ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
Кафедра ЕОМ

Кваліфікаційна робота
Перший (бакалаврський) рівень

Багатокористувацький веб-сервіс для гри в шахи на основі протоколу WebSocket

Автор:
Бєліков А. О.,
ст. гр. КІУКІ-21-1

Керівник:
Ярошевич Р. О.,
ст. викл. каф. ЕОМ

02

Мета та завдання кваліфікаційної роботи

Мета: розробка багатокористувацького веб -сервісу для гри в шахи.

Завдання:
- провести аналіз сучасних ігрових веб-сервісів;
- розробити механізм створення ігрових кімнат з обраними користувачем параметрами;
- реалізувати правила гри «Шахи», враховуючи відображення допустимих ходів для обраної користувачем фігури;
- реалізувати можливість комунікації гравців за допомогою вбудованого чату;
- організувати збереження завершених ігор із забезпеченням можливості їх перегляду;
- реалізувати безпечний стандарт JWT для керування доступом до ресурсів сервера;
- впровадити шифрування трафіку з використанням протоколів HTTP Secure та WebSocket Secure;
- провести тестування та вимірювання продуктивності роботи розробленого додатку.

03

Огляд існуючих рішень

Популярний сервіс Chess.com не надає відкритого доступу до вихідного коду та відомостей про внутрішню імплементацію. З позиції користувача, наявність платного функціоналу та велика кількість реклами погіршує загальний досвід використання даної платформи.

Lichess.org

Безкоштовний сервіс Lichess є відкритим проєктом, побудованим з використанням наступних технологій: мова програмування Scala, фреймворк Play, бази даних Redis, MongoDB та пошуковий рушій ElasticSearch. Велика кількість залежностей ускладнює аналіз архітектури і робить недоцільним використання Lichess у навчальних цілях.

Основним недоліком сучасних платформ є складність їх використання у навчальних цілях та в якості основи для власних проєктів.

04

Використані технології

Серверний додаток: мова програмування Go та бібліотека Gorilla WebSocket.

- вбудована підтримка паралельних обчислень;
- статична типізація;
- швидкість компіляції та виконання;
- стандартна бібліотека охоплює протокол HTTP

- швидка та добре протестована реалізація протоколу WebSocket

Клієнтський додаток: мова програмування TypeScript та бібліотека React.

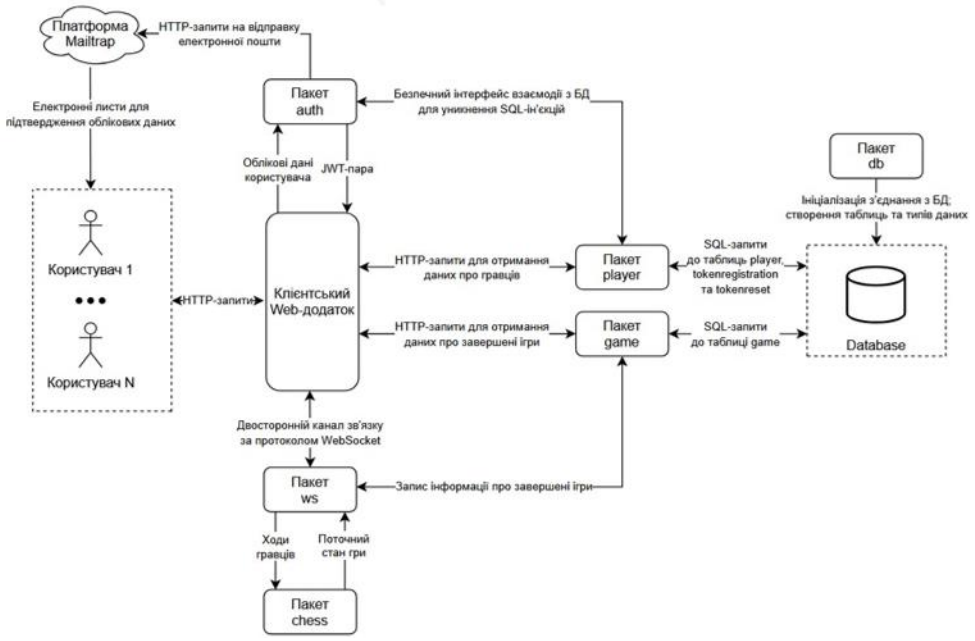
- статична типізація;
- підтримка виконання у браузері за рахунок компіляції у JavaScript

- полегшення розробки динамічних веб-сторінок;
- можливість багаторазово використовувати створені графічні компоненти

Зберігання завершених ігор та акаунтів гравців: безкоштовна СКБД з реляційною моделлю даних PostgreSQL.

