internal/domain/game/game.go

```
package game

import (
 "github.com/gorilla/websocket"
 "time"
)

type Game struct {
 Users         []*GameUser     `json:"users" bson:"users"`
 CreatedAt     time.Time       `json:"created_at" bson:"created_at"`
 StartedAt     *time.Time      `json:"started_at,omitempty" bson:"started_at,omitempty"`
 Status        string          `json:"status" bson:"status"`
 BoardSize     int             `json:"board_size" bson:"board_size"`
 GameKey       string          `json:"game_key" bson:"game_key"` // Ñƒ½Ð¸ÐºÐ°Ð»ÑŒÐ½Ñ‹Ð¹ ÐºÐ»ÑŽÑ‡
 CurrentTurn   string          `json:"current_turn" bson:"current_turn"`
 Moves         []Move          `json:"moves" bson:"moves"`
 WhoIsNext     string          `json:"who_is_next" bson:"who_is_next"` // color
 PlayerBlack   string          `json:"player_black" bson:"player_black"`
 PlayerWhite   string          `json:"player_white" bson:"player_white"`
 PlayerBlackWS *websocket.Conn `json:"-"`
 PlayerWhiteWS *websocket.Conn `json:"-"`
 Komi          float64         `json:"komi" bson:"komi"`
}

type GameUser struct {
 ID     string          `json:"id" bson:"id"`
 Role   string          `json:"role" bson:"role"`
 Color  string          `json:"color" bson:"color"`
 Rating float64         `json:"rating" bson:"rating"`
 Score  float64         `json:"score" bson:"score"`
 WS     *websocket.Conn `json:"-"`
}

type GameCreateResponse struct {
 UniqueKey string `json:"unique_key" bson:"unique_key"`
}

type GameJoinRequest struct {
 GameKey string `json:"game_key" bson:"game_key"`
 UserID  string `json:"user_id" bson:"user_id"`
 Role    string `json:"role" bson:"role"`
}

type GameStateResponse struct {
 Move Move   `json:"move"`
 SGF  string `json:"sgf"`
}
```

internal/domain/user/user.go

```
package user

type User struct {
 ID           string
 Username     string `json:"Username"`
 PasswordHash string
 PasswordSalt string
}
```

internal/usecase/katago/katago.go

```
package katago

import (
 "context"
 "team_exe/internal/domain/game"
 katagoRPC "team_exe/microservices/proto"
)

func GenMove(ctx context.Context, moves game.Moves, katagoGRPC
katagoRPC.KatagoServiceClient) (game.Move, error) {
 movesRPC := ConvertDomainMovesToRPC(moves)

 botResponse, err := katagoGRPC.GenerateMove(ctx, &movesRPC)
 if err != nil {
  return game.Move{}, err
 }

 return game.Move{
  Coordinates: botResponse.BotMove,
  Color:       "w",
 }, nil
}

func ConvertDomainMovesToRPC(movesDomain game.Moves) katagoRPC.Moves {
 rpcMoves := make([]*katagoRPC.Move, 0)
 for _, m := range movesDomain.Moves {
  move := &katagoRPC.Move{
   Coordinates: m.Coordinates,
   Color:       m.Color,
  }
  rpcMoves = append(rpcMoves, move)
 }
 return katagoRPC.Moves{Moves: rpcMoves}
}
```

internal/adapters/redis.go

```
package adapters

import (
 "context"
 "fmt"
 "log"
 "time"

 "github.com/redis/go-redis/v9"
 "team_exe/internal/bootstrap"
)

type AdapterRedis struct {
 client *redis.Client
 cfg    *bootstrap.Config
}

func NewAdapterRedis(cfg *bootstrap.Config) *AdapterRedis {
 return &AdapterRedis{cfg: cfg}
}

func (a *AdapterRedis) Init(ctx context.Context) error {
 addr := a.cfg.RedisUrl
 password := "" // Ð•Ñ•Ð»Ð¸ ÐµÑ•Ñ‚ÑŒ Ð¿Ð°Ñ€Ð¾Ð»ÑŒ, Ñƒ Ð°Ð´Ð¼Ð¸Ð½Ð¸ÑÑ‚Ñ€Ð°Ñ‚Ð¾Ñ€Ð° ÐµÐ³Ð¾ Ð·Ð´ÐµÑ•ÑŒ
Ð¸Ð»Ð¸ Ð²Ð¾Ð·ÑŒÐ¼Ð¸Ñ‚Ðµ Ð¸Ð· cfg

 a.client = redis.NewClient(&redis.Options{
  Addr:     addr,
  Password: password,
  DB:       0,
 })

 ctxPing, cancel := context.WithTimeout(ctx, 5*time.Second)
 defer cancel()

 if err := a.client.Ping(ctxPing).Err(); err != nil {
  return fmt.Errorf("Ð¾ÑˆÐ¸Ð±Ð°Ð° Ð¿Ð¾Ð´Ð»ÑŽÑ‡ÐµÐ½Ð¸Ñ• Ð° Redis: %w", err)
 }

 log.Println("Ð£Ñ•Ð¿ÐµÑˆÐ½Ð¾ Ð¿Ð¾Ð´Ð»ÑŽÑ‡ÐµÐ½Ð¾ Ð° Redis")
 return nil
}

func (a *AdapterRedis) GetClient() *redis.Client {
 return a.client
}

func (a *AdapterRedis) Close(ctx context.Context) error {
 if a.client != nil {
  return a.client.Close()
 }
 return nil
}
```

internal/middleware/cors.go

```
package middleware

import "net/http"

func CORS(next http.Handler) http.Handler {
 return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
  header := w.Header()
  header.Add("Access-Control-Allow-Origin", "*")
  header.Add("Access-Control-Allow-Methods", "DELETE, POST, GET, OPTIONS")
  header.Add("Access-Control-Allow-Headers", "Content-Type, Authorization,
X-Requested-With")
  header.Add("Access-Control-Allow-Credentials", "true")
  if r.Method == "OPTIONS" {
   w.WriteHeader(http.StatusOK)
  }
  next.ServeHTTP(w, r)
 })
}
```

internal/random/string.go

```
package random

import (
 "math/rand"
 "strings"
 "time"
)

var randSrc rand.Source

func init() {
 randSrc = rand.NewSource(time.Now().UnixNano())
}

func RandString(n int) string {
 letterBytes := "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
 letterIdxBits := 6                    // 6 bits to represent a letter index
 letterIdxMask := 1<<letterIdxBits - 1 // All 1-bits, as many as letterIdxBits
 letterIdxMax := 63 / letterIdxBits    // # of letter indices fitting in 63 bits
 sb := strings.Builder{}
 sb.Grow(n)
 // A src.Int63() generates 63 random bits, enough for letterIdxMax characters!
 for i, cache, remain := n-1, randSrc.Int63(), letterIdxMax; i >= 0; {
  if remain == 0 {
   cache, remain = randSrc.Int63(), letterIdxMax
  }
  if idx := int(cache & int64(letterIdxMask)); idx < len(letterBytes) {
   sb.WriteByte(letterBytes[idx])
   i--
  }
  cache >>= letterIdxBits
  remain--
 }

 return sb.String()
}
```

internal/repository/session.go

```
package repo

import (
 "context"
 "errors"
 "log/slog"
 "time"

 "github.com/redis/go-redis/v9"
)

type RedisSessionStorage struct {
 client *redis.Client
}

func NewSessionRedisStorage(redis *redis.Client) *RedisSessionStorage {
 return &RedisSessionStorage{
  client: redis,
 }
}

func (r RedisSessionStorage) GetUserIdBySession(sessionID string) (string, bool) {
 v, err := r.client.Get(context.Background(), sessionID).Result()
 if err != nil {
  if errors.Is(err, redis.Nil) {
   return "", false
  }
  slog.Error(err.Error())
  return "", false
 }

 return v, true
}

func (r RedisSessionStorage) StoreSession(sessionID string, userID string) {
 err := r.client.Set(context.Background(), sessionID, userID, time.Hour*11).Err()
 if err != nil {
  slog.Error("ĐžÑˆÐ¸Ð±ÐºÐ° Ð·Ð°Ð¿Ð¸Ñ•Ð¸ Ñ•ÐµÑ•Ñ•Ð¸Ð¸ Ð² Redis: " + err.Error())
 }
}

func (r RedisSessionStorage) DeleteSession(sessionID string) bool {
 err := r.client.Del(context.Background(), sessionID).Err()
 if err != nil {
  slog.Error("ĐžÑˆÐ¸Ð±ÐºÐ° Ñƒ´Ð°»ÐµÐ½Ð¸Ñ• Ñ•ÐµÑ•Ñ•Ð¸Ð¸ Ð¸Ð· Redis: " + err.Error())
  return false
 }
 return true
}
```

internal/usecase/game/game.go

```
package game

import (
 "context"
 "fmt"
 "strconv"
 "strings"
 "team_exe/internal/domain/game"
 sgf "team_exe/internal/domain/sgf"
 "team_exe/internal/errors"
)

type GameStore interface {
 GenerateGameKey(ctx context.Context) string
 PutGameToMongoDatabase(ctx context.Context, gameData game.Game) bool
 AddPlayer(ctx context.Context, newUser game.GameUser, gameKey string) bool
 ConvertToUserFromJoinReq(ctx context.Context, joinRequest game.GameJoinRequest)
game.GameUser
 GetGameByGameKey(ctx context.Context, gameKey string) game.Game
 SaveSGFToRedis(key string, sgfText string) error
 LoadSGFFromRedis(key string) (string, error)
 GetActiveGameByUserId(ctx context.Context, userID string) ([]game.Game, error)
}

type GameUseCase struct {
 store GameStore
}

func NewGameUseCase(store GameStore) *GameUseCase {
 return &GameUseCase{store: store}
}

func (g *GameUseCase) CreateGame(ctx context.Context, gameData game.Game) (err error,
gameUniqueKey string) {
 gameUniqueKey = g.store.GenerateGameKey(ctx)
 gameData.GameKey = gameUniqueKey

 ok := g.store.PutGameToMongoDatabase(ctx, gameData)
 if !ok {
  return errors.ErrCreateGameFailed, ""
 }
 return nil, gameUniqueKey
}

func (g *GameUseCase) JoinGame(ctx context.Context, gameJoinData game.GameJoinRequest)
(err error) {
 newUser := g.store.ConvertToUserFromJoinReq(ctx, gameJoinData)
 ok := g.store.AddPlayer(ctx, newUser, gameJoinData.GameKey)
 if !ok {
  return errors.ErrCreateGameFailed
 }

 foundGame, err := g.GetGameByID(ctx, gameJoinData.GameKey)
 if err != nil {
  return err
 }
```

```
 minSGF := g.PrepareSgfFile(foundGame)
 sgfString := SerializeSGF(&minSGF)
 err = g.store.SaveSGFToRedis(foundGame.GameKey, sgfString)
 if err != nil {
  return err
 }

 return nil
}

func (g *GameUseCase) GetGameByID(ctx context.Context, gameUniqueKey string) (game.Game,
error) {
 gameFromDb := g.store.GetGameByGameKey(ctx, gameUniqueKey)
 if gameFromDb.GameKey == "" {
  return game.Game{}, errors.ErrGameNotFound
 }
 return g.store.GetGameByGameKey(ctx, gameUniqueKey), nil
}

func (g *GameUseCase) PrepareSgfFile(gameData game.Game) sgf.SGF {
 minSGF := sgf.SGF{
  Root: &sgf.GameTree{
   Nodes: []sgf.Node{
    {
     Properties: map[string][]string{
      "FF": {"4"},
      "GM": {"1"},
      "SZ": {strconv.Itoa(gameData.BoardSize)},
      "PB": {gameData.PlayerBlack},
      "PW": {gameData.PlayerWhite},
      "DT": {gameData.CreatedAt.String()},
      "RE": {""},
      "KM": {strconv.FormatFloat(gameData.Komi, 'f', 1, 64)},
      "RU": {"Chinese"},
      "C":  {"Game 1 x 1"},
     },
    },
   },
  },
 }
 return minSGF
}

func AddMovesToSgf(tree *sgf.GameTree, moves []game.Move) {
 for _, move := range moves {
  node := sgf.Node{
   Properties: map[string][]string{
    move.Color: {move.Coordinates},
   },
  }
  tree.Nodes = append(tree.Nodes, node)
 }
}

func (g *GameUseCase) GetSgfStringByGameKey(key string) (string, error) {
 return g.store.LoadSGFFromRedis(key)
}

func SerializeSGF(s *sgf.SGF) string {
```

```
 var builder strings.Builder
 builder.WriteString("(")
 serializeGameTree(&builder, s.Root)
 builder.WriteString(")")
 return builder.String()
}

func serializeGameTree(builder *strings.Builder, tree *sgf.GameTree) {
 for _, node := range tree.Nodes {
  builder.WriteString(";")

  // Ñ„Ð¸ÐºÐ¸Ñ€Ð¾Ð²Ð°Ð½Ð½Ñ‹Ðµ Ð¿Ð¾Ñ€ÑÐ´Ð¾Ðº Ñ•Ð²Ð¾Ð¹Ñ•Ñ‚Ð² SGF
  orderedKeys := []string{"FF", "GM", "SZ", "PB", "PW", "DT", "RE", "KM", "RU", "C",
"B", "W"}
  used := make(map[string]bool)
  for _, key := range orderedKeys {
   if values, ok := node.Properties[key]; ok {
    used[key] = true
    for _, v := range values {
     builder.WriteString(fmt.Sprintf("%s[%s]", key, v))
    }
   }
  }

  for key, values := range node.Properties {
   if !used[key] {
    for _, v := range values {
     builder.WriteString(fmt.Sprintf("%s[%s]", key, v))
    }
   }
  }
 }

 for _, child := range tree.Children {
  builder.WriteString("(")
  serializeGameTree(builder, child)
  builder.WriteString(")")
 }
}

func (g *GameUseCase) AddMoveToGameSgf(key string, move game.Move) (string, error) {
 sgfString, err := g.GetSgfStringByGameKey(key)
 if err != nil {
  return "", err
 }
 newSgfString := AppendMoveToSgf(sgfString, move)
 err = g.store.SaveSGFToRedis(key, newSgfString)
 if err != nil {
  return "", err
 }
 return newSgfString, nil
}

func AppendMoveToSgf(sgfText string, move game.Move) string {
 if strings.HasSuffix(sgfText, ")") {
  sgfText = sgfText[:len(sgfText)-1]
 }
 return sgfText + fmt.Sprintf(";%s[%s])", move.Color, move.Coordinates)
}
```

```
func (g *GameUseCase) IsUserInGameByGameId(ctx context.Context, userID string, gameKey
string) bool {
 play := g.store.GetGameByGameKey(ctx, gameKey)
 if play.PlayerWhite == userID || play.PlayerBlack == userID {
  return true
 }
 return false
}

func (g *GameUseCase) HasUserActiveGamesByUserId(ctx context.Context, userID string)
(bool, error) {
 plays, err := g.store.GetActiveGameByUserId(ctx, userID)
 if err != nil {
  return true, err
 }
 if len(plays) == 0 {
  return false, nil
 }
 return true, nil
}
```

