := Config{}
	err = yaml.Unmarshal(file, &cfg)
	if err != nil {
		return nil, fmt.Errorf("unmarshal config file: %w", err)
	}

	return &cfg, err
}

type ServerConfig struct {
	Port int `yaml:"port"`
}

type NatsConfig struct {
	Connection messaging.NatsConfig `yaml:"connection"`
	Topics     TopicsConfig         `yaml:"topics"`
}

type TopicsConfig struct {
}
```

## internal/grpctransport/route-comment.go

```go
package grpctransport

import (
	"context"
	"github.com/google/uuid"
	"github.com/pedrecho/vkr-activity/internal/token"
	pb "github.com/pedrecho/vkr-pkg/pb/activity"
	"google.golang.org/grpc/codes"
	"google.golang.org/grpc/status"
	"google.golang.org/protobuf/types/known/timestamppb"
)

func (s *Server) CreateRouteComment(ctx context.Context, req
*pb.CreateRouteCommentRequest) (*pb.CreateRouteCommentResponse, error) {
	userID, err := token.ExtractUserIDFromToken(ctx)
	if err != nil {
		s.log.Error("CreateRouteComment: extract user id", "error", err)
		return nil, status.Errorf(codes.Unauthenticated, "%v", err)
	}

	routeUUID, err := uuid.Parse(req.GetRouteId())
	if err != nil {
		s.log.Error("CreateRouteComment: invalid route id", "route_id",
req.GetRouteId(), "error", err)
		return nil, status.Errorf(codes.InvalidArgument, "invalid route_id:
%v", err)
	}

	commentModel, err := s.service.CreateRouteComment(ctx, routeUUID, userID,
req.GetCommentText())
	if err != nil {
		s.log.Error("CreateRouteComment: service error", "error", err)
		return nil, status.Errorf(codes.Internal, "could not create comment:
%v", err)
```

```
    }

    pbComment := &pb.Comment{
        CommentId: commentModel.CommentID,
        RouteId:   commentModel.RouteID.String(),
        UserId:    commentModel.UserID.String(),
        Text:      commentModel.Text,
        CreatedAt: timestamppb.New(commentModel.CreatedAt),
    }

    return &pb.CreateRouteCommentResponse{Comment: pbComment}, nil
}

func (s *Server) GetRouteComments(ctx context.Context, req
*pb.GetRouteCommentsRequest) (*pb.GetRouteCommentsResponse, error) {
    routeUUID, err := uuid.Parse(req.GetRouteId())
    if err != nil {
        s.log.Error("GetRouteComments: invalid route id", "route_id",
req.GetRouteId(), "error", err)
        return nil, status.Errorf(codes.InvalidArgument, "invalid route_id:
%v", err)
    }

    comments, err := s.service.GetRouteComments(ctx, routeUUID)
    if err != nil {
        s.log.Error("GetRouteComments: service error", "route_id", routeUUID,
"error", err)
        return nil, status.Errorf(codes.Internal, "could not get comments:
%v", err)
    }

    pbComments := make([]*pb.Comment, 0, len(comments))
    for _, c := range comments {
        pbComments = append(pbComments, &pb.Comment{
            CommentId: c.CommentID,
            RouteId:   c.RouteID.String(),
            UserId:    c.UserID.String(),
            Text:      c.Text,
            CreatedAt: timestamppb.New(c.CreatedAt),
        })
    }

    return &pb.GetRouteCommentsResponse{Comments: pbComments}, nil
}
```

## internal/grpctransport/route-like.go

```
package grpctransport

import (
    "context"
    "github.com/google/uuid"
    "google.golang.org/grpc/codes"
    "google.golang.org/grpc/status"
    "google.golang.org/protobuf/types/known/emptypb"

    "github.com/pedrecho/vkr-activity/internal/token"
    pb "github.com/pedrecho/vkr-pkg/pb/activity"
)

func (s *Server) LikeRoute(ctx context.Context, req *pb.LikeRouteRequest)
(*emptypb.Empty, error) {
    userID, err := token.ExtractUserIDFromToken(ctx)
```

```
	if err != nil {
		s.log.Error("LikeRoute: extract user id", "error", err)
		return nil, status.Errorf(codes.Unauthenticated, "%v", err)
	}

	routeUUID, err := uuid.Parse(req.GetRouteId())
	if err != nil {
		s.log.Error("LikeRoute: invalid route id", "route_id",
req.GetRouteId(), "error", err)
		return nil, status.Errorf(codes.InvalidArgument, "invalid route_id:
%v", err)
	}

	err = s.service.LikeRoute(ctx, routeUUID, userID)
	if err != nil {
		s.log.Error("LikeRoute: service error", "route_id", routeUUID,
"user_id", userID, "error", err)
		return nil, status.Errorf(codes.Internal, "could not like route: %v",
err)
	}

	return &emptypb.Empty{}, nil
}

func (s *Server) UnlikeRoute(ctx context.Context, req *pb.UnlikeRouteRequest)
(*emptypb.Empty, error) {
	userID, err := token.ExtractUserIDFromToken(ctx)
	if err != nil {
		s.log.Error("UnlikeRoute: extract user id", "error", err)
		return nil, status.Errorf(codes.Unauthenticated, "%v", err)
	}

	routeUUID, err := uuid.Parse(req.GetRouteId())
	if err != nil {
		s.log.Error("UnlikeRoute: invalid route id", "route_id",
req.GetRouteId(), "error", err)
		return nil, status.Errorf(codes.InvalidArgument, "invalid route_id:
%v", err)
	}

	err = s.service.UnlikeRoute(ctx, routeUUID, userID)
	if err != nil {
		s.log.Error("UnlikeRoute: service error", "route_id", routeUUID,
"user_id", userID, "error", err)
		return nil, status.Errorf(codes.Internal, "could not unlike route:
%v", err)
	}

	return &emptypb.Empty{}, nil
}

func (s *Server) IsRouteLiked(ctx context.Context, req
*pb.IsRouteLikedRequest) (*pb.IsRouteLikedResponse, error) {
	userID, err := token.ExtractUserIDFromToken(ctx)
	if err != nil {
		s.log.Error("IsRouteLiked: extract user id", "error", err)
		return nil, status.Errorf(codes.Unauthenticated, "%v", err)
	}

	routeUUID, err := uuid.Parse(req.GetRouteId())
	if err != nil {
		s.log.Error("IsRouteLiked: invalid route id", "route_id",
req.GetRouteId(), "error", err)
		return nil, status.Errorf(codes.InvalidArgument, "invalid route_id:
```

```
%v", err)
    }

    liked, err := s.service.IsRouteLiked(ctx, routeUUID, userID)
    if err != nil {
       s.log.Error("IsRouteLiked: service error", "route_id", routeUUID,
"user_id", userID, "error", err)
       return nil, status.Errorf(codes.Internal, "could not check like
status: %v", err)
    }

    return &pb.IsRouteLikedResponse{Liked: liked}, nil
}
```

## internal/grpctransport/service.go

```
package grpctransport

import (
    "github.com/pedrecho/vkr-activity/internal/service"
    "github.com/pedrecho/vkr-pkg/logger"
    pb "github.com/pedrecho/vkr-pkg/pb/activity"
)

type Server struct {
    pb.UnimplementedActivityServiceServer
    log     logger.Logger
    service *service.Service
}

func NewServer(log logger.Logger, service *service.Service) *Server {
    return &Server{
       log:     log,
       service: service,
    }
}
```

## internal/models/route-comment.go

```
package models

import (
    "github.com/google/uuid"
    "time"
)

type RouteComment struct {
    CommentID int64
    RouteID   uuid.UUID
    UserID    uuid.UUID
    Text      string
    CreatedAt time.Time
}
```

## internal/models/route-like.go

```
package models

import (
    "time"
```

```go
    "github.com/google/uuid"
)

type RouteLike struct {
    RouteID   uuid.UUID
    UserID    uuid.UUID
    CreatedAt time.Time
}
```

## internal/models/token.go

```go
package models

import "github.com/golang-jwt/jwt/v5"

type AccessClaims struct {
    jwt.RegisteredClaims
}
```

## internal/repository/postgres/postgres.go

```go
package postgres

import (
    "database/sql"
    "fmt"
    "github.com/pedrecho/vkr-pkg/db"
)

type Postgres struct {
    db *sql.DB
}

func New(cfg db.PostgresConfig) (*Postgres, error) {
    sqlDB, err := db.PostgresConnect(cfg)
    if err != nil {
       return nil, fmt.Errorf("postgres connect: %w", err)
    }
    return &Postgres{
       db: sqlDB,
    }, nil
}
```

## internal/repository/postgres/route-comments.go

```go
package postgres

import (
    "context"
    "fmt"
    "github.com/google/uuid"
    "github.com/pedrecho/vkr-activity/internal/models"
)

func (p *Postgres) CreateRouteComment(ctx context.Context, routeID, userID
uuid.UUID, text string) (*models.RouteComment, error) {
    const query = `
       INSERT INTO route_comments (route_id, user_id, text)
       VALUES ($1, $2, $3)
       RETURNING comment_id, created_at
    `
    var comment models.RouteComment
    comment.RouteID = routeID
    comment.UserID = userID
    comment.Text = text
    // textscan returns generated ID and timestamp
    if err := p.db.QueryRowContext(ctx, query, routeID, userID,
text).Scan(&comment.CommentID, &comment.CreatedAt); err != nil {
        return nil, fmt.Errorf("CreateRouteComment exec: %w", err)
    }
    return &comment, nil
}

func (p *Postgres) GetRouteComments(ctx context.Context, routeID uuid.UUID)
([]*models.RouteComment, error) {
    const query = `
        SELECT comment_id, route_id, user_id, text, created_at
        FROM route_comments
        WHERE route_id = $1
        ORDER BY created_at ASC
    `

    rows, err := p.db.QueryContext(ctx, query, routeID)
    if err != nil {
        return nil, fmt.Errorf("GetRouteComments query: %w", err)
    }
    defer rows.Close()

    var comments []*models.RouteComment
    for rows.Next() {
        var c models.RouteComment
        if err := rows.Scan(&c.CommentID, &c.RouteID, &c.UserID, &c.Text,
&c.CreatedAt); err != nil {
            return nil, fmt.Errorf("GetRouteComments scan: %w", err)
        }
        comments = append(comments, &c)
    }
    if err := rows.Err(); err != nil {
        return nil, fmt.Errorf("GetRouteComments rows: %w", err)
    }
    return comments, nil
}
```

## internal/repository/postgres/route-likes.go

```
package postgres

import (
    "context"
    "database/sql"
    "errors"
    "fmt"