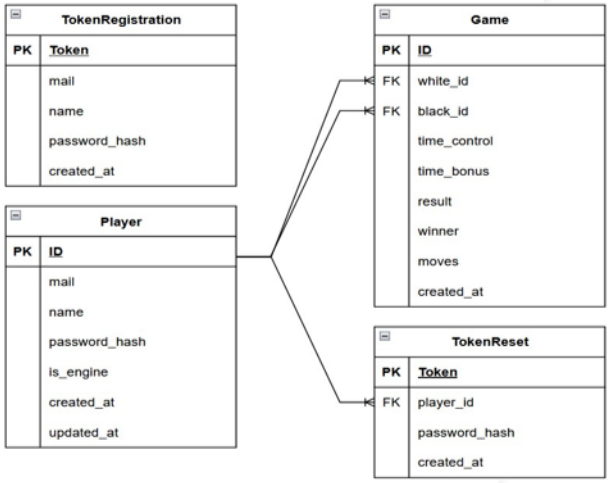
05

Архітектура розробленого веб -сервісу



06

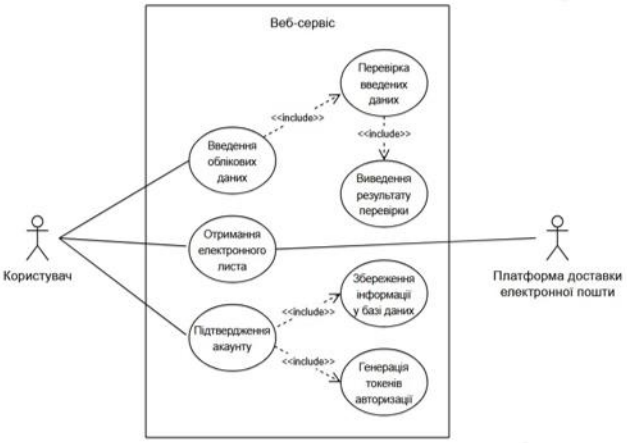
Діаграма сутностей та зв'язків бази даних

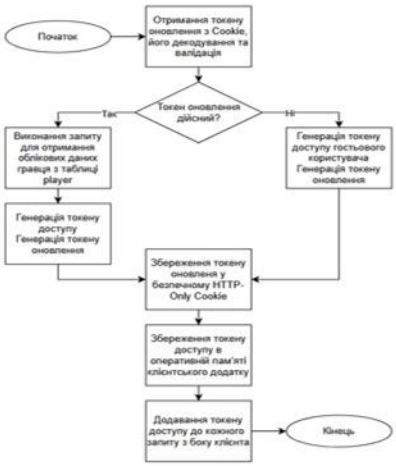


- Таблиця player зберігає інформацію про зареєстрованих гравців.
- Таблиця tokenregistration тимчасового зберігає облікові дані користувачів під час процедури підтвердження електронної пошти.
- У разі ініціації запиту на скидання пароля створюється запис у таблиці tokenreset, який містить хешоване значення нового пароля та унікальний токен для підтведження операції.
- Завершені шахові партії зберігаються у таблиці game.