microservices/cmd/katago/main.go

```
package main

import (
 "fmt"
 "go.uber.org/zap"
 "google.golang.org/grpc"
 "log"
 "net"
 "team_exe/internal/bootstrap"
 katago "team_exe/microservices/proto"
 "team_exe/microservices/repository"
 "team_exe/microservices/usecase"
)

func main() {
 logger := NewLogger()
 cfg, err := bootstrap.Setup(".env")
 if err != nil {
  logger.Error("Failed to setup configuration", zap.Error(err))
  return
 }

 lis, err := net.Listen("tcp", ":8082")
 if err != nil {
  log.Fatalln("cant listen port", err)
 }

 server := grpc.NewServer()
 katagoStorage := repository.NewKatagoRepository(cfg, logger)
 katago.RegisterKatagoServiceServer(server, usecase.NewKatagoUseCase(katagoStorage))
 fmt.Println("starting server at :8082")
 server.Serve(lis)
}

func NewLogger() *zap.SugaredLogger {
 logger, err := zap.NewProduction()
 if err != nil {
  panic("failed to initialize logger: " + err.Error())
 }

 return logger.Sugar()
}
```

microservices/proto/katago.pb.go

```
// Code generated by protoc-gen-go. DO NOT EDIT.
// versions:
//  protoc-gen-go v1.36.5
//  protoc        v3.21.12
// source: katago.proto

package katago

import (
 protoreflect "google.golang.org/protobuf/reflect/protoreflect"
 protoimpl "google.golang.org/protobuf/runtime/protoimpl"
 reflect "reflect"
 sync "sync"
 unsafe "unsafe"
)

const (
 // Verify that this generated code is sufficiently up-to-date.
 _ = protoimpl.EnforceVersion(20 - protoimpl.MinVersion)
 // Verify that runtime/protoimpl is sufficiently up-to-date.
 _ = protoimpl.EnforceVersion(protoimpl.MaxVersion - 20)
)

type BotResponse struct {
 state         protoimpl.MessageState `protogen:"open.v1"`
 BotMove       string
`protobuf:"bytes,1,opt,name=bot_move,json=botMove,proto3" json:"bot_move,omitempty"`
 Diagnostics   *Diagnostics           `protobuf:"bytes,2,opt,name=diagnostics,proto3"
json:"diagnostics,omitempty"`
 RequestId     string
`protobuf:"bytes,3,opt,name=request_id,json=requestId,proto3"
json:"request_id,omitempty"`
 unknownFields protoimpl.UnknownFields
 sizeCache     protoimpl.SizeCache
}

func (x *BotResponse) Reset() {
 *x = BotResponse{}
 mi := &file_katago_proto_msgTypes[0]
 ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
 ms.StoreMessageInfo(mi)
}

func (x *BotResponse) String() string {
 return protoimpl.X.MessageStringOf(x)
}

func (*BotResponse) ProtoMessage() {}

func (x *BotResponse) ProtoReflect() protoreflect.Message {
 mi := &file_katago_proto_msgTypes[0]
 if x != nil {
  ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
  if ms.LoadMessageInfo() == nil {
   ms.StoreMessageInfo(mi)
  }
  return ms
```

```
 }
 return mi.MessageOf(x)
}

// Deprecated: Use BotResponse.ProtoReflect.Descriptor instead.
func (*BotResponse) Descriptor() ([]byte, []int) {
 return file_katago_proto_rawDescGZIP(), []int{0}
}

func (x *BotResponse) GetBotMove() string {
 if x != nil {
  return x.BotMove
 }
 return ""
}

func (x *BotResponse) GetDiagnostics() *Diagnostics {
 if x != nil {
  return x.Diagnostics
 }
 return nil
}

func (x *BotResponse) GetRequestId() string {
 if x != nil {
  return x.RequestId
 }
 return ""
}

type Diagnostics struct {
 state         protoimpl.MessageState `protogen:"open.v1"`
 BestTen       []*MovePSV
`protobuf:"bytes,1,rep,name=best_ten,json=bestTen,proto3" json:"best_ten,omitempty"`
 BotMove       string
`protobuf:"bytes,2,opt,name=bot_move,json=botMove,proto3" json:"bot_move,omitempty"`
 Score         float64                `protobuf:"fixed64,3,opt,name=score,proto3"
json:"score,omitempty"`
 WinProb       float64
`protobuf:"fixed64,4,opt,name=win_prob,json=winProb,proto3" json:"win_prob,omitempty"`
 unknownFields protoimpl.UnknownFields
 sizeCache     protoimpl.SizeCache
}

func (x *Diagnostics) Reset() {
 *x = Diagnostics{}
 mi := &file_katago_proto_msgTypes[1]
 ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
 ms.StoreMessageInfo(mi)
}

func (x *Diagnostics) String() string {
 return protoimpl.X.MessageStringOf(x)
}

func (*Diagnostics) ProtoMessage() {}

func (x *Diagnostics) ProtoReflect() protoreflect.Message {
 mi := &file_katago_proto_msgTypes[1]
```

```
 if x != nil {
  ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
  if ms.LoadMessageInfo() == nil {
   ms.StoreMessageInfo(mi)
  }
  return ms
 }
 return mi.MessageOf(x)
}

// Deprecated: Use Diagnostics.ProtoReflect.Descriptor instead.
func (*Diagnostics) Descriptor() ([]byte, []int) {
 return file_katago_proto_rawDescGZIP(), []int{1}
}

func (x *Diagnostics) GetBestTen() []*MovePSV {
 if x != nil {
  return x.BestTen
 }
 return nil
}

func (x *Diagnostics) GetBotMove() string {
 if x != nil {
  return x.BotMove
 }
 return ""
}

func (x *Diagnostics) GetScore() float64 {
 if x != nil {
  return x.Score
 }
 return 0
}

func (x *Diagnostics) GetWinProb() float64 {
 if x != nil {
  return x.WinProb
 }
 return 0
}

type MovePSV struct {
 state         protoimpl.MessageState `protogen:"open.v1"`
 Move          string                 `protobuf:"bytes,1,opt,name=move,proto3"
json:"move,omitempty"`
 Psv           int32                  `protobuf:"varint,2,opt,name=psv,proto3"
json:"psv,omitempty"`
 unknownFields protoimpl.UnknownFields
 sizeCache     protoimpl.SizeCache
}

func (x *MovePSV) Reset() {
 *x = MovePSV{}
 mi := &file_katago_proto_msgTypes[2]
 ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
 ms.StoreMessageInfo(mi)
}
```

```
func (x *MovePSV) String() string {
 return protoimpl.X.MessageStringOf(x)
}

func (*MovePSV) ProtoMessage() {}

func (x *MovePSV) ProtoReflect() protoreflect.Message {
 mi := &file_katago_proto_msgTypes[2]
 if x != nil {
  ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
  if ms.LoadMessageInfo() == nil {
   ms.StoreMessageInfo(mi)
  }
  return ms
 }
 return mi.MessageOf(x)
}

// Deprecated: Use MovePSV.ProtoReflect.Descriptor instead.
func (*MovePSV) Descriptor() ([]byte, []int) {
 return file_katago_proto_rawDescGZIP(), []int{2}
}

func (x *MovePSV) GetMove() string {
 if x != nil {
  return x.Move
 }
 return ""
}

func (x *MovePSV) GetPsv() int32 {
 if x != nil {
  return x.Psv
 }
 return 0
}

type Move struct {
 state         protoimpl.MessageState `protogen:"open.v1"`
 Color         string                 `protobuf:"bytes,1,opt,name=color,proto3"
json:"color,omitempty"`
 Coordinates   string                 `protobuf:"bytes,2,opt,name=coordinates,proto3"
json:"coordinates,omitempty"`
 unknownFields protoimpl.UnknownFields
 sizeCache     protoimpl.SizeCache
}

func (x *Move) Reset() {
 *x = Move{}
 mi := &file_katago_proto_msgTypes[3]
 ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
 ms.StoreMessageInfo(mi)
}

func (x *Move) String() string {
 return protoimpl.X.MessageStringOf(x)
}
```

```
func (*Move) ProtoMessage() {}

func (x *Move) ProtoReflect() protoreflect.Message {
 mi := &file_katago_proto_msgTypes[3]
 if x != nil {
  ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