    "github.com/google/uuid"
)

func (p *Postgres) LikeRoute(ctx context.Context, routeID, userID uuid.UUID)
error {
    const query = `
        INSERT INTO route_likes (route_id, user_id)
        VALUES ($1, $2)
        ON CONFLICT DO NOTHING
    `
    if _, err := p.db.ExecContext(ctx, query, routeID, userID); err != nil {
       return fmt.Errorf("LikeRoute exec: %w", err)
    }
    return nil
}

func (p *Postgres) UnlikeRoute(ctx context.Context, routeID, userID
uuid.UUID) error {
    const query = `
       DELETE FROM route_likes
       WHERE route_id = $1 AND user_id = $2
    `

    if _, err := p.db.ExecContext(ctx, query, routeID, userID); err != nil {
       return fmt.Errorf("UnlikeRoute exec: %w", err)
    }
    return nil
}

func (p *Postgres) IsRouteLiked(ctx context.Context, routeID, userID
uuid.UUID) (bool, error) {
    const query = `
       SELECT EXISTS(
          SELECT 1 FROM route_likes WHERE route_id = $1 AND user_id = $2
       )
    `

    var exists bool
    if err := p.db.QueryRowContext(ctx, query, routeID,
userID).Scan(&exists); err != nil {
       if errors.Is(err, sql.ErrNoRows) {
          return false, nil
       }
       return false, fmt.Errorf("IsRouteLiked query: %w", err)
    }
    return exists, nil
}
```

## internal/service/route-comment.go

```
package service

import (
    "context"
    "fmt"
    "github.com/google/uuid"
    "github.com/pedrecho/vkr-activity/internal/models"
)

func (s *Service) CreateRouteComment(ctx context.Context, routeID, userID
uuid.UUID, text string) (*models.RouteComment, error) {
    comment, err := s.db.CreateRouteComment(ctx, routeID, userID, text)
    if err != nil {
       return nil, fmt.Errorf("service: create route comment failed: %w",
err)
    }
    return comment, nil
}

func (s *Service) GetRouteComments(ctx context.Context, routeID uuid.UUID)
([]*models.RouteComment, error) {
    comments, err := s.db.GetRouteComments(ctx, routeID)
```

```go
    if err != nil {
       return nil, fmt.Errorf("service: get route comments failed: %w", err)
    }
    return comments, nil
}
```

## internal/service/route-like.go

```go
package service

import (
    "context"
    "fmt"

    "github.com/google/uuid"
)

func (s *Service) LikeRoute(ctx context.Context, routeID, userID uuid.UUID)
error {
    if err := s.db.LikeRoute(ctx, routeID, userID); err != nil {
       return fmt.Errorf("service: like route failed: %w", err)
    }
    return nil
}

func (s *Service) UnlikeRoute(ctx context.Context, routeID, userID uuid.UUID)
error {
    if err := s.db.UnlikeRoute(ctx, routeID, userID); err != nil {
       return fmt.Errorf("service: unlike route failed: %w", err)
    }
    return nil
}

func (s *Service) IsRouteLiked(ctx context.Context, routeID, userID
uuid.UUID) (bool, error) {
    liked, err := s.db.IsRouteLiked(ctx, routeID, userID)
    if err != nil {
       return false, fmt.Errorf("service: check route liked failed: %w", err)
    }
    return liked, nil
}
```

## internal/service/service.go

```go
package service

import (
    "context"
    "github.com/google/uuid"
    "github.com/pedrecho/vkr-activity/internal/models"
    _ "image/jpeg"
    _ "image/png"
)

type Database interface {
    LikeRoute(ctx context.Context, routeID, userID uuid.UUID) error
    UnlikeRoute(ctx context.Context, routeID, userID uuid.UUID) error
    IsRouteLiked(ctx context.Context, routeID, userID uuid.UUID) (bool,
error)
    CreateRouteComment(ctx context.Context, routeID, userID uuid.UUID, text
string) (*models.RouteComment, error)
    GetRouteComments(ctx context.Context, routeID uuid.UUID)
([]*models.RouteComment, error)
}

type Service struct {
    db Database
}

func New(db Database) *Service {
    return &Service{db: db}
}
```

## internal/token/access.go

```go
package token

import (
    "context"
    "fmt"
    "github.com/golang-jwt/jwt/v5"
    "github.com/google/uuid"
    "github.com/pedrecho/vkr-activity/internal/models"
    "google.golang.org/grpc/metadata"
)

// ParseAccessToken парсит access token и возвращает claims без верификации
подписи.
func ParseAccessToken(tokenString string) (*models.AccessClaims, error) {
    claims := &models.AccessClaims{}

    _, _, err := jwt.NewParser().ParseUnverified(tokenString, claims)
    if err != nil {
       return nil, err
    }

    return claims, nil
}

func ExtractUserIDFromToken(ctx context.Context) (uuid.UUID, error) {
    md, ok := metadata.FromIncomingContext(ctx)
    if !ok {
       return uuid.Nil, fmt.Errorf("no metadata found")
    }

    authHeaders := md.Get("authorization")
    if len(authHeaders) == 0 {
       return uuid.Nil, fmt.Errorf("authorization header not found")
    }

    const bearerPrefix = "Bearer "
    tokenStr := authHeaders[0]
    if len(tokenStr) <= len(bearerPrefix) || tokenStr[:len(bearerPrefix)] !=
bearerPrefix {
       return uuid.Nil, fmt.Errorf("invalid bearer token format")
    }

    claims, err := ParseAccessToken(tokenStr[len(bearerPrefix):])
    if err != nil {
       return uuid.Nil, fmt.Errorf("parse token: %w", err)
    }

    uid, err := uuid.Parse(claims.Subject)
    if err != nil {
       return uuid.Nil, fmt.Errorf("invalid subject in token: %w", err)
    }

    return uid, nil
}
```

### migrations/20250508182700_route_likes.down.sql

```
DROP TABLE IF EXISTS route_likes;
```

### migrations/20250508182700_route_likes.up.sql

```
CREATE TABLE route_likes (
                             route_id    UUID    NOT NULL,
                             user_id     UUID    NOT NULL,
                             created_at  TIMESTAMPTZ NOT NULL DEFAULT NOW(),
                             PRIMARY KEY (route_id, user_id)
);

CREATE INDEX idx_route_likes_route_id ON route_likes(route_id);
```

### migrations/20250508182800_route_comments.down.sql

```
DROP TABLE IF EXISTS route_comments;
```

### migrations/20250508182800_route_comments.up.sql

```
CREATE TABLE route_comments (
                                comment_id  BIGSERIAL    PRIMARY KEY,
                                route_id    UUID         NOT NULL,
                                user_id     UUID         NOT NULL,
                                text        TEXT         NOT NULL,
                                created_at  TIMESTAMP WITH TIME ZONE DEFAULT
CURRENT_TIMESTAMP NOT NULL
);

CREATE INDEX idx_route_comments_route_id ON route_comments(route_id);
```

## Репозиторий github.com/pedrecho/vkr-auth

### .github/workflows/publish-auth-app.yaml

```
name: Build & Push auth-app to GHCR

on:
  push:
    branches: [master]
  workflow_dispatch:

jobs:
  docker:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v3
```

```yaml
      - name: Log in to GHCR
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ secrets.GHCR_USERNAME }}
          password: ${{ secrets.GHCR_TOKEN }}

      - name: Build and Push Docker Image
        uses: docker/build-push-action@v5
        with:
          context: .
          file: docker/auth-app/Dockerfile
          push: true
          tags: ghcr.io/${{ secrets.GHCR_USERNAME }}/auth-app:latest
          build-args: |
            GITHUB_TOKEN=${{ secrets.GOPRIVATE_PAT }}
```

## .github/workflows/publish-auth-chart.yaml

```yaml
name: Publish auth Helm Chart

on:
  push:
    tags:
      - 'chart-*'

jobs:
  helm:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Helm
        uses: azure/setup-helm@v3

      - name: Log in to GHCR
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ secrets.GHCR_USERNAME }}
          password: ${{ secrets.GHCR_TOKEN }}

      - name: Extract Chart Version
        id: chart
        run: echo "VERSION=$(grep '^version:' auth-chart/Chart.yaml | awk
'{print $2}')" >> $GITHUB_OUTPUT

      - name: Package Helm Chart
        run: helm package auth-chart

      - name: Push Helm Chart to GHCR
        run: helm push auth-chart-${{ steps.chart.outputs.VERSION }}.tgz
oci://ghcr.io/${{ secrets.GHCR_USERNAME }}
```

## .github/workflows/publish-auth-migrations.yaml

```yaml
name: Build & Push auth-migrations to GHCR

on:
  push:
    branches: [master]
    paths:
      - 'migrations/**'
      - 'docker/auth-migrations/**'
  workflow_dispatch:

jobs:
  docker:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Log in to GHCR
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ secrets.GHCR_USERNAME }}
          password: ${{ secrets.GHCR_TOKEN }}

      - name: Build and Push Docker Image
        uses: docker/build-push-action@v5
        with:
          context: .
          file: docker/auth-migrations/Dockerfile
          push: true
          tags: ghcr.io/${{ secrets.GHCR_USERNAME }}/auth-migrations:latest
```

## Makefile

```makefile
# ==== Docker image ====
IMAGE_NAME := ghcr.io/pedrecho/auth-app
DOCKER_TAG := latest

docker-build:
	docker build -t $(IMAGE_NAME):$(DOCKER_TAG) .

docker-push:
	docker push $(IMAGE_NAME):$(DOCKER_TAG)


# ==== Helm chart ====
CHART_NAME := auth-chart
CHART_VERSION := $(shell grep "^version:" $(CHART_NAME)/Chart.yaml | awk
"{print $$2}")
RELEASE_NAME := auth

helm-tag:
	git tag chart-$(CHART_VERSION)
	@echo "Created local tag: chart-$(CHART_VERSION)"
	@echo "To push it: git push origin chart-$(CHART_VERSION)"

helm-install:
	helm upgrade --install $(RELEASE_NAME) $(CHART_NAME) \
	   -f ./auth-chart/values.yaml \
	   -f ./auth-chart/values.secret.yaml

helm-uninstall:
	helm uninstall $(RELEASE_NAME) || true

helm-clean:
	kubectl delete all -l app=$(RELEASE_NAME) || true


# ==== DB ====
db-clean:
	kubectl delete pvc auth-db-pvc || true
```

## auth-chart/Chart.yaml

```yaml
apiVersion: v2
name: auth-chart
description: A Helm chart for deploying auth-app with PostgreSQL and Redis
version: 0.1.29
```

## auth-chart/files/config.yaml