07

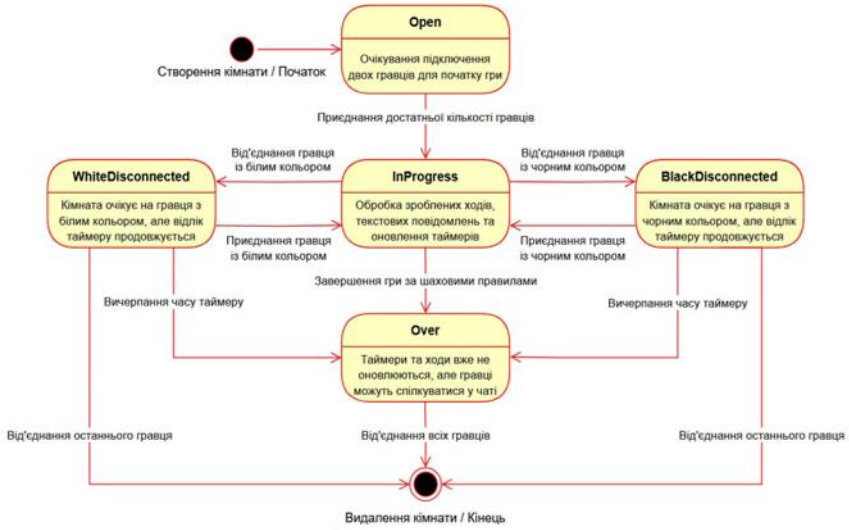
Автентифікація та авторизація





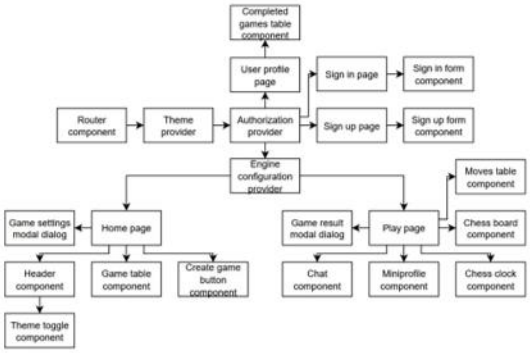
08

Діаграма станів об’єкту ігрової кімнати



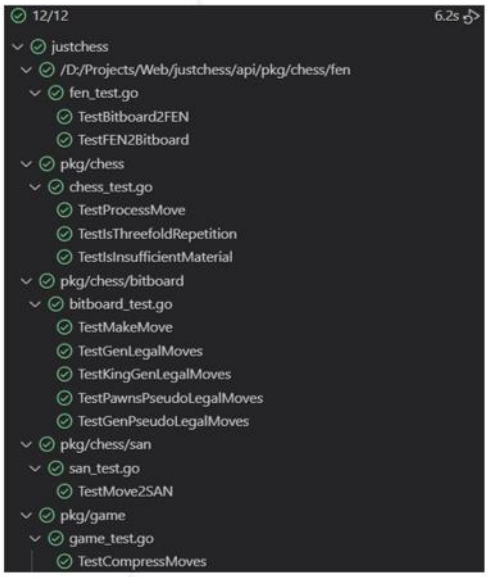
09

Розроблені компоненти графічного інтерфейсу



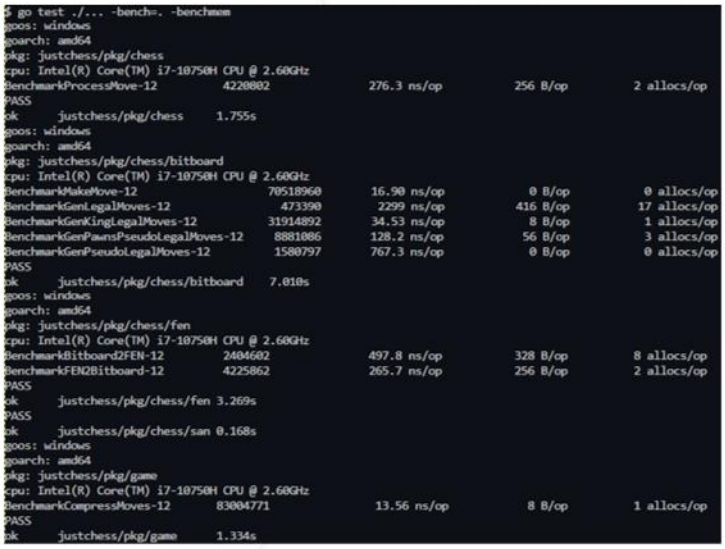
10

Аналіз отриманих результатів (1) модульне тестування



11

Аналіз отриманих результатів (2) вимірювання продуктивності



```
$ go test ./... -bench=. -benchmem
goos: windows
goarch: amd64
pkg: justchess/pkg/chess
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkProcessMove-12          4220802               276.3 ns/op           256 B/op          2 allocs/op
PASS
ok      justchess/pkg/chess     1.755s
goos: windows
goarch: amd64
pkg: justchess/pkg/chess/bitboard
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkMakeMove-12                70518960        16.90 ns/op            0 B/op          0 allocs/op
BenchmarkGenLegalMoves-12             473390         2299 ns/op           416 B/op         17 allocs/op
BenchmarkGenKingLegalMoves-12       31914892        34.53 ns/op            8 B/op          1 allocs/op
BenchmarkGenPawnsPseudoLegalMoves-12  8881086        128.2 ns/op           56 B/op          3 allocs/op
BenchmarkGenPseudoLegalMoves-12       1580797        767.3 ns/op            0 B/op          0 allocs/op
PASS
ok      justchess/pkg/chess/bitboard    7.010s
goos: windows
goarch: amd64
pkg: justchess/pkg/chess/fen
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkBitboard2FEN-12         2404602               497.8 ns/op           328 B/op          8 allocs/op
BenchmarkFEN2Bitboard-12         4225862               265.7 ns/op           256 B/op          2 allocs/op
PASS
ok      justchess/pkg/chess/fen 3.269s
PASS
ok      justchess/pkg/chess/san 0.168s
goos: windows
goarch: amd64
pkg: justchess/pkg/game
cpu: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
BenchmarkCompressMoves-12        83004771              13.56 ns/op            8 B/op          1 allocs/op
PASS
ok      justchess/pkg/game      1.334s
```

12

Висновки

У ході виконання кваліфікаційної роботи було:

- проаналізовано та визначено недоліки сучасних шахових веб-сервісів Chess.com та Lichess;
- спроєктовано алгоритми створення ігрових кімнат, проведення автентифікації та авторизації;
- реалізовано правила гри та засоби комунікації гравців;
- розроблено та розгорнуто багатокористувацький веб-сервіс з використанням безпечних протоколів WSS та HTTPS;
- спроєктовано структуру бази даних для збереження завершених ігор та облікових даних гравців;
- проведено тестування та вимірювання продуктивності роботи веб-сервісу.

Програмні компоненти реалізовано на сучасних мовах програмування Go та TypeScript у середовищі розробки Visual Studio Code. Розроблений додаток можна використовувати для вивчення правил гри та організації турнірів з шахів серед здобувачів університету. Середовище виконання мови Go підтримує виконання на всіх сучасних операційних системах.

Для порівняння, сучасний шаховий веб-сервіс Lichess орієнтований на розгортання лише у Linux-системах та споживає значну кількість апаратних ресурсів під час виконання. Велика кількість використаних технологій збільшує фінальний розмір серверного додатку Lichess близьким до 120 мегабайтів, а розроблений в даній роботі додаток займає 14 мегабайтів.

Проєкт є повністю робочим та виконує всі поставлені завдання. Перспективними напрямами подальшого удосконалення проєкту є впровадження штучного інтелекту для аналізу партій та розширення можливостей комунікації гравців шляхом реалізації форумів.