  if ms.LoadMessageInfo() == nil {
   ms.StoreMessageInfo(mi)
  }
  return ms
 }
 return mi.MessageOf(x)
}

// Deprecated: Use Move.ProtoReflect.Descriptor instead.
func (*Move) Descriptor() ([]byte, []int) {
 return file_katago_proto_rawDescGZIP(), []int{3}
}

func (x *Move) GetColor() string {
 if x != nil {
  return x.Color
 }
 return ""
}

func (x *Move) GetCoordinates() string {
 if x != nil {
  return x.Coordinates
 }
 return ""
}

type Moves struct {
 state         protoimpl.MessageState `protogen:"open.v1"`
 Moves         []*Move                `protobuf:"bytes,1,rep,name=moves,proto3"
json:"moves,omitempty"`
 unknownFields protoimpl.UnknownFields
 sizeCache     protoimpl.SizeCache
}

func (x *Moves) Reset() {
 *x = Moves{}
 mi := &file_katago_proto_msgTypes[4]
 ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
 ms.StoreMessageInfo(mi)
}

func (x *Moves) String() string {
 return protoimpl.X.MessageStringOf(x)
}

func (*Moves) ProtoMessage() {}

func (x *Moves) ProtoReflect() protoreflect.Message {
 mi := &file_katago_proto_msgTypes[4]
 if x != nil {
  ms := protoimpl.X.MessageStateOf(protoimpl.Pointer(x))
  if ms.LoadMessageInfo() == nil {
```

```
   ms.StoreMessageInfo(mi)
  }
  return ms
 }
 return mi.MessageOf(x)
}

// Deprecated: Use Moves.ProtoReflect.Descriptor instead.
func (*Moves) Descriptor() ([]byte, []int) {
 return file_katago_proto_rawDescGZIP(), []int{4}
}

func (x *Moves) GetMoves() []*Move {
 if x != nil {
  return x.Moves
 }
 return nil
}

var File_katago_proto protoreflect.FileDescriptor

var file_katago_proto_rawDesc = string([]byte{
 0x0a, 0x0c, 0x6b, 0x61, 0x74, 0x61, 0x67, 0x6f, 0x2e, 0x70, 0x72, 0x6f, 0x74, 0x6f,
0x12, 0x06,
 0x6b, 0x61, 0x74, 0x61, 0x67, 0x6f, 0x22, 0x7e, 0x0a, 0x0b, 0x42, 0x6f, 0x74, 0x52,
0x65, 0x73,
 0x70, 0x6f, 0x6e, 0x73, 0x65, 0x12, 0x19, 0x0a, 0x08, 0x62, 0x6f, 0x74, 0x5f, 0x6d,
0x6f, 0x76,
 0x65, 0x18, 0x01, 0x20, 0x01, 0x28, 0x09, 0x52, 0x07, 0x62, 0x6f, 0x74, 0x4d, 0x6f,
0x76, 0x65,
 0x12, 0x35, 0x0a, 0x0b, 0x64, 0x69, 0x61, 0x67, 0x6e, 0x6f, 0x73, 0x74, 0x69, 0x63,
0x73, 0x18,
 0x02, 0x20, 0x01, 0x28, 0x0b, 0x32, 0x13, 0x2e, 0x6b, 0x61, 0x74, 0x61, 0x67, 0x6f,
0x2e, 0x44,
 0x69, 0x61, 0x67, 0x6e, 0x6f, 0x73, 0x74, 0x69, 0x63, 0x73, 0x52, 0x0b, 0x64, 0x69,
0x61, 0x67,
 0x6e, 0x6f, 0x73, 0x74, 0x69, 0x63, 0x73, 0x12, 0x1d, 0x0a, 0x0a, 0x72, 0x65, 0x71,
0x75, 0x65,
 0x73, 0x74, 0x5f, 0x69, 0x64, 0x18, 0x03, 0x20, 0x01, 0x28, 0x09, 0x52, 0x09, 0x72,
0x65, 0x71,
 0x75, 0x65, 0x73, 0x74, 0x49, 0x64, 0x22, 0x85, 0x01, 0x0a, 0x0b, 0x44, 0x69, 0x61,
0x67, 0x6e,
 0x6f, 0x73, 0x74, 0x69, 0x63, 0x73, 0x12, 0x2a, 0x0a, 0x08, 0x62, 0x65, 0x73, 0x74,
0x5f, 0x74,
 0x65, 0x6e, 0x18, 0x01, 0x20, 0x03, 0x28, 0x0b, 0x32, 0x0f, 0x2e, 0x6b, 0x61, 0x74,
0x61, 0x67,
 0x6f, 0x2e, 0x4d, 0x6f, 0x76, 0x65, 0x50, 0x53, 0x56, 0x52, 0x07, 0x62, 0x65, 0x73,
0x74, 0x54,
 0x65, 0x6e, 0x12, 0x19, 0x0a, 0x08, 0x62, 0x6f, 0x74, 0x5f, 0x6d, 0x6f, 0x76, 0x65,
0x18, 0x02,
 0x20, 0x01, 0x28, 0x09, 0x52, 0x07, 0x62, 0x6f, 0x74, 0x4d, 0x6f, 0x76, 0x65, 0x12,
0x14, 0x0a,
 0x05, 0x73, 0x63, 0x6f, 0x72, 0x65, 0x18, 0x03, 0x20, 0x01, 0x28, 0x01, 0x52, 0x05,
0x73, 0x63,
 0x6f, 0x72, 0x65, 0x12, 0x19, 0x0a, 0x08, 0x77, 0x69, 0x6e, 0x5f, 0x70, 0x72, 0x6f,
0x62, 0x18,
 0x04, 0x20, 0x01, 0x28, 0x01, 0x52, 0x07, 0x77, 0x69, 0x6e, 0x50, 0x72, 0x6f, 0x62,
0x22, 0x2f,
 0x0a, 0x07, 0x4d, 0x6f, 0x76, 0x65, 0x50, 0x53, 0x56, 0x12, 0x12, 0x0a, 0x04, 0x6d,
```

```
0x6f, 0x76,
 0x65, 0x18, 0x01, 0x20, 0x01, 0x28, 0x09, 0x52, 0x04, 0x6d, 0x6f, 0x76, 0x65, 0x12,
0x10, 0x0a,
 0x03, 0x70, 0x73, 0x76, 0x18, 0x02, 0x20, 0x01, 0x28, 0x05, 0x52, 0x03, 0x70, 0x73,
0x76, 0x22,
 0x3e, 0x0a, 0x04, 0x4d, 0x6f, 0x76, 0x65, 0x12, 0x14, 0x0a, 0x05, 0x63, 0x6f, 0x6c,
0x6f, 0x72,
 0x18, 0x01, 0x20, 0x01, 0x28, 0x09, 0x52, 0x05, 0x63, 0x6f, 0x6c, 0x6f, 0x72, 0x12,
0x20, 0x0a,
 0x0b, 0x63, 0x6f, 0x6f, 0x72, 0x64, 0x69, 0x6e, 0x61, 0x74, 0x65, 0x73, 0x18, 0x02,
0x20, 0x01,
 0x28, 0x09, 0x52, 0x0b, 0x63, 0x6f, 0x6f, 0x72, 0x64, 0x69, 0x6e, 0x61, 0x74, 0x65,
0x73, 0x22,
 0x2b, 0x0a, 0x05, 0x4d, 0x6f, 0x76, 0x65, 0x73, 0x12, 0x22, 0x0a, 0x05, 0x6d, 0x6f,
0x76, 0x65,
 0x73, 0x18, 0x01, 0x20, 0x03, 0x28, 0x0b, 0x32, 0x0c, 0x2e, 0x6b, 0x61, 0x74, 0x61,
0x67, 0x6f,
 0x2e, 0x4d, 0x6f, 0x76, 0x65, 0x52, 0x05, 0x6d, 0x6f, 0x76, 0x65, 0x73, 0x32, 0x43,
0x0a, 0x0d,
 0x4b, 0x61, 0x74, 0x61, 0x67, 0x6f, 0x53, 0x65, 0x72, 0x76, 0x69, 0x63, 0x65, 0x12,
0x32, 0x0a,
 0x0c, 0x47, 0x65, 0x6e, 0x65, 0x72, 0x61, 0x74, 0x65, 0x4d, 0x6f, 0x76, 0x65, 0x12,
0x0d, 0x2e,
 0x6b, 0x61, 0x74, 0x61, 0x67, 0x6f, 0x2e, 0x4d, 0x6f, 0x76, 0x65, 0x73, 0x1a, 0x13,
0x2e, 0x6b,
 0x61, 0x74, 0x61, 0x67, 0x6f, 0x2e, 0x42, 0x6f, 0x74, 0x52, 0x65, 0x73, 0x70, 0x6f,
0x6e, 0x73,
 0x65, 0x42, 0x0b, 0x5a, 0x09, 0x2e, 0x2f, 0x3b, 0x6b, 0x61, 0x74, 0x61, 0x67, 0x6f,
0x62, 0x06,
 0x70, 0x72, 0x6f, 0x74, 0x6f, 0x33,
})

var (
 file_katago_proto_rawDescOnce sync.Once
 file_katago_proto_rawDescData []byte
)

func file_katago_proto_rawDescGZIP() []byte {
 file_katago_proto_rawDescOnce.Do(func() {
  file_katago_proto_rawDescData =
protoimpl.X.CompressGZIP(unsafe.Slice(unsafe.StringData(file_katago_proto_rawDesc),
len(file_katago_proto_rawDesc)))
 })
 return file_katago_proto_rawDescData
}

var file_katago_proto_msgTypes = make([]protoimpl.MessageInfo, 5)
var file_katago_proto_goTypes = []any{
 (*BotResponse)(nil), // 0: katago.BotResponse
 (*Diagnostics)(nil), // 1: katago.Diagnostics
 (*MovePSV)(nil),     // 2: katago.MovePSV
 (*Move)(nil),        // 3: katago.Move
 (*Moves)(nil),       // 4: katago.Moves
}
var file_katago_proto_depIdxs = []int32{
 1, // 0: katago.BotResponse.diagnostics:type_name -> katago.Diagnostics
 2, // 1: katago.Diagnostics.best_ten:type_name -> katago.MovePSV
 3, // 2: katago.Moves.moves:type_name -> katago.Move
 4, // 3: katago.KatagoService.GenerateMove:input_type -> katago.Moves
```

```
 0, // 4: katago.KatagoService.GenerateMove:output_type -> katago.BotResponse
 4, // [4:5] is the sub-list for method output_type
 3, // [3:4] is the sub-list for method input_type
 3, // [3:3] is the sub-list for extension type_name
 3, // [3:3] is the sub-list for extension extendee
 0, // [0:3] is the sub-list for field type_name
}

func init() { file_katago_proto_init() }
func file_katago_proto_init() {
 if File_katago_proto != nil {
  return
 }
 type x struct{}
 out := protoimpl.TypeBuilder{
  File: protoimpl.DescBuilder{
   GoPackagePath: reflect.TypeOf(x{}).PkgPath(),
   RawDescriptor: unsafe.Slice(unsafe.StringData(file_katago_proto_rawDesc),
len(file_katago_proto_rawDesc)),
   NumEnums:      0,
   NumMessages:   5,
   NumExtensions: 0,
   NumServices:   1,
  },
  GoTypes:           file_katago_proto_goTypes,
  DependencyIndexes: file_katago_proto_depIdxs,
  MessageInfos:      file_katago_proto_msgTypes,
 }.Build()
 File_katago_proto = out.File
 file_katago_proto_goTypes = nil
 file_katago_proto_depIdxs = nil
}
```