```yaml
server:
  port: {{ .Values.authApp.containerPort }}

logger:
  level: debug

nats:
  connection:
    host: {{ .Values.nats.connection.host }}
    port: {{ .Values.nats.connection.port }}
    ssl: {{ .Values.nats.connection.ssl }}
    durable_name: {{ .Values.nats.connection.durableName }}
  topics:
    send_email_verification: {{ .Values.nats.topics.sendEmailVerification }}

postgres:
  host: {{ .Values.database.host }}
  port: {{ .Values.database.port }}
  user: {{ .Values.database.user }}
  password: {{ .Values.database.password }}
  dbname: {{ .Values.database.name }}
  ssl: false

redis:
  host: {{ .Values.cache.host }}
  port: {{ .Values.cache.port }}
  password: {{ .Values.cache.password }}
  db: 0
  ssl: false

user_client:
  host: {{ .Values.userService.host }}
```

## auth-chart/templates/auth-app-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-app
spec:
  replicas: {{ .Values.authApp.replicaCount }}
  selector:
    matchLabels:
      app: auth-app
  template:
    metadata:
      labels:
        app: auth-app
    spec:
      serviceAccountName: {{ .Values.serviceAccount.name }}
      containers:
        - name: auth-app
          image: {{ .Values.authApp.image }}
          imagePullPolicy: {{ .Values.authApp.imagePullPolicy }}
          ports:
            - containerPort: {{ .Values.authApp.containerPort }}
          command: ["./auth-app"]
          args: ["--config", "{{ .Values.authApp.configPath }}"]
          volumeMounts:
            - name: config-volume
              mountPath: {{ .Values.authApp.configPath }}
              subPath: config.yaml
      volumes:
        - name: config-volume
          configMap:
            name: auth-app-config
```

## auth-chart/templates/auth-app-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: auth-app
spec:
  selector:
    app: auth-app
  ports:
    - port: {{ .Values.authApp.externalPort }}
      targetPort: {{ .Values.authApp.containerPort }}
  type: ClusterIP
```

## auth-chart/templates/configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: auth-app-config
  labels:
    app: auth-app

data:
  config.yaml: |-
{{ tpl (.Files.Get "files/config.yaml") . | indent 4 }}
```

## auth-chart/templates/migration-job.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: auth-db-migrate
  labels:
    app: auth-db
  annotations:
    "helm.sh/hook": post-install
    "helm.sh/hook-delete-policy": before-hook-creation
spec:
  template:
    spec:
      serviceAccountName: {{ .Values.serviceAccount.name }}
      restartPolicy: OnFailure
      containers:
        - name: migrate
          image: "{{ .Values.migrations.image.repository }}:{{
.Values.migrations.image.tag }}"
          imagePullPolicy: {{ .Values.migrations.imagePullPolicy }}
          args:
            - "-source=file:///migrations"
            - "-database=$(DB_URL)"
            - "up"
          env:
            - name: DB_URL
              value: "postgres://{{ .Values.database.user }}:{{
.Values.database.password }}@{{ .Values.database.host }}:{{
.Values.database.port }}/{{ .Values.database.name }}?sslmode=disable"
```

auth-chart/templates/postgres-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-db
spec:
  replicas: {{ .Values.database.replicaCount }}
  selector:
    matchLabels:
      app: auth-db
  template:
    metadata:
      labels:
        app: auth-db
    spec:
      serviceAccountName: {{ .Values.serviceAccount.name }}
      containers:
        - name: postgres
          image: {{ .Values.database.image }}
          ports:
            - containerPort: {{ .Values.database.containerPort }}
          env:
            - name: POSTGRES_PASSWORD
              value: "{{ .Values.database.password }}"
            - name: POSTGRES_DB
              value: "{{ .Values.database.name }}"
            - name: POSTGRES_USER
              value: "{{ .Values.database.user }}"
            - name: PGDATA
              value: /var/lib/postgresql/data/pgdata/data
          volumeMounts:
            - mountPath: /var/lib/postgresql/data/pgdata
              name: pgdata
      volumes:
        - name: pgdata
          persistentVolumeClaim:
            claimName: auth-db-pvc
```

auth-chart/templates/postgres-pvc.yaml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: auth-db-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: {{ .Values.database.storage }}
```

## auth-chart/templates/postgres-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: auth-db
spec:
  selector:
    app: auth-db
  ports:
    - port: {{ .Values.database.containerPort }}
```

## auth-chart/templates/redis-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-cache
spec:
  replicas: {{ .Values.cache.replicaCount }}
  selector:
    matchLabels:
      app: auth-cache
  template:
    metadata:
      labels:
        app: auth-cache
    spec:
      serviceAccountName: {{ .Values.serviceAccount.name }}
      containers:
        - name: redis
          image: {{ .Values.cache.image }}
          ports:
            - containerPort: {{ .Values.cache.containerPort }}
```

## auth-chart/templates/redis-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: auth-cache
spec:
  selector:
    app: auth-cache
  ports:
    - port: {{ .Values.cache.containerPort }}
```

## auth-chart/values.secret.example.yaml

```yaml
database:
  password: "<db-password>"

cache:
  password: "<redis-password>"
```

## auth-chart/values.yaml

```yaml
authApp:
  image: ghcr.io/pedrecho/auth-app:latest
  imagePullPolicy: Always
  replicaCount: 1
  host: auth-app
  externalPort: 13801
  containerPort: 50051
  configPath: /app/config.yaml

database:
  image: ghcr.io/pedrecho/postgres:15
  replicaCount: 1
  containerPort: 5432
  host: auth-db
  port: 5432
  user: postgres
  name: auth
  storage: 1Gi

cache:
  image: redis:7
  replicaCount: 1
  containerPort: 6379
  host: auth-cache
  port: 6379
  db: 0
  ssl: false

nats:
  connection:
    host: "nats"
    port: 4222
    ssl: false
    durableName: "auth-service"
  topics:
    sendEmailVerification: "notifications.email.verification"

serviceAccount:
  name: vkr-ghcr-access

migrations:
  image:
    repository: ghcr.io/pedrecho/auth-migrations
    tag: latest
  imagePullPolicy: Always

userService:
  host: "profile-app:13804"
```

## cmd/main.go

```go
package main

import (
    "flag"
    "fmt"
    "github.com/pedrecho/vkr-auth/internal/app"
)

var (
    configPath = flag.String("config", "", "config path")
)

func main() {
    flag.Parse()

    app, err := app.New(*configPath)
    if err != nil {
       panic(fmt.Errorf("init app: %w", err))
    }

    if err = app.Run(); err != nil {
       panic(err)
    }
}
```

## docker/auth-app/Dockerfile

```dockerfile
# Стадия сборки
FROM golang:1.24 AS builder

ARG GITHUB_TOKEN
ENV GOPRIVATE=github.com/pedrecho
ENV CGO_ENABLED=0

RUN git config --global url."https://${GITHUB_TOKEN}:x-oauth-
basic@github.com/".insteadOf "https://github.com/"

WORKDIR /app
COPY ../../go.mod go.sum ./
RUN go mod download

COPY ../.. .
RUN go build -o auth-app ./cmd/main.go

# Финальный образ
FROM alpine:3.19
WORKDIR /app
COPY --from=builder /app/auth-app .
RUN chmod +x ./auth-app
CMD ["./auth-app"]
```

## docker/auth-migrations/Dockerfile

```dockerfile
FROM migrate/migrate:v4.15.2

COPY ./migrations /migrations

ENTRYPOINT ["/migrate"]
```

## go.mod

```
module github.com/pedrecho/vkr-auth

go 1.24

require (
    github.com/golang-jwt/jwt/v5 v5.2.2
    github.com/google/uuid v1.6.0
    github.com/lib/pq v1.10.9
    github.com/nats-io/nats.go v1.39.1
    github.com/pedrecho/vkr-pkg v0.0.0-20250505190913-de00c4a305c8
    github.com/redis/go-redis/v9 v9.7.1
    golang.org/x/crypto v0.36.0
    google.golang.org/grpc v1.70.0
    google.golang.org/protobuf v1.36.5
    gopkg.in/yaml.v3 v3.0.1
)

require (
    github.com/cespare/xxhash/v2 v2.3.0 // indirect
    github.com/dgryski/go-rendezvous v0.0.0-20200823014737-9f7001d12a5f //
indirect
    github.com/klauspost/compress v1.18.0 // indirect
    github.com/nats-io/nkeys v0.4.9 // indirect
    github.com/nats-io/nuid v1.0.1 // indirect
    go.uber.org/multierr v1.10.0 // indirect
    go.uber.org/zap v1.27.0 // indirect
    golang.org/x/net v0.38.0 // indirect
    golang.org/x/sys v0.31.0 // indirect
    golang.org/x/text v0.23.0 // indirect
    google.golang.org/genproto/googleapis/rpc v0.0.0-20241202173237-
19429a94021a // indirect
    gopkg.in/natefinch/lumberjack.v2 v2.2.1 // indirect
)
```

## internal/app/app.go

```
package app

import (
    "fmt"
    "github.com/pedrecho/vkr-auth/internal/config"
    "github.com/pedrecho/vkr-auth/internal/grpctransport"
    "github.com/pedrecho/vkr-auth/internal/repository/nats"
    "github.com/pedrecho/vkr-auth/internal/repository/postgres"
    "github.com/pedrecho/vkr-auth/internal/repository/redis"
    uc "github.com/pedrecho/vkr-auth/internal/repository/user"
    "github.com/pedrecho/vkr-auth/internal/service/token"
    "github.com/pedrecho/vkr-auth/internal/service/user"
    "github.com/pedrecho/vkr-pkg/logger"
    pb "github.com/pedrecho/vkr-pkg/pb/auth"
    "google.golang.org/grpc"
    "net"
)

type App struct {
    cfg *config.Config
}

func New(configPath string) (*App, error) {
    cfg, err := config.Load(configPath)
    if err != nil {
       return nil, fmt.Errorf("config init: %w", err)
    }
    return &App{
       cfg: cfg,
    }, nil
}

func (a *App) Run() error {
    zapSLogger, err := logger.NewZapSLogger(a.cfg.Logger)
    if err != nil {
       return fmt.Errorf("zaplogger init: %w", err)
    }

    zapSLogger.Info("auth-app started")

    db, err := postgres.New(a.cfg.Postgres)
    if err != nil {
       return fmt.Errorf("init postgres: %w", err)
    }

    redisClient, err := redis.New(a.cfg.Redis)
    if err != nil {
       return fmt.Errorf("init redis: %w", err)
    }

    natsClient, err := nats.New(a.cfg.Nats)
    if err != nil {
       return fmt.Errorf("init nats: %w", err)
    }

    client, err := uc.New(a.cfg.UserClient)
    if err != nil {
       return fmt.Errorf("init user: %w", err)
    }

    tokenService := token.New(a.cfg.Token, redisClient)

    userService := user.New(zapSLogger, db, redisClient, natsClient,
tokenService, client)

    grpcSrv := grpc.NewServer()
    authTransport := grpctransport.NewAuthService(userService, zapSLogger)
    pb.RegisterAuthServiceServer(grpcSrv, authTransport)

    listener, err := net.Listen("tcp", fmt.Sprintf(":%d", a.cfg.Server.Port))
    if err != nil {
       return fmt.Errorf("listen: %w", err)
    }

    zapSLogger.Infof("starting gRPC server on %d", a.cfg.Server.Port)
    return grpcSrv.Serve(listener)
}
```