ДОДАТОК Б

Програмний код

Б.1 Реалізація автентифікації та авторизації

Лістинг Б.1.1 – Вміст файлу auth.go

```
import (
    "bytes"
    "crypto/rand"
    "encoding/json"
    "errors"
    "justchess/pkg/db"
    "justchess/pkg/player"
    "log"
    "net/http"
    "os"
    "regexp"
    "text/template"
    "time"
    "github.com/google/uuid"
    "golang.org/x/crypto/bcrypt"
)
type signInDTO struct {
    Login    string `json:"login"`
    Password string `json:"password"`
}
type mailData struct {
    Name            string
    VerificationURL string
}
type passwordResetDTO struct {
    Mail     string `json:"mail"`
    Password string `json:"password"`
}
type PlayerDTO struct {
    Player      player.Player `json:"player"`
    Role        Role          `json:"role"`
    AccessToken string        `json:"accessToken"`
}
type mailtrapDTO struct {
    From     map[string]string   `json:"from"`
    To       []map[string]string `json:"to"`
    Subject  string              `json:"subject"`
    Html     string              `json:"html"`
    Category string              `json:"category"`
}
```

```
var (
    nameRE = regexp.MustCompile(`[a-zA-Z]{1}[a-zA-Z0-9_]+`)
    mailRE = regexp.MustCompile(`[a-zA-Z0-9._]+@[a-zA-Z0-
9._]+\.[a-zA-Z0-9._]+`)
)
func Mux() *http.ServeMux {
    mux := http.NewServeMux()
    mux.HandleFunc("GET /auth/refresh", refresh)
    mux.HandleFunc("GET /auth/guest", guest)
    mux.HandleFunc("GET /auth/verify", verify)
    mux.HandleFunc("POST /auth/signup", signUp)
    mux.HandleFunc("POST /auth/signin", signIn)
    mux.HandleFunc("POST /auth/reset", passwordReset)
    return mux
}
func signUp(rw http.ResponseWriter, r *http.Request) {
    var req player.Register
    err := json.NewDecoder(r.Body).Decode(&req)
    if err != nil {
        rw.WriteHeader(http.StatusBadRequest)
        return
    }
    if !nameRE.Match([]byte(req.Name)) ||
!mailRE.Match([]byte(req.Mail)) ||
        len(req.Password) < 5 || len(req.Password) > 72 {
        rw.WriteHeader(http.StatusBadRequest)
        return
    }
    if player.IsTakenNameOrMail(req.Name, req.Mail) {
        rw.WriteHeader(http.StatusConflict)
        return
    }
    hash, err :=
bcrypt.GenerateFromPassword([]byte(req.Password),
bcrypt.DefaultCost)
    if err != nil {
        log.Printf("cannot generate password hash: %v\n", err)
        rw.WriteHeader(http.StatusBadRequest)
        return
    }
    req.Password = string(hash)
    tx, err := db.Pool.Begin()
    if err != nil {
        log.Printf("Cannot begin a new transaction: %v\n", err)
        rw.WriteHeader(http.StatusInternalServerError)
        return
    }
    token := rand.Text()
    err = player.InsertTokenRegistration(token, req, tx)
    defer tx.Rollback()
    if err != nil {
        rw.WriteHeader(http.StatusConflict)
        return
```

```
    }
    data := mailData{
        Name:            req.Name,
        VerificationURL: os.Getenv("DOMAIN") +
"/verify?action=registration&token=" + token,
    }
    err = sendMail(req.Mail, "Email Verification",
        "./templates/mail-verify.html", data)
    if err != nil {
        log.Printf("cannot send verification email: %v\n", err)
        rw.WriteHeader(http.StatusBadRequest)
        return
    }
    if err = tx.Commit(); err != nil {
        log.Printf("cannot commit transaction: %v\n", err)
        rw.WriteHeader(http.StatusInternalServerError)
}}
func verify(rw http.ResponseWriter, r *http.Request) {
    action := r.URL.Query().Get("action")
    token := r.URL.Query().Get("token")
    if token == "" {
        rw.WriteHeader(http.StatusBadRequest)
        return
    }
    switch action {
    case "reset":
        completeReset(rw, token)
    case "registration":
        completeSignUp(rw, token)
    default:
        rw.WriteHeader(http.StatusBadRequest)
}}
func signIn(rw http.ResponseWriter, r *http.Request) {
    var req signInDTO
    err := json.NewDecoder(r.Body).Decode(&req)
    if err != nil {
        rw.WriteHeader(http.StatusBadRequest)
        return
    }
    p, err := player.SelectPlayerByLogin(req.Login)
    if err != nil ||
        bcrypt.CompareHashAndPassword([]byte(p.PasswordHash),
[]byte(req.Password)) != nil {
        rw.WriteHeader(http.StatusUnauthorized)
        return
    }
    completeAuth(rw, p, RolePlayer)
}
func passwordReset(rw http.ResponseWriter, r *http.Request) {
    var req passwordResetDTO
    err := json.NewDecoder(r.Body).Decode(&req)
    if err != nil || len(req.Password) < 5 || len(req.Password)
> 72 {
```

```
        rw.WriteHeader(http.StatusBadRequest)
        return
    }
    p, err := player.SelectPlayerByMail(req.Mail)
    if err != nil || p.Id == uuid.Nil {
        rw.WriteHeader(http.StatusNotFound)
        return
    }
    token := rand.Text()
    hash, err :=
bcrypt.GenerateFromPassword([]byte(req.Password),
bcrypt.DefaultCost)
    if err != nil {
        log.Printf("cannot generate hash: %v\n", err)
        rw.WriteHeader(http.StatusInternalServerError)
        return }
    tx, err := db.Pool.Begin()
    if err != nil {
        log.Printf("cannot begin transaction: %v\n", err)
        rw.WriteHeader(http.StatusInternalServerError)
        return
    }
    defer tx.Rollback()
    err = player.InsertTokenReset(token, p.Id.String(),
string(hash), tx)
    if err != nil {
        log.Printf("cannot insert token reset: %v\n", err)
        rw.WriteHeader(http.StatusInternalServerError)
        return
    }
    data := mailData{
        Name:            p.Name,
        VerificationURL: os.Getenv("DOMAIN") +
"/verify?action=reset&token=" + token,
    }
    if err = sendMail(p.Mail, "Password Reset",
        "./templates/password-reset.html", data); err != nil {
        log.Printf("cannot send mail: %v\n", err)
        rw.WriteHeader(http.StatusInternalServerError)
        return
    }
    if err = tx.Commit(); err != nil {
        rw.WriteHeader(http.StatusInternalServerError)
        return
}}
func completeSignUp(rw http.ResponseWriter, token string) {
    r, err := player.SelectTokenRegistration(token)
    if err != nil || r.Mail == "" {
        rw.WriteHeader(http.StatusNotFound)
        return
    }
    id := uuid.New()
    err = player.InsertPlayer(id.String(), r)
```

```
    if err != nil {
        log.Printf("cannot insert player %s: %v\n", r.Mail, err)
        rw.WriteHeader(http.StatusConflict)
        return
    }
    completeAuth(rw, player.Player{Id: id, Name: r.Name,
CreatedAt: time.Now()}, RolePlayer)
}
func completeReset(rw http.ResponseWriter, token string) {
    id, hash, err := player.SelectTokenReset(token)
    if err != nil || hash == "" {
        rw.WriteHeader(http.StatusNotFound)
        return
    }
    p, err := player.UpdatePasswordHash(hash, id)
    if err != nil || p.Id == uuid.Nil {
        rw.WriteHeader(http.StatusNotFound)
        return
    }
    completeAuth(rw, p, RolePlayer)
}
func sendMail(addr, subject, templatePath string, data mailData)
error {
    body, err := genTemplate(templatePath, data)
    if err != nil {
        return err
    }
    dto := mailtrapDTO{
        From: map[string]string{
            "email": os.Getenv("SMTP_FROM"),
            "name":  "Support",
        },
        To:       []map[string]string{{"email": addr}},
        Subject:  subject,
        Html:     body,
        Category: "Email verification",
    }
    payload, err := json.Marshal(dto)
    if err != nil {
        return err
    }
    req, err := http.NewRequest("POST",
"https://send.api.mailtrap.io/api/send",
        bytes.NewReader(payload))
    if err != nil {
        return err
    }
    req.Header.Add("Authorization", "Bearer
"+os.Getenv("MAILTRAP_API_TOKEN"))
    req.Header.Add("Content-Type", "application/json")
    res, err := http.DefaultClient.Do(req)
    if err != nil || (res.StatusCode != http.StatusOK &&
res.StatusCode != http.StatusNoContent) {
```

```
        return errors.New("Mailtrap error")
    }
    return res.Body.Close()
}
func genTemplate(path string, data mailData) (string, error) {
    t, err := template.ParseFiles(path)
    if err != nil {
        return "", err
    }
    var buff bytes.Buffer
    err = t.Execute(&buff, data)
    return buff.String(), err
}
func refresh(rw http.ResponseWriter, r *http.Request) {
    encoded, err := parseRefreshTokenCookie(r)
    if err != nil {
        rw.WriteHeader(http.StatusUnauthorized)
        return
    }
    cms, err := DecodeToken(encoded, "REFRESH_TOKEN_SECRET")
    if err != nil {
        rw.WriteHeader(http.StatusUnauthorized)
        return
    }
    p, err := player.SelectPlayerById(cms.Id.String())
    if err != nil {
        rw.WriteHeader(http.StatusUnauthorized)
        return
    }
    completeAuth(rw, p, RolePlayer)
}
func guest(rw http.ResponseWriter, r *http.Request) {
    id := uuid.New()
    guest := player.Player{
        Id:        id,
        Name:      "Guest-" + id.String()[0:8],
        CreatedAt: time.Now(),
    }
    completeAuth(rw, guest, RoleGuest)
}
func completeAuth(rw http.ResponseWriter, p player.Player, r
Role) {
    at, rt, err := generatePair(p.Id, p.Name, r)
    if err != nil {
        log.Printf("cannot generate pair: %v\n", err)
        rw.WriteHeader(http.StatusInternalServerError)
        return
    }
    setRefreshTokenCookie(rw, rt)
    rw.Header().Add("Content-Type", "application/json")
    err = json.NewEncoder(rw).Encode(PlayerDTO{Player: p,
AccessToken: at, Role: r})
    if err != nil {
```

```
        log.Printf("cannot decode response: %v\n", err)
        rw.WriteHeader(http.StatusInternalServerError)
        return
    }
}
func setRefreshTokenCookie(rw http.ResponseWriter, et string) {
    cookie := http.Cookie{
        Name:     "Authorization",
        Value:    "Bearer " + et,
        Path:     "/",
        MaxAge:   2592000, // Token will last 30 days.
        HttpOnly: true,
        Secure:   true,
        SameSite: http.SameSiteStrictMode,
    }
    http.SetCookie(rw, &cookie)
}
func parseRefreshTokenCookie(r *http.Request) (string, error) {
    cookie, err := r.Cookie("Authorization")
    if err != nil {
        return "", err
    }
    if len(cookie.Value) < 100 || cookie.Value[:7] != "Bearer "
{
        return "", err
    }
    return cookie.Value[7:], nil
}
```