microservices/repository/katago.go

```
package repository

import (
 "bytes"
 "context"
 "encoding/json"
 "fmt"
 "github.com/google/uuid"
 "go.uber.org/zap"
 "net/http"
 "team_exe/internal/domain/game"

 "team_exe/internal/adapters"
 "team_exe/internal/bootstrap"
)

type KatagoRepository struct {
 cfg       *bootstrap.Config
 log       *zap.SugaredLogger
 redis     *adapters.AdapterRedis
 mongo     *adapters.AdapterMongo
 kataGoURL string
 client    *http.Client
}

func NewKatagoRepository(cfg *bootstrap.Config, log *zap.SugaredLogger)
*KatagoRepository {
 kataGoURL := cfg.KatagoBotUrl
 return &KatagoRepository{
  cfg:       cfg,
  log:       log,
  redis:     adapters.NewAdapterRedis(cfg),
  mongo:     adapters.NewAdapterMongo(cfg),
  kataGoURL: kataGoURL,
  client:    &http.Client{},
 }
}

func generateUUID() string {
 return uuid.New().String()
}

type SelectMoveRequest struct {
 BoardSize int      `json:"board_size"`
 Moves     []string `json:"moves"`
}

func (k *KatagoRepository) GenerateMove(ctx context.Context, moves []string)
(game.BotResponse, error) {
 reqBody, err := json.Marshal(SelectMoveRequest{
  BoardSize: 19,
  Moves:     moves,
 })
 if err != nil {
  return game.BotResponse{}, fmt.Errorf("failed to marshal request: %w", err)
 }
```

```
 req, err := http.NewRequestWithContext(ctx, http.MethodPost, k.kataGoURL,
bytes.NewBuffer(reqBody))
 k.log.Info(req)
 if err != nil {
  return game.BotResponse{}, fmt.Errorf("failed to create request: %w", err)
 }
 req.Header.Set("Content-Type", "application/json")

 resp, err := k.client.Do(req)
 if err != nil {
  return game.BotResponse{}, fmt.Errorf("failed to send request: %w", err)
 }
 defer resp.Body.Close()

 if resp.StatusCode != http.StatusOK {
  return game.BotResponse{}, fmt.Errorf("unexpected status code: %d", resp.StatusCode)
 }

 var result game.BotResponse
 if err := json.NewDecoder(resp.Body).Decode(&result); err != nil {
  return game.BotResponse{}, fmt.Errorf("failed to decode response: %w", err)
 }

 return result, nil
}
```

internal/domain/game/move.go

```
package game

type Move struct {
 Color       string `json:"color"`
 Coordinates string `json:"coordinates"`
}

type MovePSV struct {
 Move string `json:"move"`
 PSV  int    `json:"psv"`
}

type Diagnostics struct {
 BestTen []MovePSV `json:"best_ten"`
 BotMove string    `json:"bot_move"`
 Score   float64   `json:"score"`
 WinProb float64   `json:"winprob"`
}

type BotResponse struct {
 BotMove     string      `json:"bot_move"`
 Diagnostics Diagnostics `json:"diagnostics"`
 RequestID   string      `json:"request_id"`
}

type Moves struct {
 Moves []Move `json:"moves"`
}
```

internal/httpresponse/response.go

```
package httpresponse

import (
 "encoding/json"
 "fmt"
 "net/http"
)

type Response[T any] struct {
 Status int `json:"Status"`
 Body   any `json:"Body,omitempty"`
}

type ErrorResponse struct {
 ErrorDescription string `json:"ErrorDescription"`
}

const INTERNALERRORJSON = "{\"status\": 500,\"body\":{\"error\": \"Internal server
error\"}}"

const MALFORMEDJSON_errorDesc = "json unmarshalling error"

func WriteResponseWithStatus(w http.ResponseWriter, status int, body any) {
 //logger := slog.With("requestID", ctx.Value("traceID"))
 w.Header().Set("Content-Type", "application/json")
 jsonByte, err := marshalStatusJson(status, body)
 if err != nil {
  WriteInternalErrorResponse(w)
  return
 }
 _, err = w.Write(jsonByte)
 if err != nil {
  WriteInternalErrorResponse(w)
  return
 }
 //logger.Info("response", "status", status, "body", body)
}

func marshalStatusJson(status int, body any) ([]byte, error) {
 response := Response[any]{
  Status: status,
  Body:   body,
 }
 marshal, err := json.Marshal(response)
 if err != nil {
  return nil, err
 }
 return marshal, nil
}

func WriteInternalErrorResponse(w http.ResponseWriter) {
 // := slog.With("requestID", ctx.Value("traceID"))
 // implementation similar to http.Error, only difference is the Content-type
 w.Header().Set("Content-Type", "application/json")
 w.WriteHeader(500)
 _, _ = fmt.Fprintln(w, INTERNALERRORJSON)
 //logger.Info("response internal error", "body", INTERNALERRORJSON)
```

```
}
```

internal/usecase/auth/auth.go

```
package auth

import (
 "errors"
 userDomain "team_exe/internal/domain/user"
 "team_exe/internal/random"
)

type AuthUsecaseHandler struct {
 userStorage    UserStorage
 sessionStorage SessionStorage
}

func NewUserUsecaseHandler(u UserStorage, s SessionStorage) *AuthUsecaseHandler {
 return &AuthUsecaseHandler{
  userStorage:    u,
  sessionStorage: s,
 }
}

type UserStorage interface {
 CheckExists(username string) bool
 GetUser(username string) (userDomain.User, bool)
 GetUserByID(id int) (userDomain.User, bool)
}

type SessionStorage interface {
 GetUserIdBySession(sessionID string) (userID string, ok bool)
 StoreSession(sessionID string, userID string)
 DeleteSession(sessionID string) (ok bool)
}

var (
 ErrUserNotFound    = errors.New("user with provided username was not found")
 ErrWrongPassword   = errors.New("wrong password")
 ErrSessionNotFound = errors.New("session was not found")
)

func (a *AuthUsecaseHandler) CheckAuthorized(sessionID string) (ok bool, user
userDomain.User) {
 userID, found := a.sessionStorage.GetUserIdBySession(sessionID)
 if !found {
  return false, userDomain.User{}
 }
 user, ok = a.userStorage.GetUserByID(userID)
 if !ok {
  return false, userDomain.User{}
 }
 return ok, user
}

func (a *AuthUsecaseHandler) LoginUser(providedUsername string, providedPassword string)
(sessionID string, err error) {
 exists := a.userStorage.CheckExists(providedUsername)
 if !exists {
  return "", ErrUserNotFound
 }
```

```
 userFromDb, _ := a.userStorage.GetUser(providedUsername)
 if providedPassword != userFromDb.PasswordHash {
  return "", ErrWrongPassword
 }

 sessionID = random.RandString(64)
 a.sessionStorage.StoreSession(sessionID, userFromDb.ID)
 return sessionID, nil
}

func (a *AuthUsecaseHandler) LogoutUser(sessionID string) error {
 _, ok := a.sessionStorage.GetUserIdBySession(sessionID)
 if !ok {
  return ErrSessionNotFound
 }
 if !a.sessionStorage.DeleteSession(sessionID) {
  return ErrSessionNotFound
 }
 return nil
}

// Đ•Đ¾Đ²Ñ‹Đ¹ Đ¼ĐµÑ‚Đ¾Đ´ Đ´Đ»Ñ• Đ¿Đ¾Đ»ÑƒÑ‡ĐµĐ½Đ¸Ñ• userID Đ¸Đ· Ñ•ĐµÑ•Ñ•Đ¸Đ¸
func (a *AuthUsecaseHandler) GetUserIdFromSession(sessionID string) (string, error) {
 userID, ok := a.sessionStorage.GetUserIdBySession(sessionID)
 if !ok {
  return "", ErrSessionNotFound
 }
 return userID, nil
}
```

main.go

```
package main

import (
 "fmt"
 "io/fs"
 "os"
 "path/filepath"
 "strings"

 "github.com/jung-kurt/gofpdf"
)

func collectGoFiles(root string) (map[string]string, error) {
 files := make(map[string]string)

 err := filepath.WalkDir(root, func(path string, d fs.DirEntry, err error) error {
  if err != nil {
   return err
  }
  if !d.IsDir() && strings.HasSuffix(path, ".go") {
   content, err := os.ReadFile(path)
   if err != nil {
    return err
   }
   files[path] = string(content)
  }
  return nil
 })

 return files, err
}

func generatePDF(files map[string]string, output string) error {
 pdf := gofpdf.New("P", "mm", "A4", "")
 pdf.SetFont("Courier", "", 10)

 for path, content := range files {
  pdf.AddPage()
  pdf.Cell(40, 10, path)
  pdf.Ln(10)

  lines := strings.Split(content, "\n")
  for _, line := range lines {
   pdf.MultiCell(0, 4.5, line, "", "L", false)
  }
 }

 return pdf.OutputFileAndClose(output)
}

func main() {
 root := "." // Ð¼Ð¾Ð¶Ð½Ð¾ Ð·Ð°Ð¼ÐµÐ½Ð¸Ñ‚ÑŒ Ð½Ð° Ð¿ÑƒÑ‚ÑŒ Ð´Ð¾ Ð¿Ñ€Ð¾ÐµÐºÑ‚Ð°
 output := "project_code.pdf"

 files, err := collectGoFiles(root)
 if err != nil {
  fmt.Println("ÐžÑˆÐ¸Ð±ÐºÐ° Ð¿Ñ€Ð¸ Ñ•Ð±Ð¾Ñ€Ðµ Ñ„Ð°Ð¹Ð»Ð¾Ð²:", err)
```

```
  return
 }

 err = generatePDF(files, output)
 if err != nil {
  fmt.Println("ÐžÑˆÐ¸Ð±ÐºÐ° Ð¿Ñ€Ð¸ Ñ•Ð¾Ð·Ð´Ð°Ð½Ð¸Ð¸ PDF:", err)
  return
 }

 fmt.Println("âœ… PDF Ñ•Ð¾Ð·Ð´Ð°Ð½:", output)
}
```

microservices/proto/katago_grpc.pb.go

```
// Code generated by protoc-gen-go-grpc. DO NOT EDIT.
// versions:
// - protoc-gen-go-grpc v1.5.1
// - protoc             v3.21.12
// source: katago.proto

package katago

import (
 context "context"
 grpc "google.golang.org/grpc"
 codes "google.golang.org/grpc/codes"
 status "google.golang.org/grpc/status"
)

// This is a compile-time assertion to ensure that this generated file
// is compatible with the grpc package it is being compiled against.
// Requires gRPC-Go v1.64.0 or later.
const _ = grpc.SupportPackageIsVersion9

const (
 KatagoService_GenerateMove_FullMethodName = "/katago.KatagoService/GenerateMove"
)

// KatagoServiceClient is the client API for KatagoService service.
//
// For semantics around ctx use and closing/ending streaming RPCs, please refer to
https://pkg.go.dev/google.golang.org/grpc/?tab=doc#ClientConn.NewStream.
type KatagoServiceClient interface {
 GenerateMove(ctx context.Context, in *Moves, opts ...grpc.CallOption) (*BotResponse,
error)
}

type katagoServiceClient struct {
 cc grpc.ClientConnInterface
}

func NewKatagoServiceClient(cc grpc.ClientConnInterface) KatagoServiceClient {
 return &katagoServiceClient{cc}
}

func (c *katagoServiceClient) GenerateMove(ctx context.Context, in *Moves, opts
...grpc.CallOption) (*BotResponse, error) {
 cOpts := append([]grpc.CallOption{grpc.StaticMethod()}, opts...)
 out := new(BotResponse)
 err := c.cc.Invoke(ctx, KatagoService_GenerateMove_FullMethodName, in, out, cOpts...)
 if err != nil {
  return nil, err
 }
 return out, nil
}

// KatagoServiceServer is the server API for KatagoService service.
// All implementations must embed UnimplementedKatagoServiceServer
// for forward compatibility.
type KatagoServiceServer interface {
 GenerateMove(context.Context, *Moves) (*BotResponse, error)
```

```
 mustEmbedUnimplementedKatagoServiceServer()
}

// UnimplementedKatagoServiceServer must be embedded to have
// forward compatible implementations.
//
// NOTE: this should be embedded by value instead of pointer to avoid a nil
// pointer dereference when methods are called.
type UnimplementedKatagoServiceServer struct{}

func (UnimplementedKatagoServiceServer) GenerateMove(context.Context, *Moves)
(*BotResponse, error) {
 return nil, status.Errorf(codes.Unimplemented, "method GenerateMove not implemented")
}
func (UnimplementedKatagoServiceServer) mustEmbedUnimplementedKatagoServiceServer() {}
func (UnimplementedKatagoServiceServer) testEmbeddedByValue()                       {}

// UnsafeKatagoServiceServer may be embedded to opt out of forward compatibility for
this service.
// Use of this interface is not recommended, as added methods to KatagoServiceServer
will
// result in compilation errors.
type UnsafeKatagoServiceServer interface {
 mustEmbedUnimplementedKatagoServiceServer()
}

func RegisterKatagoServiceServer(s grpc.ServiceRegistrar, srv KatagoServiceServer) {
 // If the following call pancis, it indicates UnimplementedKatagoServiceServer was
 // embedded by pointer and is nil.  This will cause panics if an
 // unimplemented method is ever invoked, so we test this at initialization
 // time to prevent it from happening at runtime later due to I/O.
 if t, ok := srv.(interface{ testEmbeddedByValue() }); ok {
  t.testEmbeddedByValue()
 }
 s.RegisterService(&KatagoService_ServiceDesc, srv)
}

func _KatagoService_GenerateMove_Handler(srv interface{}, ctx context.Context, dec
func(interface{}) error, interceptor grpc.UnaryServerInterceptor) (interface{}, error) {
 in := new(Moves)
 if err := dec(in); err != nil {
  return nil, err
 }
 if interceptor == nil {
  return srv.(KatagoServiceServer).GenerateMove(ctx, in)
 }
 info := &grpc.UnaryServerInfo{
  Server:     srv,
  FullMethod: KatagoService_GenerateMove_FullMethodName,
 }
 handler := func(ctx context.Context, req interface{}) (interface{}, error) {
  return srv.(KatagoServiceServer).GenerateMove(ctx, req.(*Moves))
 }
 return interceptor(ctx, in, info, handler)
}

// KatagoService_ServiceDesc is the grpc.ServiceDesc for KatagoService service.
// It's only intended for direct use with grpc.RegisterService,
// and not to be introspected or modified (even as a copy)
```

```
var KatagoService_ServiceDesc = grpc.ServiceDesc{
 ServiceName: "katago.KatagoService",
 HandlerType: (*KatagoServiceServer)(nil),
 Methods: []grpc.MethodDesc{
  {
   MethodName: "GenerateMove",
   Handler:    _KatagoService_GenerateMove_Handler,
  },
 },
 Streams:  []grpc.StreamDesc{},
 Metadata: "katago.proto",
}
```