## internal/config/config.go

```
package config

import (
    "fmt"
    "github.com/pedrecho/vkr-pkg/cache"
    "github.com/pedrecho/vkr-pkg/db"
    "github.com/pedrecho/vkr-pkg/logger"
    "github.com/pedrecho/vkr-pkg/messaging"
    "gopkg.in/yaml.v3"
```

```go
    "os"
)

type Config struct {
    Server     ServerConfig      `yaml:"server"`
    Logger     logger.ZapConfig  `yaml:"logger"`
    Postgres   db.PostgresConfig `yaml:"postgres"`
    Redis      cache.RedisConfig `yaml:"redis"`
    Nats       NatsConfig        `yaml:"nats"`
    Token      TokenConfig       `yaml:"token"`
    UserClient UserServiceConfig `yaml:"user_client"`
}

func Load(filename string) (*Config, error) {
    file, err := os.ReadFile(filename)
    if err != nil {
       return nil, fmt.Errorf("read config file: %w", err)
    }
    cfg := Config{}
    err = yaml.Unmarshal(file, &cfg)
    if err != nil {
       return nil, fmt.Errorf("unmarshal config file: %w", err)
    }

    return &cfg, err
}

type ServerConfig struct {
    Port int `yaml:"port"`
}

type NatsConfig struct {
    Connection messaging.NatsConfig `yaml:"connection"`
    Topics     TopicsConfig         `yaml:"topics"`
}

type TopicsConfig struct {
    SendEmailVerification string `yaml:"send_email_verification"`
}

type TokenConfig struct {
    //AccessTokenTTL  time.Duration `yaml:"access_duration"`
    //RefreshTokenTTL time.Duration `yaml:"refresh_duration"`
    SecretKey string `yaml:"secret_key"`
}

type UserServiceConfig struct {
    Host string `yaml:"host"`
}
```

## internal/dto/token.go

```go
package dto

import (
    "github.com/golang-jwt/jwt/v5"
    "time"
)

type TokenPair struct {
    AccessToken     string
    RefreshToken    string
    RefreshTokenTTL time.Duration
}

type AccessClaims struct {
    jwt.RegisteredClaims
}

type RefreshTokenData struct {
    UserID      string `json:"user_id"`
    Email       string `json:"email"`
    Fingerprint string `json:"fingerprint"`
}
```

## internal/dto/user.go

```go
package dto

import (
    "github.com/google/uuid"
    "time"
)

type UserStatus string

const (
    UserStatusPending   UserStatus = "pending"
    UserStatusConfirmed UserStatus = "confirmed"
)

type User struct {
    ID        uuid.UUID  `db:"id"`
    Email     string     `db:"email"`
    Password  string     `db:"password"`
    Status    UserStatus `db:"status"`
    CreatedAt time.Time  `db:"created_at"`
    UpdatedAt time.Time  `db:"updated_at"`
}
```

## internal/grpctransport/auth-service.go

```go
package grpctransport

import (
    "github.com/pedrecho/vkr-auth/internal/service/user"
    "github.com/pedrecho/vkr-pkg/logger"
    pb "github.com/pedrecho/vkr-pkg/pb/auth"
)

type AuthService struct {
    pb.UnimplementedAuthServiceServer
    userService *user.Service
    log         logger.Logger
}

func NewAuthService(userSvc *user.Service, log logger.Logger) *AuthService {
    return &AuthService{
       userService: userSvc,
       log:         log,
    }
}
```

## internal/grpctransport/login.go

```go
package grpctransport

import (
    "context"
    "errors"
    "strings"

    "github.com/pedrecho/vkr-auth/internal/service/user"
    pb "github.com/pedrecho/vkr-pkg/pb/auth"

    "google.golang.org/grpc/codes"
    "google.golang.org/grpc/status"
)

func (s *AuthService) Login(
    ctx context.Context,
    req *pb.LoginRequest,
) (*pb.LoginResponse, error) {

    if req == nil {
        return nil, status.Error(codes.InvalidArgument, "request must not be
nil")
    }
    email := strings.TrimSpace(req.Email)
    password := req.Password
    fingerprint := req.Fingerprint

    if email == "" || password == "" || fingerprint == "" {
        return nil, status.Error(codes.InvalidArgument, "email, password and
fingerprint are required")
    }

    tokenPair, err := s.userService.Login(ctx, email, password, fingerprint)
    if err != nil {
        switch {
        case errors.Is(err, user.ErrUserNotFound),
            errors.Is(err, user.ErrWrongPassword):
            return nil, status.Error(codes.Unauthenticated, "invalid email or
password")

        default:
            s.log.Errorf("login: %v", err)
            return nil, status.Error(codes.Internal, "internal server error")
        }
    }

    return &pb.LoginResponse{
        AccessToken:  tokenPair.AccessToken,
        RefreshToken: tokenPair.RefreshToken,
    }, nil
}
```

## internal/grpctransport/refresh-token.go

```go
package grpctransport

import (
    "context"
    "errors"
    "strings"

    "github.com/pedrecho/vkr-auth/internal/service/user"
    pb "github.com/pedrecho/vkr-pkg/pb/auth"
    "google.golang.org/grpc/codes"
    "google.golang.org/grpc/status"
)

func (s *AuthService) RefreshToken(
    ctx context.Context,
    req *pb.RefreshTokenRequest,
) (*pb.RefreshTokenResponse, error) {

    if req == nil {
       return nil, status.Error(codes.InvalidArgument, "request must not be
nil")
    }
    refreshToken := strings.TrimSpace(req.RefreshToken)
    fingerprint := req.Fingerprint

    if refreshToken == "" || fingerprint == "" {
       return nil, status.Error(codes.InvalidArgument, "refresh_token and
fingerprint are required")
    }

    tokens, err := s.userService.RefreshTokens(ctx, refreshToken,
fingerprint)
    if err != nil {
       switch {
       case errors.Is(err, user.ErrRefreshTokenNotFound):
          return nil, status.Error(codes.Unauthenticated, "refresh token not
found or expired") // 401
       case errors.Is(err, user.ErrRefreshTokenCorrupted):
          return nil, status.Error(codes.Unauthenticated, "refresh token data
corrupted") // 401
       case errors.Is(err, user.ErrFingerprintMismatch):
          return nil, status.Error(codes.PermissionDenied, "fingerprint
mismatch") // 403
       default:
          s.log.Errorf("refresh token error: %v", err)
          return nil, status.Error(codes.Internal, "internal server error")
// 500
       }
    }

    return &pb.RefreshTokenResponse{
       AccessToken:  tokens.AccessToken,
       RefreshToken: tokens.RefreshToken,
    }, nil
}
```

## internal/grpctransport/register-step-one.go

```
package grpctransport

import (
    "context"
    "errors"
    "github.com/pedrecho/vkr-auth/internal/dto"
    "github.com/pedrecho/vkr-auth/internal/service/user"
    pb "github.com/pedrecho/vkr-pkg/pb/auth"
    "google.golang.org/grpc/codes"
    "google.golang.org/grpc/status"
)

func (s *AuthService) RegisterStepOne(ctx context.Context, req
*pb.RegisterStepOneRequest) (*pb.RegisterStepOneResponse, error) {
    if req.GetEmail() == "" || req.GetPassword() == "" {
        s.log.Warn("missing email or password in RegisterStepOne")
        return nil, status.Error(codes.InvalidArgument, "email and password
are required")
    }

    _, err := s.userService.CreateUser(ctx, req.Email, req.Password,
dto.UserStatusPending)
    if err != nil {
        switch {
        case errors.Is(err, user.ErrUserAlreadyExists):
            s.log.Infof("user already exists: %s", req.Email)
            return nil, status.Error(codes.AlreadyExists, "user already
exists")

        case errors.Is(err, user.ErrInvalidEmail):
            s.log.Warnf("invalid email format: %s", req.Email)
            return nil, status.Error(codes.InvalidArgument, "invalid email
format")

        case errors.Is(err, user.ErrInvalidPassword):
            s.log.Warnf("invalid password for: %s", req.Email)
            return nil, status.Error(codes.InvalidArgument, "invalid password
format")

        default:
            s.log.Errorf("failed to create user: %v", err)
            return nil, status.Error(codes.Internal, "failed to create user")
        }
    }

    return &pb.RegisterStepOneResponse{
        Message: "verification code sent",
    }, nil
}
```

## internal/grpctransport/resend-verification.go

```
package grpctransport

import (
    "context"
    "errors"
    "github.com/pedrecho/vkr-auth/internal/service/user"
    "google.golang.org/grpc/codes"
    "google.golang.org/grpc/status"

    pb "github.com/pedrecho/vkr-pkg/pb/auth"
)

func (s *AuthService) ResendVerification(ctx context.Context, req
*pb.ResendVerificationRequest) (*pb.ResendVerificationResponse, error) {
    email := req.GetEmail()
    if email == "" {
        s.log.Warn("missing email in ResendVerification")
        return nil, status.Error(codes.InvalidArgument, "email is required")
    }

    err := s.userService.ResendVerificationCode(ctx, email)
    if err != nil {
        switch {
        case errors.Is(err, user.ErrInvalidEmail):
            s.log.Warnf("invalid email format: %s", email)
            return nil, status.Error(codes.InvalidArgument, "invalid email
format")

        case errors.Is(err, user.ErrUserNotFound):
            s.log.Infof("user not found: %s", email)
            return nil, status.Error(codes.NotFound, "user not found")

        case errors.Is(err, user.ErrUserNotPending):
            s.log.Infof("user already confirmed: %s", email)
            return nil, status.Error(codes.FailedPrecondition, "user already
confirmed")

        default:
            s.log.Errorf("resend verification failed: %v", err)
            return nil, status.Error(codes.Internal, "failed to resend
verification")
        }
    }

    return &pb.ResendVerificationResponse{
        Message: "verification code resent",
    }, nil
}
```

internal/grpctransport/validate-access-token.go

```
package grpctransport

import (
    "context"
    "errors"
    "strings"

    "google.golang.org/grpc/codes"
    "google.golang.org/grpc/metadata"
    "google.golang.org/grpc/status"

    "github.com/pedrecho/vkr-auth/internal/service/user"
    pb "github.com/pedrecho/vkr-pkg/pb/auth"
)

func (s *AuthService) ValidateAccessToken(
    ctx context.Context,
    _ *pb.ValidateAccessTokenRequest,
) (*pb.ValidateAccessTokenResponse, error) {

    md, ok := metadata.FromIncomingContext(ctx)
    if !ok {
        return nil, status.Error(codes.Unauthenticated, "missing metadata") //
401
    }

    auth := md.Get("authorization")
    if len(auth) == 0 {
        return nil, status.Error(codes.Unauthenticated, "authorization header
is required") // 401
    }

    parts := strings.Fields(auth[0])
    if len(parts) != 2 || !strings.EqualFold(parts[0], "Bearer") {
        return nil, status.Error(
            codes.InvalidArgument,
```

```go
			`authorization header must be in format "Bearer <token>"`, // 400
		)
	}
	tokenStr := parts[1]

	if err := s.userService.ValidateAccessToken(ctx, tokenStr); err != nil {
		switch {
		case errors.Is(err, user.ErrInvalidAccessToken):
			return nil, status.Error(codes.Unauthenticated, "invalid access 
token") // 401
		case errors.Is(err, user.ErrTokenExpired):
			return nil, status.Error(codes.Unauthenticated, "access token 
expired") // 401
		default:
			s.log.Errorf("validate access token: %v", err)
			return nil, status.Error(codes.Internal, "internal server error") 
// 500
		}
	}

	return &pb.ValidateAccessTokenResponse{
		Message: "token is valid",
	}, nil
}
```