Лістинг Б.1.2 – Вміст файлу jwt.go

```
package auth
import (
    "errors"
    "log"
    "os"
    "time"
    "github.com/golang-jwt/jwt/v5"
    "github.com/google/uuid"
)
type Role int
type CtxKey string
const (
    RoleGuest Role = iota
    RolePlayer
    Cms CtxKey = "claims"
)
type Claims struct {
    Id   uuid.UUID `json:"id"`
    Name string    `json:"username"`
    Role Role      `json:"role"`
    jwt.RegisteredClaims
}
```

```
func generateToken(id uuid.UUID, name string, r Role, secret
string, d time.Duration) (string, error) {
    unsigned := jwt.NewWithClaims(jwt.SigningMethodHS256,
Claims{id, name, r,
        jwt.RegisteredClaims{ExpiresAt:
jwt.NewNumericDate(time.Now().Add(d))},
    })
    return unsigned.SignedString([]byte(os.Getenv(secret)))
}
func generatePair(id uuid.UUID, name string, r Role) (at, rt
string, err error) {
    at, err = generateToken(id, name, r, "ACCESS_TOKEN_SECRET",
time.Minute*30)
    if err != nil {
        log.Printf("cannot generate access token: %v\n", err)
        return
    }
    rt, err = generateToken(id, name, r, "REFRESH_TOKEN_SECRET",
time.Hour*24*30)
    if err != nil {
        log.Printf("cannot generate refresh token: %v\n", err)
    }
    return
}
func DecodeToken(encoded, secret string) (Claims, error) {
    t, err := jwt.ParseWithClaims(encoded, &Claims{},
        func(t *jwt.Token) (any, error) {
            return []byte(os.Getenv(secret)), nil
        },
    )
    if err != nil {
        return Claims{}, err
    }
    c, ok := t.Claims.(*Claims)
    if !ok {
        err = errors.New("cannot parse claims")
    }
    return *c, err
}
```