internal/bootstrap/config.go

```
package bootstrap

import (
 "github.com/spf13/viper"
)

type Config struct {
 ServerPort    string `mapstructure:"SERVER_PORT"`
 GpuServerIp   string `mapstructure:"GPU_SERVER_IP"`
 GpuServerPort string `mapstructure:"GPU_SERVER_PORT"`
 KatagoBotUrl  string `mapstructure:"KATAGO_BOT_URL"`
 RedisUrl      string `mapstructure:"REDIS_URL"`
 MongoUri      string `mapstructure:"MONGO_URI"`
 IsLocalCors   bool   `mapstructure:"LOCAL_CORS"`
}

func Setup(cfgPath string) (*Config, error) {
 viper.SetConfigFile(cfgPath)

 err := viper.ReadInConfig()
 if err != nil {
  return nil, err
 }

 var cfg Config

 err = viper.Unmarshal(&cfg)
 if err != nil {
  return nil, err
 }

 return &cfg, nil
}
```

internal/domain/katago.go

```
package domain
```

internal/domain/sgf/sgf.go

```
package sgf

// GameTree Đ¿Ñ€ĐµĐ´Ñ•Ñ,Đ°Đ²Đ»Ñ•ĐµÑ, Đ¾Đ´Đ½Đ¾ Đ´ĐµÑ€ĐµĐ²Đ¾ Đ² SGF (ÑƒĐ·ĐµĐ» +
Đ²Đ°Ñ€Đ¸Đ°Đ½Ñ‚)
type GameTree struct {
 Nodes    []Node      // ĐŸĐ¾Ñ•Đ»ĐµĐ´Đ¾Đ²Đ°Ñ,ĐµĐ»ŃŒĐ½Đ¾Ñ•Ñ,ŃŒ ÑƒĐ·Đ»Đ¾Đ²
(Đ¾Ñ•Đ½Đ¾Đ²Đ½Đ°Ñ• Đ»Đ¸Đ½Đ¸Ñ•)
 Children []*GameTree // Đ’Đ°Ñ€Đ¸Đ°Đ½Ñ‚ (Đ²Đ°Ñ€Đ¸Đ°Ñ,Đ¸Đ²Đ½Ñ‚Đµ Đ»Đ¸Đ½Đ¸Đ¸)
}

// Node Đ¿Ñ€ĐµĐ´Ñ•Ñ,Đ°Đ²Đ»Ñ•ĐµÑ, Đ¾Đ´Đ¸Đ½ ÑƒĐ·ĐµĐ» SGF (Đ½Đ°Đ±Đ¾Ñ€ Ñ•Đ²Đ¾Đ¹Ñ•Ñ,Đ²,
Ñ,Đ°ĐºĐ¸Ñ… ĐºĐ°Đº B[pd], W[dd], C[...])
type Node struct {
 Properties map[string][]string // Đ¡Đ²Đ¾Đ¹Ñ•Ñ,Đ²Đ° Đ¼Đ¾Đ³ÑƒÑ, Đ¿Đ¾Đ²Ñ,Đ¾Ñ€Ñ•Ñ,ŃŒÑ•Ñ•
(Đ½Đ°Đ¿Ñ€Đ¸Đ¼ĐµÑ€, AB[aa][bb])
}

// SGF Đ¿Ñ€ĐµĐ´Ñ•Ñ,Đ°Đ²Đ»Ñ•ĐµÑ, ĐºĐ¾Ñ€Đ½ĐµĐ²Đ¾Đ¹ Ñ•Đ»ĐµĐ¼ĐµĐ½Ñ, SGF-Ñ„Đ°Đ¹Đ»Đ°
type SGF struct {
 Root *GameTree
}
```

internal/errors/error.go

```
package errors

import "errors"

var (
 ErrUserNotFound     = errors.New("user with provided username was not found")
 ErrWrongPassword    = errors.New("wrong password")
 ErrSessionNotFound  = errors.New("session was not found")
 ErrCreateGameFailed = errors.New("create game failed")
 ErrJoinGameFailed   = errors.New("join game failed")
 ErrGameNotFound     = errors.New("game not found")
)
```

internal/repository/auth.go

```
package repo

import "team_exe/internal/domain/user"

type UserMapStorage struct {
 users map[int]user.User
}

func NewMapUserStorage() *UserMapStorage {
 storage := &UserMapStorage{users: make(map[int]user.User)}
 storage.users[5] = user.User{
  ID:           "5",
  Username:     "artem",
  PasswordHash: "755",
  PasswordSalt: "",
 }

 storage.users[4] = user.User{
  ID:           "4",
  Username:     "FunnyRockfish",
  PasswordHash: "770",
  PasswordSalt: "",
 }
 return storage
}

func (u UserMapStorage) CheckExists(username string) bool {
 for _, v := range u.users {
  if v.Username == username {
   return true
  }
 }
 return false
}

func (u UserMapStorage) GetUser(username string) (user.User, bool) {
 for _, v := range u.users {
  if v.Username == username {
   return v, true
  }
 }
 return user.User{}, false
}

func (u UserMapStorage) GetUserByID(id int) (user.User, bool) {
 for _, v := range u.users {
  if v.ID == id {
   return v, true
  }
 }
 return user.User{}, false
}

type SessionMapStorage struct {
 sessions map[string]string
 users    map[string]string
}
```

```
func (u SessionMapStorage) DeleteSession(sessionID string) (ok bool) {
 _, found := u.sessions[sessionID]
 if !found {
  return false
 }
 delete(u.sessions, sessionID)
 return true
}

func NewSessionMapStorage() *SessionMapStorage {
 return &SessionMapStorage{
  sessions: make(map[string]string),
  users:    make(map[string]string),
 }
}

func (u SessionMapStorage) GetUserIdBySession(sessionID string) (string, bool) {
 if v, ok := u.sessions[sessionID]; ok {
  return v, true
 } else {
  return "", false
 }
}

func (u SessionMapStorage) StoreSession(sessionID string, userID string) {
 u.sessions[sessionID] = userID
 u.users[userID] = sessionID
 return
}
```

internal/repository/game.go

```
package repo

import (
 "context"
 "errors"
 "github.com/google/uuid"
 "github.com/redis/go-redis/v9"
 "go.mongodb.org/mongo-driver/bson"
 "go.mongodb.org/mongo-driver/mongo"
 "go.mongodb.org/mongo-driver/mongo/options"
 "go.uber.org/zap"
 "net/http"
 "team_exe/internal/bootstrap"
 "team_exe/internal/domain/game"
 "time"
)

type GameRepository struct {
 cfg    bootstrap.Config
 log    *zap.SugaredLogger
 redis  *redis.Client
 mongo  *mongo.Database
 client *http.Client
}

func NewGameRepository(cfg bootstrap.Config, log *zap.SugaredLogger, redis
*redis.Client, mongo *mongo.Database) *GameRepository {
 return &GameRepository{
  cfg:    cfg,
  log:    log,
  redis:  redis,
  mongo:  mongo,
  client: &http.Client{},
 }
}

func (g *GameRepository) GenerateGameKey(ctx context.Context) string {
 return uuid.New().String()
}

func (g *GameRepository) PutGameToMongoDatabase(ctx context.Context, gameData game.Game)
bool {
 ctx, cancel := context.WithTimeout(ctx, 5*time.Second)
 defer cancel()

 collection := g.mongo.Collection("games")

 _, err := collection.InsertOne(ctx, gameData)
 if err != nil {
  g.log.Errorf("failed to insert game to database: %v", err)
  return false
 }

 g.log.Infof("game inserted successfully with key: %s", gameData.GameKey)

 return true
}
```

```
func (g *GameRepository) AddPlayer(ctx context.Context, newUser game.GameUser, gameKey
string) bool {
 ctx, cancel := context.WithTimeout(ctx, 5*time.Second)
 defer cancel()

 collection := g.mongo.Collection("games")

 filter := bson.M{"game_key": gameKey}

 update := bson.M{}
 if newUser.Color == "white" {
  update = bson.M{
   "$push": bson.M{
    "users": newUser,
   },
   "$set": bson.M{
    "player_white": newUser.ID,
   },
  }
 } else {
  update = bson.M{
   "$push": bson.M{
    "users": newUser,
   },
   "$set": bson.M{
    "player_black": newUser.ID,
   },
  }
 }

 opts := options.Update().SetUpsert(false)

 res, err := collection.UpdateOne(ctx, filter, update, opts)
 if err != nil {
  g.log.Errorf("failed to update game to database: %v", err)
  return false
 }

 if res.MatchedCount == 0 {
  g.log.Infof("Đ¸Đ³Ñ€Đ° Ñ• ĐºĐ»ÑŽÑ‡Đ¾Đ¼ %s Đ½Đµ Đ½Đ°Đ¹Đ´ĐµĐ½Đ°", gameKey)
 }

 g.log.Infof("ĐŸĐ¾Đ»ÑŒĐ·Đ¾Đ²Đ°Ñ‚ĐµĐ»Ñ Đ´Đ¾Đ±Đ°Đ²Đ»ĐµĐ½ Đº Đ¸Đ³Ñ€Đµ Ñ• ĐºĐ»ÑŽÑ‡Đ¾Đ¼ %s",
gameKey)

 return true
}

func (g *GameRepository) ConvertToUserFromJoinReq(ctx context.Context, joinRequest
game.GameJoinRequest) game.GameUser {
 user := game.GameUser{}
 user.ID = joinRequest.UserID
 user.Role = joinRequest.Role
 user.Color = g.CalculateUserColor(ctx, joinRequest.GameKey, joinRequest.UserID)
 return user
}

func (g *GameRepository) CalculateUserColor(ctx context.Context, gameKey string, userID
```

```
string) string {
 ctx, cancel := context.WithTimeout(ctx, 5*time.Second)
 defer cancel()

 collection := g.mongo.Collection("games")

 filter := bson.M{"game_key": gameKey}

 var result game.Game
 err := collection.FindOne(ctx, filter).Decode(&result)

 if err != nil {
  if errors.Is(err, mongo.ErrNoDocuments) {
   g.log.Error("Ð¸Ð³Ñ€Ð° Ñ• ID %s Ð½Ðµ Ð½Ð°Ð¹Ð´ÐµÐ½Ð°", gameKey)
  }
  return ""
 }

 colorOfOpponent := ""
 for _, user := range result.Users {
  if user.ID != userID {
   colorOfOpponent = user.Color
  }
 }

 if colorOfOpponent == "black" {
  return "white"
 }
 return "black"
}

func (g *GameRepository) GetGameByGameKey(ctx context.Context, gameKey string) game.Game
{
 ctx, cancel := context.WithTimeout(ctx, 5*time.Second)
 defer cancel()

 collection := g.mongo.Collection("games")

 filter := bson.M{"game_key": gameKey}

 var result game.Game
 err := collection.FindOne(ctx, filter).Decode(&result)

 if err != nil {
  if err == mongo.ErrNoDocuments {
   g.log.Error("Ð¸Ð³Ñ€Ð° Ñ• ID %s Ð½Ðµ Ð½Ð°Ð¹Ð´ÐµÐ½Ð°", gameKey)
  }
 }

 return result
}

func (g *GameRepository) SaveSGFToRedis(key string, sgfText string) error {
 ctx := context.Background()
 return g.redis.Set(ctx, key, sgfText, 0).Err()
}

func (g *GameRepository) LoadSGFFromRedis(key string) (string, error) {
 ctx := context.Background()
```

```
 return g.redis.Get(ctx, key).Result()
}

func (g *GameRepository) GetAllActiveGames() ([]game.Game, error) {
 ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
 defer cancel()
 collection := g.mongo.Collection("games")
 filter := bson.M{
  "status": "active",
 }
 var result []game.Game
 cursor, err := collection.Find(ctx, filter)
 if err != nil {
  g.log.Error(err)
  return result, err
 }

 defer cursor.Close(ctx)
 for cursor.Next(ctx) {
  var play game.Game
  err = cursor.Decode(&play)
  if err != nil {
   g.log.Error(err)
   return result, err
  }
  result = append(result, play)
 }

 return result, nil
}

func (g *GameRepository) GetActiveGameByUserId(ctx context.Context, userID string)
([]game.Game, error) {
 ctx, cancel := context.WithTimeout(ctx, 5*time.Second)
 defer cancel()
 collection := g.mongo.Collection("games")
 filter := bson.M{
  "$or": []bson.M{
   {"player_black": userID},
   {"player_white": userID},
  },
 }
 var result []game.Game
 cursor, err := collection.Find(ctx, filter)
 if err != nil {
  g.log.Error(err)
  return result, err
 }
 defer cursor.Close(ctx)
 for cursor.Next(ctx) {
  var play game.Game
  err = cursor.Decode(&play)
  if err != nil {
   g.log.Error(err)
   return result, err
  }
  result = append(result, play)
 }
 return result, nil
```

```
}
```