## internal/grpctransport/verify-registration.go

```go
package grpctransport

import (
	"context"
	"errors"
	"github.com/pedrecho/vkr-auth/internal/service/user"
	pb "github.com/pedrecho/vkr-pkg/pb/auth"
	"google.golang.org/grpc/codes"
	"google.golang.org/grpc/status"
)

func (s *AuthService) VerifyRegistration(ctx context.Context, req 
*pb.VerifyRegistrationRequest) (*pb.VerifyRegistrationResponse, error) {
	email := req.GetEmail()
	code := req.GetVerificationCode()
	fingerprint := req.GetFingerprint()

	if email == "" || code == "" || fingerprint == "" {
		s.log.Warn("missing email, code or fingerprint in VerifyRegistration")
		return nil, status.Error(codes.InvalidArgument, "email, code and 
fingerprint are required")
	}

	tokens, err := s.userService.VerifyUser(ctx, email, code, fingerprint)
	if err != nil {
		switch {
		case errors.Is(err, user.ErrUserNotFound):
			s.log.Infof("user not found: %s", req.GetEmail())
			return nil, status.Error(codes.NotFound, "user not found")

		case errors.Is(err, user.ErrUserNotPending):
			s.log.Infof("user already confirmed: %s", req.GetEmail())
			return nil, status.Error(codes.FailedPrecondition, "user already 
confirmed")

		case errors.Is(err, user.ErrVerificationCodeNotFound):
			s.log.Infof("verification code not found for: %s", req.GetEmail())
			return nil, status.Error(codes.NotFound, "verification code not
found")

		case errors.Is(err, user.ErrInvalidVerificationCode),
			errors.Is(err, user.ErrInvalidEmail),
			errors.Is(err, user.ErrInvalidFingerprint):
			s.log.Infof("invalid input for: %s", req.GetEmail())
			return nil, status.Error(codes.InvalidArgument, "invalid input")

		default:
			s.log.Errorf("verify user failed: %v", err)
			return nil, status.Error(codes.Internal, "failed to verify user")
		}
	}

	return &pb.VerifyRegistrationResponse{
		AccessToken:  tokens.AccessToken,
		RefreshToken: tokens.RefreshToken,
	}, nil
}
```

## internal/repository/nats/nats.go

```
package nats

import (
	"fmt"

	"github.com/nats-io/nats.go"
	"github.com/pedrecho/vkr-auth/internal/config"
	"github.com/pedrecho/vkr-pkg/messaging"
)

type Nats struct {
	conn   *nats.Conn
	js     nats.JetStreamContext
	topics config.TopicsConfig
}

func New(cfg config.NatsConfig) (*Nats, error) {
	conn, js, err := messaging.NatsJetStreamConnect(cfg.Connection)
	if err != nil {
		return nil, fmt.Errorf("connect to nats: %w", err)
	}

	return &Nats{
		conn:   conn,
		js:     js,
		topics: cfg.Topics,
	}, nil
}
```

## internal/repository/nats/verification.go

```
package nats

import (
	"context"
	"fmt"
	"github.com/pedrecho/vkr-pkg/pb/events"
	"google.golang.org/protobuf/proto"
)

func (n *Nats) PublishEmailVerification(ctx context.Context, email string,
code int32) error {
    msg := &events.SendEmailVerificationRequest{
        Email:            email,
        ConfirmationCode: code,
    }

    data, err := proto.Marshal(msg)
    if err != nil {
        return fmt.Errorf("marshal proto: %w", err)
    }

    _, err = n.js.Publish(n.topics.SendEmailVerification, data)
    if err != nil {
        return fmt.Errorf("publish message: %w", err)
    }

    return nil
}
```

## internal/repository/postgres/errors.go

```
package postgres

import "errors"

var (
    ErrDuplicateEmail = errors.New("duplicate email")
    ErrUserNotFound   = errors.New("user not found")
)
```

## internal/repository/postgres/postgres.go

```
package postgres

import (
    "database/sql"
    "fmt"
    "github.com/pedrecho/vkr-pkg/db"
)

type Postgres struct {
    db *sql.DB
}

func New(cfg db.PostgresConfig) (*Postgres, error) {
    sqlDB, err := db.PostgresConnect(cfg)
    if err != nil {
        return nil, fmt.Errorf("postgres connect: %w", err)
    }
    return &Postgres{
        db: sqlDB,
    }, nil
}
```

## internal/repository/postgres/user-status.go

```
package postgres
```

```go
import (
    "context"
    "fmt"
    "github.com/google/uuid"
    "github.com/pedrecho/vkr-auth/internal/dto"
)

func (p *Postgres) UpdateUserStatus(ctx context.Context, id uuid.UUID, status
dto.UserStatus) error {
    const query = `
        UPDATE users
        SET status = $1, updated_at = NOW()
        WHERE id = $2
    `

    res, err := p.db.ExecContext(ctx, query, status, id)
    if err != nil {
        return fmt.Errorf("update user status: %w", err)
    }

    affected, err := res.RowsAffected()
    if err != nil {
        return fmt.Errorf("check update rows: %w", err)
    }
    if affected == 0 {
        return ErrUserNotFound
    }

    return nil
}
```

internal/repository/postgres/user.go

```go
package postgres

import (
    "context"
    "database/sql"
    _ "database/sql"
    "errors"
    "fmt"
    "github.com/google/uuid"
    "github.com/lib/pq"
    "github.com/pedrecho/vkr-auth/internal/dto"
    "time"
)

func (p *Postgres) CreateUser(ctx context.Context, email, password string,
status dto.UserStatus) (uuid.UUID, error) {
    id := uuid.New()
    now := time.Now()

    const query = `
        INSERT INTO users (id, email, password, status, created_at,
updated_at)
        VALUES ($1, $2, $3, $4, $5, $6)
    `

    _, err := p.db.ExecContext(ctx, query, id, email, password, status, now,
now)
    if err != nil {
        if isUniqueViolation(err) {
            return uuid.Nil, ErrDuplicateEmail
        }
        return uuid.Nil, fmt.Errorf("create user: %w", err)
    }

    return id, nil
}

func (p *Postgres) GetUserByEmail(ctx context.Context, email string)
(*dto.User, error) {
    const query = `
        SELECT id, email, password, status, created_at, updated_at
        FROM users
        WHERE email = $1
    `

    row := p.db.QueryRowContext(ctx, query, email)

    var user dto.User
    err := row.Scan(
        &user.ID,
        &user.Email,
        &user.Password,
        &user.Status,
        &user.CreatedAt,
        &user.UpdatedAt,
    )

    if err != nil {
        if errors.Is(err, sql.ErrNoRows) {
            return nil, ErrUserNotFound
        }
        return nil, fmt.Errorf("get user by email: %w", err)
    }

    return &user, nil
}

func isUniqueViolation(err error) bool {
    var pqErr *pq.Error
    if errors.As(err, &pqErr) {
        return pqErr.Code == "23505" // unique_violation
    }
    return false
}
```

## internal/repository/redis/errors.go

```
package redis

import (
    "errors"
)

var ErrKeyNotFound = errors.New("key not found")
```

## internal/repository/redis/redis.go

```
package redis

import (
    "context"
    "errors"
```

```go
	"fmt"
	"time"

	"github.com/pedrecho/vkr-pkg/cache"
	"github.com/redis/go-redis/v9"
)

type Redis struct {
	client *redis.Client
}

func New(cfg cache.RedisConfig) (*Redis, error) {
	client, err := cache.RedisConnect(cfg)
	if err != nil {
		return nil, fmt.Errorf("connect redis: %w", err)
	}

	return &Redis{client: client}, nil
}

func (r *Redis) Set(ctx context.Context, key, value string, ttl
time.Duration) error {
	return r.client.Set(ctx, key, value, ttl).Err()
}

func (r *Redis) Get(ctx context.Context, key string) (string, error) {
	val, err := r.client.Get(ctx, key).Result()
	if err != nil {
		if errors.Is(err, redis.Nil) {
			return "", ErrKeyNotFound
		}
		return "", fmt.Errorf("get key from redis: %w", err)
	}

	return val, nil
}

func (r *Redis) Delete(ctx context.Context, key string) error {
	n, err := r.client.Del(ctx, key).Result()
	if err != nil {
		return fmt.Errorf("delete redis key: %w", err)
	}
	if n == 0 {
		return ErrKeyNotFound
	}
	return nil
}
```