Б.2 Генерація можливих ходів пішака

Лістинг Б.2.1 – Функція genPawnsPseudoLegalMoves (файл movegen.go)

```
func genPawnsPseudoLegalMoves(pawns, allies, enemies uint64, c
enums.Color,
    epTarget int) (moves []Move) {
    occupied := allies | enemies
    for ; pawns > 0; pawns &= pawns - 1 {
        pawnFrom := GetLSB(pawns)
        north, doubleNorth := 0, 0
```

```
        west, east, pawnBB := uint64(0), uint64(0),
uint64(1)<<pawnFrom
        canDoublePush := false
        if c == enums.White {
            north, doubleNorth = pawnFrom+8, pawnFrom+16
            west, east = pawnBB&notA<<7, pawnBB&notH<<9
            canDoublePush = pawnBB&0xFF00 != 0
        } else {
            north, doubleNorth = pawnFrom-8, pawnFrom-16
            west, east = pawnBB&notA>>9, pawnBB&notH>>7
            canDoublePush = pawnBB&0xFF000000000000 != 0
        }
        if (1<<north)&occupied == 0 {
            if (1<<north)&0xFF != 0 ||
uint64(1<<north)&0xFF00000000000000 != 0 {
                moves = append(moves, NewMove(north, pawnFrom,
enums.QueenPromo))
            } else {
                moves = append(moves, NewMove(north, pawnFrom,
enums.Quiet))
            }
            if canDoublePush && (1<<doubleNorth)&occupied == 0 {
                moves = append(moves, NewMove(doubleNorth,
pawnFrom, enums.DoublePawnPush))
            }
        }
        if west&enemies != 0 {
            if west&0xFF != 0 || west&0xFF00000000000000 != 0 {
                moves = append(moves, NewMove(GetLSB(west),
pawnFrom, enums.QueenPromoCapture))
            } else {
                moves = append(moves, NewMove(GetLSB(west),
pawnFrom, enums.Capture))
            }
        } else if epTarget != enums.NoSquare &&
west&(1<<epTarget) != 0 {
            moves = append(moves, NewMove(GetLSB(west),
pawnFrom, enums.EPCapture))
        }
        if east&enemies != 0 {
            if east&0xFF != 0 || east&0xFF00000000000000 != 0 {
                moves = append(moves, NewMove(GetLSB(east),
pawnFrom, enums.QueenPromoCapture))
            } else {
                moves = append(moves, NewMove(GetLSB(east),
pawnFrom, enums.Capture))
            }
        } else if epTarget != enums.NoSquare &&
east&(1<<epTarget) != 0 {
            moves = append(moves, NewMove(GetLSB(east),
pawnFrom, enums.EPCapture))
        }
    }
```

```
    return
     }
```