microservices/usecase/katago_rpc.go

```
package usecase

import (
 "context"
 "team_exe/internal/domain/game"
 katagoRPC "team_exe/microservices/proto"
)

type KatagoStore interface {
 GenerateMove(ctx context.Context, moves []string) (game.BotResponse, error)
}

type KatagoUseCase struct {
 store KatagoStore
 katagoRPC.UnimplementedKatagoServiceServer
}

func NewKatagoUseCase(store KatagoStore) *KatagoUseCase {
 return &KatagoUseCase{
  store: store,
 }
}

func (k *KatagoUseCase) GenerateMove(ctx context.Context, in *katagoRPC.Moves)
(*katagoRPC.BotResponse, error) {
 // ÐŸÑ€ÐµÐ¾Ð±Ñ€Ð°Ð·ÑƒÐµÐ¼ RPC-Ñ•Ñ‚Ñ€ÑƒÐºÑ‚ÑƒÑ€Ñƒ Ð² Ð´Ð¾Ð¼ÐµÐ½Ð½Ñ‹Ðµ Ð¼Ð¾Ð´ÐµÐ»Ð¸
 moves := ConvertRPCMovesToDomain(*in)
 movesStrings := extractCoordinates(moves)

 // Ð’Ñ‹Ð·Ð¾Ð² Ð»Ð¾Ð³Ð¸ÐºÐ¸ Ð³ÐµÐ½ÐµÑ€Ð°Ñ†Ð¸Ð¸ Ñ…Ð¾Ð´Ð° Ñ‡ÐµÑ€ÐµÐ· store
 botResponseDomain, err := k.store.GenerateMove(ctx, movesStrings)
 if err != nil {
  return nil, err
 }

 // ÐŸÑ€ÐµÐ¾Ð±Ñ€Ð°Ð·ÑƒÐµÐ¼ Ð´Ð¾Ð¼ÐµÐ½Ð½Ñ‹Ð¹ Ð¾Ñ‚Ð²ÐµÑ‚ Ð¾Ð±Ñ€Ð°Ñ‚Ð½Ð¾ Ð² RPC-Ñ•Ñ‚Ñ€ÑƒÐºÑ‚ÑƒÑ€Ñƒ
 resp := &katagoRPC.BotResponse{
  BotMove: botResponseDomain.BotMove,
  //RequestId: botResponseDomain.RequestId,
  // Ð•Ñ•Ð»Ð¸ Ñƒ Ð²Ð°Ñ• ÐµÑ•Ñ‚ÑŒ Ð´Ð¾Ð¿Ð¾Ð»Ð½Ð¸Ñ‚ÐµÐ»ÑŒÐ½Ñ‹Ðµ Ð¿Ð¾Ð»Ñ•, Ð¸Ñ… Ñ‚Ð¾Ð¶Ðµ
Ð½ÑƒÐ¶Ð½Ð¾ Ð¿Ñ€ÐµÐ¾Ð±Ñ€Ð°Ð·Ð¾Ð²Ð°Ñ‚ÑŒ
 }
 return resp, nil
}

func extractCoordinates(moves game.Moves) []string {
 coords := make([]string, 0)
 for _, m := range moves.Moves {
  coords = append(coords, m.Coordinates)
 }
 return coords
}

func ConvertRPCMovesToDomain(movesOld katagoRPC.Moves) game.Moves {
 domainMoves := make([]game.Move, 0)
 for _, m := range movesOld.Moves {
  move := game.Move{
```

```
   Coordinates: m.Coordinates,
   Color:       m.Color,
  }
  domainMoves = append(domainMoves, move)
 }
 return game.Moves{Moves: domainMoves}
}
```

cmd/main.go

```
package main

import (
 "context"
 "github.com/go-chi/chi/v5"
 "github.com/go-chi/chi/v5/middleware"
 "go.uber.org/zap"
 "google.golang.org/grpc"
 "net/http"
 "os"
 "os/signal"
 "syscall"
 "time"

 "team_exe/internal/adapters"
 "team_exe/internal/bootstrap"
 authDelivery "team_exe/internal/delivery/auth"
 gameDelivery "team_exe/internal/delivery/game"
 katagoDelivery "team_exe/internal/delivery/katago"
 ownMiddleware "team_exe/internal/middleware"
 katagoProto "team_exe/microservices/proto"
)

type mainDeliveryHandler struct {
 auth   *authDelivery.AuthHandler
 katago *katagoDelivery.KatagoHandler
 game   *gameDelivery.GameHandler
}

type dataBaseAdapters struct {
 redisAdapter *adapters.AdapterRedis
 mongoAdapter *adapters.AdapterMongo
}

func main() {
 logger := NewLogger()
 cfg, err := bootstrap.Setup(".env")
 if err != nil {
  logger.Error("Failed to setup configuration", zap.Error(err))
  return
 }

 ctx, cancel := context.WithCancel(context.Background())
 defer cancel()

 go handleShutdown(cancel, logger)

 databaseAdapters := initDatabaseAdapters(ctx, logger, *cfg)
 defer databaseAdapters.mongoAdapter.Close(ctx)
 defer databaseAdapters.redisAdapter.Close(ctx)

 grpcKatago, err := grpc.Dial("host.docker.internal:8082", grpc.WithInsecure())
 if err != nil {
  logger.Fatal("Failed to dial grpc", zap.Error(err))
 }
 defer grpcKatago.Close()
```

```
 r := chi.NewRouter()
 handlers := initializeDeliveryHandlers(ctx, *cfg, logger, grpcKatago, databaseAdapters)
 handlers.Router(r, cfg.IsLocalCors)

 port := ":8080"
 logger.Infof("Server is running on port %s", port)
 if err := http.ListenAndServe(port, r); err != nil {
  logger.Fatal("Failed to start server", zap.Error(err))
 }
}

func NewLogger() *zap.SugaredLogger {
 logger, err := zap.NewProduction()
 if err != nil {
  panic("failed to initialize logger: " + err.Error())
 }
 return logger.Sugar()
}

func (h *mainDeliveryHandler) Router(r *chi.Mux, isLocalCors bool) {
 if isLocalCors {
  r.Use(ownMiddleware.CORS)
 }
 r.Use(middleware.Logger)

 r.Post("/login", h.auth.Login)
 r.Delete("/logout", h.auth.Logout)
 r.Post("/autoBotGenerateMove", h.katago.HandleGenerateMove)
 r.Post("/NewGame", h.game.HandleNewGame)
 r.Post("/JoinGame", h.game.HandleJoinGame)
 r.Get("/startGame", h.game.HandleStartGame)
}

func initDatabaseAdapters(ctx context.Context, log *zap.SugaredLogger, cfg
bootstrap.Config) *dataBaseAdapters {
 mongoAdapter := adapters.NewAdapterMongo(&cfg)
 if err := mongoAdapter.Init(ctx); err != nil {
  log.Fatal("Ð•Ðµ ÑƒÐ´Ð°Ð»Ð¾ÑŒ Ð¸Ð½Ñ†Ð¸Ð°Ð»Ð¸Ð·Ð¸Ñ€Ð¾Ð²Ð°Ñ‚ÑŒ MongoDB",
zap.Error(err))
 }

 redisAdapter := adapters.NewAdapterRedis(&cfg)
 if err := redisAdapter.Init(ctx); err != nil {
  log.Fatal("Ð•Ðµ ÑƒÐ´Ð°Ð»Ð¾ÑŒ Ð¸Ð½Ñ†Ð¸Ð°Ð»Ð¸Ð·Ð¸Ñ€Ð¾Ð²Ð°Ñ‚ÑŒ Redis",
zap.Error(err))
 }

 log.Info("Ð•Ð´Ð°Ð¿,ÐµÑ€Ñ‹ Ð±Ð°Ð· Ð´Ð°Ð½Ð½Ñ‹Ñ… Ð¸Ð½Ð¸Ñ†Ð¸Ð°Ð»Ð¸Ð·Ð¸Ñ€Ð¾Ð²Ð°Ð½Ñ‹")
 return &dataBaseAdapters{
  redisAdapter: redisAdapter,
  mongoAdapter: mongoAdapter,
 }
}

func initializeDeliveryHandlers(
 ctx context.Context,
 cfg bootstrap.Config,
 log *zap.SugaredLogger,
 grpcKatago *grpc.ClientConn,
```

```
 databaseAdapters *dataBaseAdapters,
) *mainDeliveryHandler {
 katagoManager := katagoProto.NewKatagoServiceClient(grpcKatago)
 katagoDeliveryHandler := katagoDelivery.NewKatagoHandler(cfg, log, katagoManager)

 authDeliveryHandler := authDelivery.NewMapAuthHandler(databaseAdapters.redisAdapter)
 gameDeliveryHandler := gameDelivery.NewGameHandler(cfg, log,
databaseAdapters.mongoAdapter, databaseAdapters.redisAdapter, authDeliveryHandler)

 return &mainDeliveryHandler{
  auth:   authDeliveryHandler,
  katago: katagoDeliveryHandler,
  game:   gameDeliveryHandler,
 }
}

func handleShutdown(cancelFunc context.CancelFunc, log *zap.SugaredLogger) {
 sigs := make(chan os.Signal, 1)
 signal.Notify(sigs, syscall.SIGINT, syscall.SIGTERM)
 <-sigs
 log.Info("Received shutdown signal")
 cancelFunc()
 time.Sleep(1 * time.Second) // Ð´Ð°Ñ‚ÑŒ Ð²Ñ€ÐµÐ¼Ñ• Ð·Ð°ÐºÑ€Ñ‹Ñ‚ÑŒ Ñ•Ð¾ÐµÐ´Ð¸Ð½ÐµÐ½Ð¸Ñ•
}
```