## internal/repository/user/client.go

```go
package user

import (
	"context"
	"fmt"
	"time"

	"google.golang.org/grpc"
	"google.golang.org/grpc/connectivity"
	"google.golang.org/grpc/credentials/insecure"

	"github.com/pedrecho/vkr-auth/internal/config"
	pb "github.com/pedrecho/vkr-pkg/pb/user"
```

```
)

const (
    dialTimeout    = 5 * time.Second
    requestTimeout = 5 * time.Second
    maxMsgSize     = 1 << 20 // 1 MB
)

type Client struct {
    cfg  config.UserServiceConfig
    conn *grpc.ClientConn
    svc  pb.UserServiceClient
}

// New создаёт и возвращает готовый к работе gRPC-клиент UserService
func New(cfg config.UserServiceConfig) (*Client, error) {
    ctx, cancel := context.WithTimeout(context.Background(), dialTimeout)
    defer cancel()

    conn, err := grpc.Dial(
       cfg.Host,
       grpc.WithTransportCredentials(insecure.NewCredentials()),
// небезопасный канал, как в вашем примере
       grpc.WithDefaultCallOptions(grpc.MaxCallRecvMsgSize(maxMsgSize)),
// по необходимости
       grpc.WithDefaultCallOptions(grpc.MaxCallSendMsgSize(maxMsgSize)),
// по необходимости
       grpc.WithConnectParams(grpc.ConnectParams{MinConnectTimeout:
dialTimeout}), // блокировать до готовности
    )
    if err != nil {
       return nil, fmt.Errorf("user service dial: %w", err)
    }

    // Эмулируем старый WithBlock — ждём, пока канал не перейдёт в READY
    if !conn.WaitForStateChange(ctx, connectivity.Idle) && conn.GetState() !=
connectivity.Ready {
       conn.Close()
       return nil, fmt.Errorf("user service dial: connection not ready")
    }

    return &Client{
       cfg:  cfg,
       conn: conn,
       svc:  pb.NewUserServiceClient(conn),
    }, nil
}

// Close закрывает соединение
func (c *Client) Close() error {
    return c.conn.Close()
}

// CreateUser вызывает rpc CreateUser и возвращает ошибку, если что-то пошло
не так
func (c *Client) CreateUser(ctx context.Context, userID string) error {
    // ограничиваем время выполнения RPC
    ctx, cancel := context.WithTimeout(ctx, requestTimeout)
    defer cancel()

    _, err := c.svc.CreateUser(ctx, &pb.CreateUserRequest{
       UserId: userID,
    })
    if err != nil {
```

```go
        return fmt.Errorf("create user: %w", err)
    }
    return nil
}
```

internal/service/token/errors.go

```go
package token

import (
    "errors"
    "fmt"
)

var (
    ErrInvalidAccessToken = fmt.Errorf("invalid access token")
    ErrTokenExpired       = fmt.Errorf("access token is expired")

    ErrRefreshTokenNotFound  = errors.New("refresh token not found")
    ErrRefreshTokenCorrupted = errors.New("refresh token data corrupted")
)
```

internal/service/token/generate.go

```go
package token

import (
    "context"
    "fmt"
    "github.com/golang-jwt/jwt/v5"
    "github.com/google/uuid"
    "github.com/pedrecho/vkr-auth/internal/dto"
    "time"
)

func (s *Service) GenerateTokens(
    ctx context.Context,
    userID, email, fingerprint, refreshPrefix string,
    accessTTL, refreshTTL time.Duration,
) (*dto.TokenPair, error) {

    now := time.Now()

    claims := &dto.AccessClaims{
        RegisteredClaims: jwt.RegisteredClaims{
            Subject:   userID,
            ExpiresAt: jwt.NewNumericDate(now.Add(accessTTL)),
            IssuedAt:  jwt.NewNumericDate(now),
        },
    }

    accessToken, err := jwt.NewWithClaims(jwt.SigningMethodHS256, claims).
        SignedString([]byte(s.cfg.SecretKey))
    if err != nil {
        return nil, fmt.Errorf("sign access token: %w", err)
    }

    refreshToken := uuid.NewString()
    if err := s.StoreRefreshToken(
        ctx, refreshPrefix, refreshToken, userID, email, fingerprint,
refreshTTL,
    ); err != nil {
```

```go
		return nil, fmt.Errorf("store refresh token: %w", err)
	}

	return &dto.TokenPair{
		AccessToken:  accessToken,
		RefreshToken: refreshToken,
	}, nil
}
```

internal/service/token/refresh-token.go

```go
package token

import (
	"context"
	"encoding/json"
	"errors"
	"fmt"
	"github.com/pedrecho/vkr-auth/internal/dto"
	"github.com/redis/go-redis/v9"
	"time"
)

func (s *Service) StoreRefreshToken(
	ctx context.Context,
	refreshPrefix, refreshToken, userID, email, fingerprint string,
	ttl time.Duration,
) error {
	key := fmt.Sprintf("%s:%s", refreshPrefix, refreshToken)

	data := dto.RefreshTokenData{
		UserID:      userID,
		Email:       email,
		Fingerprint: fingerprint,
	}

	encoded, err := json.Marshal(data)
	if err != nil {
		return fmt.Errorf("marshal refresh token data: %w", err)
	}

	return s.cache.Set(ctx, key, string(encoded), ttl)
}

func (s *Service) GetRefreshToken(
	ctx context.Context,
	refreshPrefix, refreshToken string,
) (*dto.RefreshTokenData, error) {

	key := fmt.Sprintf("%s:%s", refreshPrefix, refreshToken)

	raw, err := s.cache.Get(ctx, key)
	if err != nil {
		// redis.Nil = ключ не найден
		if errors.Is(err, redis.Nil) {
			return nil, ErrRefreshTokenNotFound
		}
		return nil, fmt.Errorf("get refresh token: %w", err)
	}

	var data dto.RefreshTokenData
	if err := json.Unmarshal([]byte(raw), &data); err != nil {
		// Данные повреждены (невалидный JSON, битый тип и т.п.)
```

```go
        return nil, fmt.Errorf("%w: %v", ErrRefreshTokenCorrupted, err)
    }

    return &data, nil
}

func (s *Service) DeleteRefreshToken(
    ctx context.Context,
    refreshPrefix, refreshToken string,
) error {
    key := fmt.Sprintf("%s:%s", refreshPrefix, refreshToken)

    if err := s.cache.Delete(ctx, key); err != nil {
        return fmt.Errorf("delete refresh token from cache: %w", err)
    }

    return nil
}
```

## internal/service/token/service.go

```go
package token

import (
    "context"
    "github.com/pedrecho/vkr-auth/internal/config"
    "time"
)

type Cache interface {
    Set(ctx context.Context, key, value string, ttl time.Duration) error
    Get(ctx context.Context, key string) (string, error)
    Delete(ctx context.Context, key string) error
}

type Service struct {
    cache Cache
    cfg   config.TokenConfig
}

func New(cfg config.TokenConfig, cache Cache) *Service {
    return &Service{
        cfg:   cfg,
        cache: cache,
    }
}
```

## internal/service/token/validate.go

```go
package token

import (
    "context"
    "errors"
    "github.com/golang-jwt/jwt/v5"
    "github.com/pedrecho/vkr-auth/internal/dto"
)

func (s *Service) ValidateAccessToken(
    ctx context.Context,
    tokenStr string,
) (*dto.AccessClaims, error) {
    claims := &dto.AccessClaims{}

    token, err := jwt.ParseWithClaims(tokenStr, claims, func(token
*jwt.Token) (interface{}, error) {
        if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
            return nil, ErrInvalidAccessToken
        }
        return []byte(s.cfg.SecretKey), nil
    })

    //todo other error cases?
    if err != nil || !token.Valid {
        if errors.Is(err, jwt.ErrTokenExpired) {
            return nil, ErrTokenExpired
        }
        return nil, ErrInvalidAccessToken
    }

    return claims, nil
}
```

## internal/service/user/cache.go

```go
package user

import (
    "context"
    "fmt"
)

func (s *Service) StoreVerificationCode(ctx context.Context, email, code
string) error {
    key := fmt.Sprintf("%s:%s", VerifyPrefix, email)
    return s.cache.Set(ctx, key, code, VerificationTTL)
}

func (s *Service) GetVerificationCode(ctx context.Context, email string)
(string, error) {
    key := fmt.Sprintf("%s:%s", VerifyPrefix, email)
    return s.cache.Get(ctx, key)
}
```

## internal/service/user/errors.go

```go
package user

import (
    "errors"
    "fmt"
)

var (
    ErrUserAlreadyExists         = errors.New("user already exists")
    ErrInvalidEmail              = errors.New("invalid email")
    ErrInvalidPassword           = errors.New("invalid password")
    ErrUserNotFound              = errors.New("user not found")
    ErrUserNotPending            = errors.New("user is not pending")
    ErrVerificationCodeNotFound = errors.New("verification code not found")
    ErrInvalidVerificationCode  = errors.New("invalid verification code")
    ErrInvalidFingerprint        = errors.New("invalid fingerprint")
    ErrWrongPassword             = errors.New("wrong password")
    // token
    ErrInvalidAccessToken = fmt.Errorf("invalid access token")
    ErrTokenExpired       = fmt.Errorf("access token is expired")

    ErrRefreshTokenNotFound  = errors.New("refresh token not found")
    ErrRefreshTokenCorrupted = errors.New("refresh token corrupted")
    ErrFingerprintMismatch   = errors.New("fingerprint mismatch")
)
```

## internal/service/user/login.go

```
package user

import (
    "context"
    "errors"
    "fmt"
    "github.com/pedrecho/vkr-auth/internal/dto"
    "github.com/pedrecho/vkr-auth/internal/repository/postgres"
)

func (s *Service) Login(ctx context.Context, email, password, fingerprint
string) (*dto.TokenPair, error) {
    user, err := s.storage.GetUserByEmail(ctx, email)
    if err != nil {
       if errors.Is(err, postgres.ErrUserNotFound) {
          return nil, fmt.Errorf("get user by email: %w %w", err,
ErrUserNotFound)
       }
       return nil, fmt.Errorf("get user by email: %w", err)
    }

    if err = s.CheckPassword(password, user.Password); err != nil {
       return nil, fmt.Errorf("check password: %w %w", err, ErrWrongPassword)
    }

    tokenPair, err := s.token.GenerateTokens(ctx, user.ID.String(), email,
fingerprint, RefreshPrefix, AccessTokenTTL, RefreshTokenTTL)
    if err != nil {
       return nil, fmt.Errorf("generate tokens: %w", err)
    }

    return tokenPair, nil
}
```

## internal/service/user/password.go

```
package user

import (
    "fmt"
    "golang.org/x/crypto/bcrypt"
)

// TODO improve
func (s *Service) HashPassword(password string) ([]byte, error) {
    hashed, err := bcrypt.GenerateFromPassword([]byte(password),
bcrypt.DefaultCost)
    if err != nil {
       return nil, fmt.Errorf("generate from password: %w", err)
    }
    return hashed, nil
}

func (s *Service) CheckPassword(password, hashedPassword string) error {
    return bcrypt.CompareHashAndPassword([]byte(hashedPassword),
[]byte(password))
}
```

internal/service/user/refresh.go

```
package user

import (
    "context"
    "errors"
    "fmt"
    "github.com/pedrecho/vkr-auth/internal/dto"
    "github.com/pedrecho/vkr-auth/internal/service/token"
)

func (s *Service) RefreshTokens(
    ctx context.Context,
    refreshToken string,
    fingerprint string,
) (*dto.TokenPair, error) {

    rtData, err := s.token.GetRefreshToken(ctx, RefreshPrefix, refreshToken)
    if err != nil {
       switch {
       case errors.Is(err, token.ErrRefreshTokenNotFound):
          return nil, ErrRefreshTokenNotFound
       case errors.Is(err, token.ErrRefreshTokenCorrupted):
          return nil, ErrRefreshTokenCorrupted
       default:
          return nil, fmt.Errorf("get refresh token: %w", err)
       }
    }

    if rtData.Fingerprint != fingerprint {
       return nil, ErrFingerprintMismatch
    }

    if err := s.token.DeleteRefreshToken(ctx, RefreshPrefix, refreshToken);
err != nil {
       s.log.Warnf("delete refresh token: %v", err)
    }

    newPair, err := s.token.GenerateTokens(
       ctx,
       rtData.UserID,
       rtData.Email,
       fingerprint,
       RefreshPrefix,
       AccessTokenTTL,
       RefreshTokenTTL,
    )
    if err != nil {
       return nil, fmt.Errorf("generate tokens: %w", err)
    }

    return newPair, nil
}
```