Б.3 Код пакету game

Лістинг Б.3.1 – Вміст файлу game.go

```
package game
import (
    "encoding/json"
    "log"
    "net/http"
    "github.com/google/uuid"
)
func Mux() *http.ServeMux {
    mux := http.NewServeMux()
    mux.HandleFunc("GET /game/id/{id}", getById)
    mux.HandleFunc("GET /game/player/{id}", getByPlayerId)
    return mux
}
func getById(rw http.ResponseWriter, r *http.Request) {
    id, err := uuid.Parse(r.PathValue("id"))
    if err != nil {
        rw.WriteHeader(http.StatusBadRequest)
        return
    }
    g, err := selectById(id.String())
    if err != nil {
        rw.WriteHeader(http.StatusNotFound)
        return
    }
    rw.Header().Add("Content-Type", "application/json")
    err = json.NewEncoder(rw).Encode(g)
    if err != nil {
        log.Printf("cannot encode game: %v\n", err)
        rw.WriteHeader(http.StatusInternalServerError)
    }
}
func getByPlayerId(rw http.ResponseWriter, r *http.Request) {
    id, err := uuid.Parse(r.PathValue("id"))
    if err != nil {
        rw.WriteHeader(http.StatusBadRequest)
        return
    }
    g, err := selectByPlayerId(id.String())
    if len(g) < 1 {
        rw.WriteHeader(http.StatusNotFound)
        return
    }
    rw.Header().Add("Content-Type", "application/json")
```

```
    err = json.NewEncoder(rw).Encode(g)
    if err != nil {
        log.Printf("cannot encode games: %v\n", err)
        rw.WriteHeader(http.StatusInternalServerError)
}}
```

Лістинг Б.3.2 – Вміст файлу gamerepository.go

```
package game
import (
    "errors"
    "justchess/pkg/chess"
    "justchess/pkg/chess/bitboard"
    "justchess/pkg/chess/enums"
    "justchess/pkg/chess/fen"
    "justchess/pkg/db"
    "time"
    "github.com/google/uuid"
    "github.com/lib/pq"
)
type GameDTO struct {
    Id          uuid.UUID             `json:"id"`
    WhiteId     uuid.UUID             `json:"whiteId"`
    BlackId     uuid.UUID             `json:"blackId"`
    TimeControl int                   `json:"timeControl"`
    TimeBonus   int                   `json:"timeBonus"`
    Result      enums.Result          `json:"result"`
    Winner      enums.Color           `json:"winner"`
    Moves       []chess.CompletedMove `json:"moves"`
    CreatedAt   time.Time             `json:"createdAt"`
    WhiteName   string                `json:"whiteName"`
    BlackName   string                `json:"blackName"`
}
type shortGameDTO struct {
    Id          uuid.UUID    `json:"id"`
    WhiteId     uuid.UUID    `json:"wid"`
    BlackId     uuid.UUID    `json:"bid"`
    Result      enums.Result `json:"r"`
    Winner      enums.Color  `json:"w"`
    MovesLen    int          `json:"m"`
    WhiteName   string       `json:"wn"`
    BlackName   string       `json:"bn"`
    TimeControl int          `json:"tc"`
    TimeBonus   int          `json:"tb"`
    CreatedAt   time.Time    `json:"ca"`
}
func selectById(id string) (g GameDTO, err error) {
    query :=
        `SELECT
            game.*,
            white_player.name AS white_name,
            black_player.name AS black_name
        FROM game
```

```
        JOIN player AS white_player ON game.white_id =
white_player.id
        JOIN player AS black_player ON game.black_id =
black_player.id
        WHERE game.id = $1;`
    rows, err := db.Pool.Query(query, id)
    if err != nil {
        return
    }
    if !rows.Next() {
        return g, errors.New("game not found")
    }
    var compressedMoves []int32
    err = rows.Scan(&g.Id, &g.WhiteId, &g.BlackId,
&g.TimeControl, &g.TimeBonus,
        &g.Result, &g.Winner, pq.Array(&compressedMoves),
&g.CreatedAt,
        &g.WhiteName, &g.BlackName)
    g.Moves = decompressMoves(compressedMoves, fen.DefaultFEN,
g.TimeControl)
    return
}
func selectByPlayerId(id string) (games []shortGameDTO, err
error) {
    query :=
        `SELECT
            game.*,
            white_player.name AS white_name,
            black_player.name AS black_name
        FROM game
        JOIN player AS white_player ON game.white_id =
white_player.id
        JOIN player AS black_player ON game.black_id =
black_player.id
        WHERE game.white_id = $1 OR game.black_id = $1
        ORDER BY game.created_at DESC LIMIT 100;`
    rows, err := db.Pool.Query(query, id)
    if err != nil {
        return
    }
    for i := 0; rows.Next(); i++ {
        var g shortGameDTO
        var compressedMoves []int32
        err = rows.Scan(&g.Id, &g.WhiteId, &g.BlackId,
&g.TimeControl, &g.TimeBonus,
            &g.Result, &g.Winner, pq.Array(&compressedMoves),
&g.CreatedAt,
            &g.WhiteName, &g.BlackName)
        g.MovesLen = len(compressedMoves)
        if err != nil {
            return
        }
        games = append(games, g)
```

```
    }
    return
}
func Insert(id string, g chess.Game) error {
    query := "INSERT INTO game (id, white_id, black_id,\n" +
        "time_control, time_bonus, result, winner, moves)\n" +
        "VALUES ($1, $2, $3, $4, $5, $6, $7, $8);"
    _, err := db.Pool.Exec(query, id, g.WhiteId, g.BlackId,
        g.TimeControl, g.TimeBonus, g.Result, g.Winner,
        pq.Array(compressMoves(g.Moves)))
    return err
}
func compressMoves(moves []chess.CompletedMove) []int {
    compressed := make([]int, len(moves))
    for i, m := range moves {
        compressed[i] = int(m.Move) | (m.TimeLeft << 16)
    }
    return compressed
}
func decompressMoves(moves []int32, fenStr string, control int)
[]chess.CompletedMove {
    g := chess.NewGame(fenStr, control, 0)
    for i, comp := range moves {
        m := bitboard.Move(comp & 0xFFFF)
        if i%2 == 0 {
            g.WhiteTime = int(comp >> 16)
        } else {
            g.BlackTime = int(comp >> 16)
        }
        g.ProcessMove(m)
    }
    return g.Moves
}
```