internal/adapters/mongo.go

```
package adapters

import (
 "context"
 "fmt"
 "log"
 "time"

 "go.mongodb.org/mongo-driver/mongo"
 "go.mongodb.org/mongo-driver/mongo/options"

 "team_exe/internal/bootstrap"
)

type AdapterMongo struct {
 Client   *mongo.Client
 Database *mongo.Database
 cfg      *bootstrap.Config
}

func NewAdapterMongo(cfg *bootstrap.Config) *AdapterMongo {
 return &AdapterMongo{
  cfg: cfg,
 }
}

func (a *AdapterMongo) Init(ctx context.Context) error {
 clientOpts := options.Client().ApplyURI(a.cfg.MongoUri)

 ctxConnect, cancel := context.WithTimeout(ctx, 10*time.Second)
 defer cancel()

 client, err := mongo.Connect(ctxConnect, clientOpts)
 if err != nil {
  return fmt.Errorf("Ð¾ÑˆÐ¸Ð±ÐºÐ° Ð¿Ð¾Ð´ÐºÐ»ÑŽÑ‡ÐµÐ½Ð¸Ñ• Ðº MongoDB: %w", err)
 }

 if err = client.Ping(ctx, nil); err != nil {
  log.Fatalf("Ð•Ðµ Ñƒ�ð´Ð°Ð»Ð¾ÑŒ Ð¿Ñ€Ð¾Ð¿Ð¸Ð½Ð³Ð¾Ð²Ð°Ñ‚ÑŒ MongoDB: %v", err)
 }

 a.Database = client.Database("team_exe")

 log.Println("Ð£Ñ•Ð¿ÐµÑˆÐ½Ð¾ Ð¿Ð¾Ð´ÐºÐ»ÑŽÑ‡ÐµÐ½Ð¾ Ðº MongoDB")
 return nil
}

func (a *AdapterMongo) Close(ctx context.Context) error {
 if a.Client != nil {
  return a.Client.Disconnect(ctx)
 }
 return nil
}
```

internal/delivery/auth/auth.go

```
package auth

import (
 "encoding/json"
 "errors"
 "io"
 "log/slog"
 "net/http"
 "team_exe/internal/adapters"
 "team_exe/internal/httpresponse"
 repo "team_exe/internal/repository"
 authUC "team_exe/internal/usecase/auth"
 "time"
)

type AuthHandler struct {
 usecaseHandler *authUC.AuthUsecaseHandler
}

func NewMapAuthHandler(redis *adapters.AdapterRedis) *AuthHandler {
 return &AuthHandler{
  usecaseHandler: authUC.NewUserUsecaseHandler(
   repo.NewMapUserStorage(),
   repo.NewSessionRedisStorage(redis.GetClient()),
  ),
 }
}

type loginRequest struct {
 Username string `json:"Username"`
 Password string `json:"Password"`
}

func (a *AuthHandler) Login(w http.ResponseWriter, r *http.Request) {
 requestBody, err := io.ReadAll(r.Body)
 if err != nil {
  slog.Error(err.Error())
  return
 }
 loginData := loginRequest{}
 err = json.Unmarshal(requestBody, &loginData)
 if err != nil {
  slog.Error(err.Error())
  httpresponse.WriteResponseWithStatus(w, 400,
   httpresponse.ErrorResponse{ErrorDescription: httpresponse.MALFORMEDJSON_errorDesc})
  return
 }
 sessionID, err := a.usecaseHandler.LoginUser(loginData.Username, loginData.Password)
 if err != nil {
  if errors.Is(err, authUC.ErrUserNotFound) {
   httpresponse.WriteResponseWithStatus(w, 400,
    httpresponse.ErrorResponse{ErrorDescription: "ÐŸÐ¾Ð»ÑŒÐ·Ð¾Ð²Ð°Ñ‚ÐµÐ»Ñ ÐµÐ
Ð½Ð°Ð¹Ð´ÐµÐ½"})
   return
  } else if errors.Is(err, authUC.ErrWrongPassword) {
   httpresponse.WriteResponseWithStatus(w, 400,
    httpresponse.ErrorResponse{ErrorDescription: "Ð•ÐµÐ²ÐµÑ€Ð½Ñ‹Ð¹ Ð¿Ð°Ñ€Ð¾Ð»ÑŒ"})
```

```
   return
  }
  // Đ¸Đ½Đ°Ñ• Đ½ĐµĐ¿Ñ€ĐµĐ´Đ²Đ¸Đ´ĐµĐ½Đ½Đ°Ñ• Đ¾Ñˆ�đ¸Đ±ĐºĐ°
  httpresponse.WriteResponseWithStatus(w, 500,
   httpresponse.ErrorResponse{ErrorDescription: err.Error()})
  return
 }

 http.SetCookie(w, &http.Cookie{
  Name:     "sessionID",
  Value:    sessionID,
  Expires:  time.Now().Add(time.Hour * 10),
  Secure:   true,
  HttpOnly: true,
 })
 httpresponse.WriteResponseWithStatus(w, 200, nil)
}

func (a *AuthHandler) Logout(w http.ResponseWriter, r *http.Request) {
 sessionIDCookie, err := r.Cookie("sessionID")
 if err != nil {
  if errors.Is(err, http.ErrNoCookie) {
   httpresponse.WriteResponseWithStatus(w, 400,
    httpresponse.ErrorResponse{ErrorDescription: http.ErrNoCookie.Error()})
   return
  }
 }
 err = a.usecaseHandler.LogoutUser(sessionIDCookie.Value)
 if err != nil {
  httpresponse.WriteResponseWithStatus(w, 400,
   httpresponse.ErrorResponse{ErrorDescription: err.Error()})
  return
 }
 httpresponse.WriteResponseWithStatus(w, 200, nil)
}

// Đ•Đ¾Đ²Ñ‹Đ¹ Đ¼ĐµÑ‚Đ¾Đ´ Đ»Ñ• Đ¿Đ¾Đ»ÑƒÑ‡ĐµĐ½Đ¸Ñ• userID Đ¸Đ· Ñ•ĐµÑ•Ñ•Đ¸Đ¸
func (a *AuthHandler) GetUserID(w http.ResponseWriter, r *http.Request) string {
 sessionIDCookie, err := r.Cookie("sessionID")
 if err != nil {
  if errors.Is(err, http.ErrNoCookie) {
   httpresponse.WriteResponseWithStatus(w, 400,
    httpresponse.ErrorResponse{ErrorDescription: "Đ•Đµ Đ½Đ°Đ¹Đ´ĐµĐ½Đ° cookie
sessionID"})
   return ""
  }
  httpresponse.WriteResponseWithStatus(w, 400,
   httpresponse.ErrorResponse{ErrorDescription: err.Error()})
  return ""
 }

 userID, err := a.usecaseHandler.GetUserIdFromSession(sessionIDCookie.Value)
 if err != nil {
  if errors.Is(err, authUC.ErrSessionNotFound) {
   httpresponse.WriteResponseWithStatus(w, http.StatusUnauthorized,
    httpresponse.ErrorResponse{ErrorDescription: "Đ¡ĐµÑ•Ñ•Đ¸Ñ• Đ½Đµ Đ½Đ°Đ¹Đ´ĐµĐ½Đ°
Đ¸Đ»Đ¸ Đ¸Ñ•Ñ‚ĐµĐºĐ»Đ°"})
   return ""
  }
```

```
  httpresponse.WriteResponseWithStatus(w, http.StatusInternalServerError,
   httpresponse.ErrorResponse{ErrorDescription: err.Error()})
  return ""
 }

 return userID
}
```