internal/service/user/registration.go

```go
package user

import (
    "context"
    "errors"
    "fmt"
    "github.com/pedrecho/vkr-auth/internal/dto"
    "github.com/pedrecho/vkr-auth/internal/repository/postgres"
    "math/rand"
    "regexp"
    "strconv"
)

var emailRegex = regexp.MustCompile(`^[^@\s]+@[^@\s]+\.[^@\s]+$`)
var passwordRegex = regexp.MustCompile(`^[\p{L}\p{N}\p{P}\p{S}]{8,}$`) // 
поддерживает латиницу, кириллицу, цифры и знаки

func (s *Service) CreateUser(ctx context.Context, email, password string,
status dto.UserStatus) (string, error) {
    if !emailRegex.MatchString(email) {
       return "", ErrInvalidEmail
    }

    if len(password) < MinPasswordLength ||
!passwordRegex.MatchString(password) {
       return "", ErrInvalidPassword
    }

    hashed, err := s.HashPassword(password)
    if err != nil {
       return "", fmt.Errorf("hash password: %w", err)
    }

    id, err := s.storage.CreateUser(ctx, email, string(hashed), status)
    if err != nil {
       if errors.Is(err, postgres.ErrDuplicateEmail) {
          return "", ErrUserAlreadyExists
       }
       return "", err
    }

    code := generateVerificationCode()
    if err := s.StoreVerificationCode(ctx, email, strconv.Itoa(int(code)));
err != nil {
       return "", fmt.Errorf("store verification code: %w", err)
    }

    if err := s.publisher.PublishEmailVerification(ctx, email, code); err !=
nil {
       return "", fmt.Errorf("publish email verification: %w", err)
    }

    return id.String(), nil
}

func (s *Service) ResendVerificationCode(ctx context.Context, email string)
error {
    user, err := s.storage.GetUserByEmail(ctx, email)
    if err != nil {
       if errors.Is(err, postgres.ErrUserNotFound) {
          return ErrUserNotFound
```

```go
		}
		return fmt.Errorf("get user: %w", err)
	}

	if user.Status != dto.UserStatusPending {
		return ErrUserNotPending
	}

	code := generateVerificationCode()

	if err := s.StoreVerificationCode(ctx, email, strconv.Itoa(int(code)));
err != nil {
		return fmt.Errorf("store verification code: %w", err)
	}

	if err := s.publisher.PublishEmailVerification(ctx, email, code); err !=
nil {
		return fmt.Errorf("publish email verification: %w", err)
	}

	return nil
}

func generateVerificationCode() int32 {
	return int32(100000 + rand.Intn(900000))
}
```

internal/service/user/service.go

```go
package user

import (
	"context"
	"github.com/google/uuid"
	"github.com/pedrecho/vkr-auth/internal/dto"
	"github.com/pedrecho/vkr-pkg/logger"
	"time"
)

const (
	VerifyPrefix  = "verify"
	RefreshPrefix = "refresh"
)

const (
	MinPasswordLength = 8
)

const (
	VerificationTTL = 24 * time.Hour
	RefreshTokenTTL = 7 * 24 * time.Hour
	AccessTokenTTL  = 15 * time.Minute
)

type Database interface {
	CreateUser(ctx context.Context, email, password string, status
dto.UserStatus) (uuid.UUID, error)
	GetUserByEmail(ctx context.Context, email string) (*dto.User, error)
	UpdateUserStatus(ctx context.Context, id uuid.UUID, status
dto.UserStatus) error
}

type Cache interface {
    Set(ctx context.Context, key, value string, ttl time.Duration) error
    Get(ctx context.Context, key string) (string, error)
    Delete(ctx context.Context, key string) error
}

type Publisher interface {
    PublishEmailVerification(ctx context.Context, email string, code int32)
error
}

type TokenService interface {
    GenerateTokens(ctx context.Context, userID, email, fingerprint,
refreshPrefix string, accessTTL, refreshTTL time.Duration) (*dto.TokenPair,
error)
    ValidateAccessToken(ctx context.Context, tokenStr string)
(*dto.AccessClaims, error)
    GetRefreshToken(ctx context.Context, refreshPrefix, refreshToken string)
(*dto.RefreshTokenData, error)
    DeleteRefreshToken(ctx context.Context, refreshPrefix, refreshToken
string) error
}

type Client interface {
    CreateUser(ctx context.Context, userID string) error
}

type Service struct {
    log       logger.Logger
    storage   Database
    cache     Cache
    publisher Publisher
    token     TokenService
    client    Client
}

func New(log logger.Logger, storage Database, cache Cache, publisher
Publisher, tokenService TokenService, client Client) *Service {
    return &Service{
       storage:   storage,
       cache:     cache,
       publisher: publisher,
       token:     tokenService,
       client:    client,
    }
}
```

## internal/service/user/validate.go

```
package user

import (
    "context"
    "errors"
    "fmt"
    "github.com/pedrecho/vkr-auth/internal/service/token"
)

func (s *Service) ValidateAccessToken(
    ctx context.Context,
    tokenStr string,
) error {
    _, err := s.token.ValidateAccessToken(ctx, tokenStr)
    switch {
```

```
    case errors.Is(err, token.ErrInvalidAccessToken):
        return fmt.Errorf("validate access token: %w %w", err,
ErrInvalidAccessToken)
    case errors.Is(err, token.ErrTokenExpired):
        return fmt.Errorf("validate access token: %w %w", err,
ErrTokenExpired)
    case err != nil:
        return fmt.Errorf("validate access token: %w", err)
    }

    return nil
}
```

## internal/service/user/verify.go

```
package user

import (
    "context"
    "errors"
    "fmt"
    "github.com/pedrecho/vkr-auth/internal/dto"
    "github.com/pedrecho/vkr-auth/internal/repository/postgres"
    "github.com/pedrecho/vkr-auth/internal/repository/redis"
)

func (s *Service) VerifyUser(ctx context.Context, email, code, fingerprint
string) (*dto.TokenPair, error) {
    if fingerprint == "" {
        return nil, ErrInvalidFingerprint
    }

    user, err := s.storage.GetUserByEmail(ctx, email)
    if err != nil {
        if errors.Is(err, postgres.ErrUserNotFound) {
            return nil, ErrUserNotFound
        }
        return nil, fmt.Errorf("get user: %w", err)
    }

    if user.Status != dto.UserStatusPending {
        return nil, ErrUserNotPending
    }

    storedCode, err := s.GetVerificationCode(ctx, email)
    if err != nil {
        if errors.Is(err, redis.ErrKeyNotFound) {
            return nil, ErrVerificationCodeNotFound
        }
        return nil, fmt.Errorf("get code: %w", err)
    }

    if storedCode != code {
        return nil, ErrInvalidVerificationCode
    }

    //if err := s.cache.Delete(ctx, fmt.Sprintf("%s:%s", VerifyPrefix,
email)); err != nil {
    // return nil, fmt.Errorf("delete code: %w", err)
    //}

    if err := s.storage.UpdateUserStatus(ctx, user.ID,
dto.UserStatusConfirmed); err != nil {
        return nil, fmt.Errorf("update user status: %w", err)
    }

    if err := s.client.CreateUser(ctx, user.ID.String()); err != nil {
        return nil, fmt.Errorf("create user in other services: %w", err)
    }

    tokens, err := s.token.GenerateTokens(ctx, user.ID.String(), email,
fingerprint, RefreshPrefix, AccessTokenTTL, RefreshTokenTTL)
    if err != nil {
        return nil, fmt.Errorf("generate tokens: %w", err)
    }

    return tokens, nil
}
```

### migrations/20250322181700_users.down.sql

```
DROP TABLE IF EXISTS users;

DROP TYPE IF EXISTS user_status;
```

### migrations/20250322181700_users.up.sql

```
DROP TABLE IF EXISTS users;

DROP TYPE IF EXISTS user_status;
```

## Репозиторий github.com/pedrecho/vkr-broker

### .github/workflows/publish-broker-app.yaml

```
name: Build & Push broker-app to GHCR

on:
  push:
    branches: [master]
  workflow_dispatch:

jobs:
  docker:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Log in to GHCR
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ secrets.GHCR_USERNAME }}
          password: ${{ secrets.GHCR_TOKEN }}

      - name: Build and Push Docker Image
        uses: docker/build-push-action@v5
        with:
          context: .
          file: docker/broker-app/Dockerfile
          push: true
          tags: ghcr.io/${{ secrets.GHCR_USERNAME }}/broker-app:latest
          build-args: |
            GITHUB_TOKEN=${{ secrets.GOPRIVATE_PAT }}
```

## .github/workflows/publish-broker-chart.yaml

```
name: Publish broker Helm Chart

on:
  push:
    tags:
      - 'chart-*'

jobs:
  helm:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Helm
        uses: azure/setup-helm@v3

      - name: Log in to GHCR
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ secrets.GHCR_USERNAME }}
          password: ${{ secrets.GHCR_TOKEN }}

      - name: Extract Chart Version
        id: chart
        run: echo "VERSION=$(grep '^version:' broker-chart/Chart.yaml | awk
'{print $2}')" >> $GITHUB_OUTPUT

      - name: Package Helm Chart
        run: helm package broker-chart

      - name: Push Helm Chart to GHCR
        run: helm push broker-chart-${{ steps.chart.outputs.VERSION }}.tgz
oci://ghcr.io/${{ secrets.GHCR_USERNAME }}
```

## Makefile

```
# ==== Docker image ====
IMAGE_NAME := ghcr.io/pedrecho/broker-app
DOCKER_TAG := latest

docker-build:
	docker build -t $(IMAGE_NAME):$(DOCKER_TAG) .

docker-push:
	docker push $(IMAGE_NAME):$(DOCKER_TAG)


# ==== Helm chart ====
CHART_DIR := broker-chart
CHART_NAME := $(CHART_DIR)
CHART_VERSION := $(shell grep "^version:" $(CHART_DIR)/Chart.yaml | awk
"{print $$2}")
RELEASE_NAME := broker

helm-tag:
	git tag chart-$(CHART_VERSION)
	@echo "Created local tag: chart-$(CHART_VERSION)"
	@echo "To push it: git push origin chart-$(CHART_VERSION)"

helm-install:
	helm upgrade --install $(RELEASE_NAME) $(CHART_NAME) \
	   -f $(CHART_DIR)/values.yaml
	   # -f $(CHART_DIR)/values.secret.yaml  # ← если появится secrets,
раскомментируй

helm-uninstall:
	helm uninstall $(RELEASE_NAME) || true

helm-clean:
	kubectl delete all -l app=$(RELEASE_NAME) || true
```