internal/delivery/game/game.go

```
package game

import (
 "bytes"
 "encoding/json"
 "github.com/gorilla/websocket"
 "go.uber.org/zap"
 "io"
 "log"
 "net/http"
 "sync"
 "team_exe/internal/adapters"
 "team_exe/internal/bootstrap"
 "team_exe/internal/delivery/auth"
 "team_exe/internal/domain/game"
 "team_exe/internal/httpresponse"
 repo "team_exe/internal/repository"
 gameuc "team_exe/internal/usecase/game"
)

type GameHandler struct {
 cfg          bootstrap.Config
 log          *zap.SugaredLogger
 gameUC       *gameuc.GameUseCase
 mongoAdapter *adapters.AdapterMongo
 redisAdapter *adapters.AdapterRedis
 authHandler  *auth.AuthHandler
}

var upgrader = websocket.Upgrader{
 CheckOrigin: func(r *http.Request) bool { return true },
}

var activeGames = make(map[string]*game.Game)
var activeGamesMu sync.RWMutex

func NewGameHandler(cfg bootstrap.Config, log *zap.SugaredLogger, mongoAdapter
*adapters.AdapterMongo, redisAdapter *adapters.AdapterRedis, authHandler
*auth.AuthHandler) *GameHandler {
 return &GameHandler{
  cfg:         cfg,
  log:         log,
  gameUC:      gameuc.NewGameUseCase(repo.NewGameRepository(cfg, log,
redisAdapter.GetClient(), mongoAdapter.Database)),
  authHandler: authHandler,
 }
}

func (g *GameHandler) HandleNewGame(w http.ResponseWriter, r *http.Request) {
 if r.Method != http.MethodPost {
  g.log.Error("Only POST method is allowed")
  httpresponse.WriteResponseWithStatus(w, http.StatusMethodNotAllowed, "Only POST method
is allowed")
  return
 }

 // Ð¿Ñ€Ð¾Ð²ÐµÑ€Ð°, Ñ‡Ñ‚Ð¸ Ð¸Ð³Ñ€Ð¾ Ñ•Ð²Ð¾Ð±Ð¾Ð´ÐµÐ½! TODO
```

```
 bodyBytes, err := io.ReadAll(r.Body)
 if err != nil {
  g.log.Error("Failed to read body:", err)
  httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, "Failed to read request
body")
  return
 }
 defer r.Body.Close()

 g.log.Infof("Incoming JSON: %s", string(bodyBytes))

 var gameData game.Game
 decoder := json.NewDecoder(bytes.NewReader(bodyBytes))
 decoder.DisallowUnknownFields()

 if err = decoder.Decode(&gameData); err != nil {
  g.log.Error("JSON decode error:", err)
  httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, "Invalid JSON:
"+err.Error())
  return
 }

 if len(gameData.Users) != 1 {
  g.log.Error("Ð½ÐµÐ²ÐµÑ€Ð½Ñ‹Ð¹ json")
  httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, "Invalid JSON:
"+string(bodyBytes))
  return
 }

 userID := g.authHandler.GetUserID(w, r)

 g.log.Infof("New game is from id: %s", userID)

 ctx := r.Context()

 alreadyIsInGame, err := g.gameUC.HasUserActiveGamesByUserId(ctx, userID)
 if err != nil {
  g.log.Error(err)
  httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, "Ð¾ÑˆÐ¸Ð±Ð°Ð° Ð¿Ñ€Ð¸
Ð´Ð¾Ð±Ð°Ð²Ð»ÐµÐ½Ð¸Ð¸ Ð² Ð¸Ð³Ñ€Ñƒ: "+err.Error())
  return
 }
 if alreadyIsInGame {
  g.log.Error("Ð¿Ð¾Ð»ÑŒÐ·Ð¾Ð²Ð°Ñ‚ÐµÐ»ÑŒ ÑƒÐ¶Ðµ Ñ•Ð¾Ñ•Ñ‚Ð¾Ð¸Ñ‚ Ð² Ð¸Ð³Ñ€Ðµ!") //TODO
Ð´Ð¾Ð±Ð°Ð²Ð¸Ñ‚ÑŒ Ð¾Ñ‚Ð¾Ð±Ñ€Ð°Ð¶ÐµÐ½Ð¸Ðµ id Ð¸Ð³Ñ€Ñ‹
  httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, "Ð¾ÑˆÐ¸Ð±Ð°Ð° Ð¿Ñ€Ð¸
Ð´Ð¾Ð±Ð°Ð²Ð»ÐµÐ½Ð¸Ð¸ Ð² Ð¸Ð³Ñ€Ñƒ: ÑƒÐ¶Ðµ Ñ•Ð¾Ñ•Ñ‚Ð¾Ð¸Ñ‚ Ð² Ð¸Ð³Ñ€Ðµ")
  return
 }

 err, gameKey := g.gameUC.CreateGame(ctx, gameData)
 if err != nil {
  g.log.Error(err)
  return
 }

 resp := game.GameCreateResponse{
  UniqueKey: gameKey,
```

```
 }

 g.log.Info("New Game Created with key: " + gameKey)
 httpresponse.WriteResponseWithStatus(w, http.StatusOK, resp)
}

func (g *GameHandler) HandleJoinGame(w http.ResponseWriter, r *http.Request) {
 if r.Method != http.MethodPost {
  g.log.Error("Only POST method is allowed")
  httpresponse.WriteResponseWithStatus(w, http.StatusMethodNotAllowed, "Only POST method
is allowed")
  return
 }

 userID := g.authHandler.GetUserID(w, r)

 g.log.Infof("New game is from id: %s", userID)

 bodyBytes, err := io.ReadAll(r.Body)
 if err != nil {
  g.log.Error("Failed to read body:", err)
  httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, "Failed to read request
body")
  return
 }
 defer r.Body.Close()

 g.log.Infof("Incoming JSON: %s", string(bodyBytes))

 var newGamerRequest game.GameJoinRequest
 decoder := json.NewDecoder(bytes.NewReader(bodyBytes))
 decoder.DisallowUnknownFields()

 if err = decoder.Decode(&newGamerRequest); err != nil {
  g.log.Error("JSON decode error:", err)
  httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, "Invalid JSON:
"+err.Error())
  return
 }

 newGamerRequest.UserID = userID

 if newGamerRequest.GameKey == "" || newGamerRequest.UserID == "" ||
newGamerRequest.Role == "" {
  g.log.Error("Ð½ÐµÐ²ÐµÑ€Ð½Ñ‹Ð¹ json")
  httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, "Invalid JSON:
"+string(bodyBytes))
  return
 }

 ctx := r.Context()

 alreadyIsInGame, err := g.gameUC.HasUserActiveGamesByUserId(ctx,
newGamerRequest.UserID)
 if err != nil {
  g.log.Error(err)
  httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, "Ð¾ÑˆÐ¸Ð±ÐºÐ° Ð¿Ñ€Ð¸
Ð´Ð¾Ð±Ð°Ð²Ð»ÐµÐ½Ð¸Ð¸ Ð² Ð¸Ð³Ñ€Ñƒ: "+err.Error())
  return
```

```
 }
 if alreadyIsInGame {
  g.log.Error("Ð¿Ð¾Ð»ÑŒÐ·Ð¾Ð²Ð°Ñ‚ÐµÐ»ÑŒ Ñƒ Ð¶Ðµ Ñ•Ð¾Ñ•Ñ‚Ð¾Ñ, Ð² Ð¸Ð³Ñ€Ðµ!") //TODO Ð´Ð¾Ð±Ð°Ð²Ð¸Ñ‚ÑŒ Ð¾Ñ‚Ð¾Ð±Ñ€Ð°Ð¶ÐµÐ½Ð¸Ðµ id Ð¸Ð³Ñ€Ñ‹
  httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, "Ð¾Ñˆ Ð±Ð°Ð° Ð¿Ñ€Ð¸ Ð´Ð¾Ð±Ð°Ð²Ð»ÐµÐ½Ð¸Ð¸ Ð² Ð¸Ð³Ñ€Ñƒ: Ñƒ Ð¶Ðµ Ñ•Ð¾Ñ•Ñ‚Ð¾Ñ, Ð² Ð¸Ð³Ñ€Ðµ")
  return
 }

 err = g.gameUC.JoinGame(ctx, newGamerRequest)
 if err != nil {
  g.log.Error(err)
  return
 }

 resp := JsonOKResponse{
  Text: "ÑŽÐ·ÐµÑ€ Ñƒ•Ð¿ÐµÑˆÐ½Ð¾ Ð´Ð¾Ð±Ð°Ð²Ð»ÐµÐ½",
 }

 g.log.Info(resp.Text)
 httpresponse.WriteResponseWithStatus(w, http.StatusOK, resp)
}

func (g *GameHandler) HandleStartGame(w http.ResponseWriter, r *http.Request) {
 conn, err := upgrader.Upgrade(w, r, nil)
 if err != nil {
  log.Println("upgrade error:", err)
  return
 }

 ctx := r.Context()
 gameID := r.URL.Query().Get("game_id")

 playerID := g.authHandler.GetUserID(w, r)

 if gameID == "" || playerID == "" {
  g.log.Error("Ð¾Ñ•Ñ‚Ñ•Ñ•Ñ‚Ð²ÑƒÑŽÑ‚ Ð¿Ð»Ð¾Ñ• gameID Ð¸Ð»Ð¸ playerID")
  httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, "Ð¾Ñ•Ñ‚Ñ•Ñ•Ñ‚Ð²ÑƒÑŽÑ‚ Ð¿Ð»Ð¾Ñ• gameID Ð¸Ð»Ð¸ playerID")
  return
 }

 if !g.gameUC.IsUserInGameByGameId(ctx, gameID, playerID) {
  g.log.Error("Ð¿Ð¾Ð»ÑŒÐ·Ð¾Ð²Ð°Ñ‚ÐµÐ»Ñ Ð½Ðµ Ð² Ð¸Ð³Ñ€Ðµ!")
  httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, "Ð¿Ð¾Ð»ÑŒÐ·Ð¾Ð²Ð°Ñ‚ÐµÐ»Ñ Ð½Ðµ Ð² Ð¸Ð³Ñ€Ðµ!")
  return
 }

 activeGamesMu.Lock()
 ag, ok := activeGames[gameID]
 if !ok {
  foundGame, err := g.gameUC.GetGameByID(ctx, gameID)
  if err != nil {
   activeGamesMu.Unlock()
   g.log.Error(err)
   httpresponse.WriteResponseWithStatus(w, http.StatusBadRequest, err.Error())
   return
  }
```

```
  ag = &foundGame
  activeGames[gameID] = ag
 }
 activeGamesMu.Unlock()

 playerBID, playerWID := ag.PlayerBlack, ag.PlayerWhite

 if playerID == playerBID {
  if ag.PlayerBlackWS != nil {
   _ = ag.PlayerBlackWS.WriteMessage(websocket.TextMessage, []byte("Ð’Ñ‹ Ð±Ñ‹Ð»Ð¸
Ð¾Ñ‚ÐºÐ»ÑŽÑ‡ÐµÐ½Ñ‹, Ð½Ð¾Ð²Ð¾Ðµ Ñ•Ð¾ÐµÐ´Ð¸Ð½ÐµÐ½Ð¸Ðµ Ñ•Ð¾Ð·Ð´Ð°Ð½Ð¾."))
   _ = ag.PlayerBlackWS.Close()
  }
  ag.PlayerBlackWS = conn
 } else if playerID == playerWID {
  if ag.PlayerWhiteWS != nil {
   _ = ag.PlayerWhiteWS.WriteMessage(websocket.TextMessage, []byte("Ð’Ñ‹ Ð±Ñ‹Ð»Ð¸
Ð¾Ñ‚ÐºÐ»ÑŽÑ‡ÐµÐ½Ñ‹, Ð½Ð¾Ð²Ð¾Ðµ Ñ•Ð¾ÐµÐ´Ð¸Ð½ÐµÐ½Ð¸Ðµ Ñ•Ð¾Ð·Ð´Ð°Ð½Ð¾."))
   _ = ag.PlayerWhiteWS.Close()
  }
  ag.PlayerWhiteWS = conn
 } else {
  g.log.Error("Unknown player id:", playerID)
  return
 }

 defer conn.Close()

 defer func() {
  activeGamesMu.Lock()
  defer activeGamesMu.Unlock()

  if ag.PlayerBlackWS == conn {
   ag.PlayerBlackWS = nil
  }
  if ag.PlayerWhiteWS == conn {
   ag.PlayerWhiteWS = nil
  }
 }()

 for {
  var move game.Move
  if err = conn.ReadJSON(&move); err != nil {
   g.log.Error("read error:", err)
   return
  }
  g.log.Info("ÐŸÐ¾Ð»ÑƒÑ‡ÐµÐ½ Ñ…Ð¾Ð´: ", move)

  var opponentWS *websocket.Conn
  if playerID == playerBID {
   opponentWS = ag.PlayerWhiteWS
  } else {
   opponentWS = ag.PlayerBlackWS
  }

  sgfString, err := g.gameUC.AddMoveToGameSgf(gameID, move)
  if err != nil {
   g.log.Error(err)
   conn.WriteMessage(websocket.TextMessage, []byte(err.Error()))
```

```
  }

  resp := game.GameStateResponse{
   Move: move,
   SGF:  sgfString,
  }

  if opponentWS != nil {
   if err := opponentWS.WriteJSON(resp); err != nil {
    g.log.Error("Write to opponent error:", err)
    opponentWS.Close()

    activeGamesMu.Lock()
    if ag.PlayerBlackWS == opponentWS {
     ag.PlayerBlackWS = nil
    }
    if ag.PlayerWhiteWS == opponentWS {
     ag.PlayerWhiteWS = nil
    }
    activeGamesMu.Unlock()
   }
  }

 }
}

type JsonOKResponse struct {
 Text string `json:"text"`
}
```

internal/delivery/katago/katago.go

```
package katago

import (
 "encoding/json"
 "go.uber.org/zap"
 "net/http"
 "team_exe/internal/bootstrap"
 "team_exe/internal/domain/game"
 katagoUC "team_exe/internal/usecase/katago"
 katagoProto "team_exe/microservices/proto"
)

type GenerateMoveRequest game.Moves

type BotMoveResponse struct {
 BotMove game.Move `json:"bot_move"`
}
type KatagoHandler struct {
 cfg        bootstrap.Config
 log        *zap.SugaredLogger
 katagoGRPC katagoProto.KatagoServiceClient
}

func NewKatagoHandler(cfg bootstrap.Config, log *zap.SugaredLogger, katago
katagoProto.KatagoServiceClient) *KatagoHandler {
 // repo := repository.NewKatagoRepository(&cfg, log)
 return &KatagoHandler{
  cfg:        cfg,
  log:        log,
  katagoGRPC: katago,
 }
}

func (k *KatagoHandler) HandleGenerateMove(w http.ResponseWriter, r *http.Request) {
 if r.Method != http.MethodPost {
  writeJSONError(k.log, w, http.StatusMethodNotAllowed, "Only POST method is allowed")
  return
 }

 var movesToBot game.Moves
 if err := json.NewDecoder(r.Body).Decode(&movesToBot); err != nil {
  writeJSONError(k.log, w, http.StatusBadRequest, "Invalid JSON: "+err.Error())
  return
 }

 ctx := r.Context()

 botMove, err := katagoUC.GenMove(ctx, movesToBot, k.katagoGRPC)
 if err != nil {
  k.log.Errorf("failed to generate bot move: %v", err)
  writeJSONError(k.log, w, http.StatusInternalServerError, "Failed to generate bot
move")
  return
 }

 resp := BotMoveResponse{BotMove: botMove}
```

```
 writeJSON(k.log, w, http.StatusOK, resp)
}

func writeJSON(log *zap.SugaredLogger, w http.ResponseWriter, status int, data
interface{}) {
 w.Header().Set("Content-Type", "application/json; charset=utf-8")
 w.WriteHeader(status)
 if err := json.NewEncoder(w).Encode(data); err != nil {
  log.Errorf("writeJSON encode error: %v", err)
 }
}

func writeJSONError(log *zap.SugaredLogger, w http.ResponseWriter, status int, msg
string) {
 w.Header().Set("Content-Type", "application/json; charset=utf-8")
 w.WriteHeader(status)
 _ = json.NewEncoder(w).Encode(map[string]string{"error": msg})
 log.Debugf("writeJSONError: %s", msg)
}
```