## broker-chart/Chart.yaml

```
apiVersion: v2
name: broker-chart
description: A Helm chart for deploying broker-app with NATS
version: 0.1.14
```

## broker-chart/files/config.yaml

```
logger:
  level: debug

nats:
  connection:
    host: {{ .Values.nats.host }}
    port: {{ .Values.nats.port }}
    ssl: {{ .Values.nats.ssl }}

  streams:
    {{ toYaml .Values.nats.streams | nindent 4 }}
```

## broker-chart/templates/broker-init-job.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: broker-init
  annotations:
    "helm.sh/hook": post-install,post-upgrade
    "helm.sh/hook-delete-policy": before-hook-creation,hook-succeeded
spec:
  template:
    metadata:
      name: broker-init
    spec:
      serviceAccountName: {{ .Values.serviceAccount.name }}
      restartPolicy: Never
      initContainers:
        - name: wait-for-nats
          image: busybox:1.36
          command:
            - sh
            - -c
            - >
              until nc -z {{ .Values.nats.host }} {{ .Values.nats.port }};
              do echo waiting for nats...;
              sleep 1;
              done
      containers:
        - name: broker-app
          image: {{ .Values.brokerApp.image }}
          imagePullPolicy: {{ .Values.brokerApp.imagePullPolicy }}
          command: ["./broker-app"]
          args: ["--config", "{{ .Values.brokerApp.configPath }}", "--up"]
          volumeMounts:
            - name: config-volume
              mountPath: {{ .Values.brokerApp.configPath }}
              subPath: config.yaml
      volumes:
        - name: config-volume
          configMap:
            name: broker-app-config
```

## broker-chart/templates/configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: broker-app-config
  labels:
    app: broker-app

data:
  config.yaml: |-
{{ tpl (.Files.Get "files/config.yaml") . | indent 4 }}
```

## broker-chart/templates/nats-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nats
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nats
  template:
    metadata:
      labels:
        app: nats
    spec:
      containers:
        - name: nats
          image: {{ .Values.nats.image }}
          args:
            - "-js"
            - "--store_dir=/data/jetstream"
          ports:
            - containerPort: {{ .Values.nats.port }}
          volumeMounts:
            - name: js-store
              mountPath: /data/jetstream
      volumes:
        - name: js-store
          emptyDir: {}
```

## broker-chart/templates/nats-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nats
spec:
  selector:
    app: nats
  ports:
    - port: {{ .Values.nats.port }}
      targetPort: {{ .Values.nats.port }}
  type: ClusterIP
```

## broker-chart/values.yaml

```
brokerApp:
  image: ghcr.io/pedrecho/broker-app:latest
  imagePullPolicy: Always
  replicaCount: 1
  containerPort: 8080
  configPath: /app/config.yaml

nats:
  image: nats:2.10.7
  host: "nats"
  port: 4222
  ssl: false

  streams:
    - name: "NOTIFICATIONS"
      subjects:
      - "notifications.email.verification"

serviceAccount:
  name: vkr-ghcr-access
```

## cmd/main.go

```
package main

import (
    "flag"
    "fmt"
    "github.com/pedrecho/vkr-broker/internal/app"
)

var (
    configPath = flag.String("config", "", "config path")
    upFlag     = flag.Bool("up", false, "run broker in up mode")
    downFlag   = flag.Bool("down", false, "run broker in down mode")
)

func main() {
    flag.Parse()

    app, err := app.New(*configPath)
    if err != nil {
       panic(fmt.Errorf("init app: %w", err))
    }

    if err = app.Run(true, false); err != nil {
       panic(err)
    }
}
```

## docker/broker-app/Dockerfile

```
# Стадия сборки
FROM golang:1.24 AS builder

ARG GITHUB_TOKEN
ENV GOPRIVATE=github.com/your-user
ENV CGO_ENABLED=0

RUN git config --global url."https://${GITHUB_TOKEN}:x-oauth-
basic@github.com/".insteadOf "https://github.com/"

WORKDIR /app
COPY ../../go.mod go.sum ./
RUN go mod download

COPY ../.. .
RUN go build -o broker-app ./cmd/main.go

# Финальный образ
FROM alpine:3.19
WORKDIR /app
COPY --from=builder /app/broker-app .
RUN chmod +x ./broker-app
CMD ["./broker-app"]
```

## go.mod

```
module github.com/pedrecho/vkr-broker

go 1.24

require (
    github.com/nats-io/nats.go v1.39.1
    github.com/pedrecho/vkr-pkg v0.0.0-20250225111939-85ee6427a470
    go.uber.org/zap v1.27.0
    gopkg.in/yaml.v3 v3.0.1
)

require (
    github.com/klauspost/compress v1.17.9 // indirect
    github.com/nats-io/nkeys v0.4.9 // indirect
    github.com/nats-io/nuid v1.0.1 // indirect
    go.uber.org/multierr v1.10.0 // indirect
    golang.org/x/crypto v0.31.0 // indirect
    golang.org/x/sys v0.28.0 // indirect
    gopkg.in/natefinch/lumberjack.v2 v2.2.1 // indirect
)
```

## internal/app/app.go

```go
package app

import (
	"fmt"
	"github.com/pedrecho/vkr-broker/internal/config"
	"github.com/pedrecho/vkr-broker/internal/infrastructure/nats"
	"github.com/pedrecho/vkr-broker/internal/service/broker"
	"github.com/pedrecho/vkr-pkg/zaplogger"
)

type App struct {
	cfg *config.Config
}

func New(configPath string) (*App, error) {
	cfg, err := config.Load(configPath)
	if err != nil {
		return nil, fmt.Errorf("config init: %w", err)
	}

	return &App{
		cfg: cfg,
	}, nil
}

func (a *App) Run(init bool, cleanup bool) error {
	zapsync, err := zaplogger.ReplaceZap(a.cfg.Logger)
	if err != nil {
		return fmt.Errorf("zaplogger init: %w", err)
	}
	defer zapsync()

	natsService, err := nats.New(a.cfg.Nats.Connection)
	if err != nil {
		return fmt.Errorf("init nats: %w", err)
	}

	brokerService := broker.New(a.cfg.Nats, natsService)

	if cleanup {
		if err = brokerService.CleanupStreams(); err != nil {
			return fmt.Errorf("cleanup streams: %w", err)
		}
	}

	if init {
		if err = brokerService.InitStreams(); err != nil {
			return fmt.Errorf("init streams: %w", err)
		}
	}

	return nil
}
```

## internal/config/config.go

```go
package config

import (
	"fmt"
```

```
    "github.com/pedrecho/vkr-pkg/messaging"
    "github.com/pedrecho/vkr-pkg/zaplogger"
    "gopkg.in/yaml.v3"
    "os"
)

type Config struct {
    Logger zaplogger.Config `yaml:"logger"`
    Nats   NatsConfig       `yaml:"nats"`
}

func Load(filename string) (*Config, error) {

    //TODO remove
    data, err := os.ReadFile(filename)
    if err != nil {
       fmt.Println("Ошибка чтения файла:", err)
       os.Exit(1)
    }
    fmt.Println(string(data))

    file, err := os.ReadFile(filename)
    if err != nil {
       return nil, fmt.Errorf("read config file: %w", err)
    }
    cfg := Config{}
    err = yaml.Unmarshal(file, &cfg)
    if err != nil {
       return nil, fmt.Errorf("unmarshal config file: %w", err)
    }

    return &cfg, err
}

type NatsConfig struct {
    Connection messaging.NatsConfig `yaml:"connection"`
    Streams    []StreamConfig       `yaml:"streams"`
}

type StreamConfig struct {
    Name     string   `yaml:"name"`
    Subjects []string `yaml:"subjects"`
}
```

## internal/infrastructure/nats/nats.go

```
package nats

import (
    "fmt"
    "github.com/nats-io/nats.go"
    "github.com/pedrecho/vkr-pkg/messaging"
)

type Service struct {
    cfg  messaging.NatsConfig
    conn *nats.Conn
    js   nats.JetStreamContext
}

func New(cfg messaging.NatsConfig) (*Service, error) {
    conn, js, err := messaging.NatsJetStreamConnect(cfg)
    if err != nil {
```

```go
        return nil, fmt.Errorf("nats jetstream connction: %w", err)
    }

    return &Service{
        cfg:  cfg,
        conn: conn,
        js:   js,
    }, nil
}
```

## internal/infrastructure/nats/streams.go

```go
package nats

import (
    "errors"
    "fmt"
    "github.com/nats-io/nats.go"
    "go.uber.org/zap"
    "log"
)

func (s *Service) AddOrUpdateStream(streamName string, subjects []string)
error {
    _, err := s.js.StreamInfo(streamName)
    if err != nil {
        if errors.Is(err, nats.ErrStreamNotFound) {
            _, err = s.js.AddStream(&nats.StreamConfig{
                Name:     streamName,
                Subjects: subjects,
            })
            if err != nil {
                return fmt.Errorf("add stream: %w", err)
            }

            zap.S().Infof("New stream created: %s", streamName)
            return nil
        }

        return fmt.Errorf("get stream info: %w", err)
    }

    _, err = s.js.UpdateStream(&nats.StreamConfig{
        Name:     streamName,
        Subjects: subjects,
    })
    if err != nil {
        return fmt.Errorf("update stream: %w", err)
    }

    log.Printf("Stream updated: %s", streamName)
    return nil
}

func (s *Service) DeleteStream(streamName string) error {
    err := s.js.DeleteStream(streamName)
    if err != nil {
        if errors.Is(err, nats.ErrStreamNotFound) {
            zap.S().Infof("Stream not found: %s", streamName)
            return nil
        }

        return fmt.Errorf("delete stream: %w", err)
    }

    zap.S().Infof("Stream deleted: %s", streamName)
    return nil
}
```

## internal/service/broker/broker.go

```go
package broker

import "github.com/pedrecho/vkr-broker/internal/config"

type StreamManager interface {
    AddOrUpdateStream(streamName string, subjects []string) error
    DeleteStream(streamName string) error
}

type Service struct {
    streamManager StreamManager
    cfg           config.NatsConfig
}

func New(cfg config.NatsConfig, streamManager StreamManager) *Service {
    return &Service{
        streamManager: streamManager,
        cfg:           cfg,
    }
}
```

## internal/service/broker/streams.go

```go
package broker

import (
    "fmt"
)

func (s *Service) InitStreams() error {
    for _, stream := range s.cfg.Streams {
        err := s.streamManager.AddOrUpdateStream(stream.Name, stream.Subjects)
        if err != nil {
            return fmt.Errorf("init stream %s: %w", stream.Name, err)
        }
    }
    return nil
}

func (s *Service) CleanupStreams() error {
    for _, stream := range s.cfg.Streams {
        err := s.streamManager.DeleteStream(stream.Name)
        if err != nil {
            return fmt.Errorf("cleanup stream %s: %w", stream.Name, err)
        }
    }
    return nil
}
```