Б.4 Код пакету player

Лістинг Б.4.1 – Вміст файлу player.go

```
package player
import (
    "encoding/json"
    "log"
    "net/http"
    "github.com/google/uuid"
)
func Mux() *http.ServeMux {
    mux := http.NewServeMux()
    mux.HandleFunc("GET /api/player/id/{id}", getById)
    mux.HandleFunc("GET /api/player/name/{name}", getByName)
    return mux
}
```

```
func getById(rw http.ResponseWriter, r *http.Request) {
    id, err := uuid.Parse(r.PathValue("id"))
    if err != nil {
        rw.WriteHeader(http.StatusBadRequest)
        return
    }
    p, err := SelectPlayerById(id.String())
    if err != nil {
        rw.WriteHeader(http.StatusNotFound)
        return
    }
    rw.Header().Add("Content-Type", "application/json")
    err = json.NewEncoder(rw).Encode(p)
    if err != nil {
        log.Printf("cannot encode response: %v\n", err)
        rw.WriteHeader(http.StatusInternalServerError)
        return
}}
```

Лістинг Б.4.2 – Вміст файлу playerrepository.go

```
package player
import (
    "database/sql"
    "errors"
    "justchess/pkg/db"
    "time"
    "github.com/google/uuid"
)
type Player struct {
    Id           uuid.UUID `json:"id"`
    Mail         string    `json:"-"`
    Name         string    `json:"username"`
    PasswordHash string    `json:"-"`
    IsEngine     bool      `json:"isEngine"`
    CreatedAt    time.Time `json:"createdAt"`
    UpdatedAt    time.Time `json:"-"`
}
type Register struct {
    Mail     string `json:"mail"`
    Name     string `json:"username"`
    Password string `json:"password"`
}
func SelectPlayerById(id string) (p Player, err error) {
    query := "SELECT * FROM player WHERE id = $1;"
    return selectPlayer(query, id)
}
func SelectPlayerByLogin(login string) (p Player, err error) {
    query := "SELECT * FROM player WHERE name = $1 OR mail = $1
AND is_engine = false;"
    return selectPlayer(query, login)
}
func SelectPlayerByMail(mail string) (p Player, err error) {
```

```
    query := "SELECT * FROM player WHERE mail = $1 AND is_engine
= false;"
    return selectPlayer(query, mail)
}
func IsTakenNameOrMail(name, mail string) bool {
    query := "SELECT id FROM player WHERE name = $1 OR mail = $2
AND is_engine = false;"
    rows, err := db.Pool.Query(query, name, mail)
    if err != nil {
        return false
    }
    defer rows.Close()
    return rows.Next()
}
func SelectTokenRegistration(token string) (r Register, err
error) {
    query := "SELECT mail, password_hash, name FROM
tokenregistration WHERE token = $1\n" +
        "AND created_at >= NOW() - INTERVAL '20 minutes';"
    rows, err := db.Pool.Query(query, token)
    if err != nil {
        return
    }
    defer rows.Close()
    if !rows.Next() {
        return r, errors.New("token not found")
    }
    err = rows.Scan(&r.Mail, &r.Password, &r.Name)
    return
}
func SelectTokenReset(token string) (id, passwordHash string,
err error) {
    query := "SELECT player_id, password_hash FROM tokenreset
WHERE token = $1\n" +
        "AND created_at >= NOW() - INTERVAL '20 minutes';"
    rows, err := db.Pool.Query(query, token)
    if err != nil {
        return
    }
    defer rows.Close()
    if !rows.Next() {
        return id, passwordHash, errors.New("token not found")
    }
    err = rows.Scan(&id, &passwordHash)
    return
}
func selectPlayer(query, arg string) (p Player, err error) {
    rows, err := db.Pool.Query(query, arg)
    if err != nil {
        return
    }
    defer rows.Close()
    if !rows.Next() {
```

```
        return p, errors.New("player not found")
    }
    err = rows.Scan(&p.Id, &p.Mail, &p.Name, &p.PasswordHash,
&p.IsEngine,
        &p.CreatedAt, &p.UpdatedAt)
    return
}
func InsertTokenRegistration(token string, r Register, tx
*sql.Tx) error {
    query := "INSERT INTO tokenregistration (token, mail,
password_hash, name)\n" +
        "VALUES ($1, $2, $3, $4);"
    _, err := tx.Exec(query, token, r.Mail, r.Password, r.Name)
    return err
}
func InsertPlayer(id string, r Register) error {
    query := "INSERT INTO player (id, mail, password_hash,
name)\n" +
        "VALUES ($1, $2, $3, $4);"
    _, err := db.Pool.Exec(query, id, r.Mail, r.Password,
r.Name)
    return err
}
func InsertTokenReset(token, userId, hash string, tx *sql.Tx)
error {
    query := "INSERT INTO tokenreset (token, player_id,
password_hash)\n" +
        "VALUES ($1, $2, $3);"
    _, err := tx.Exec(query, token, userId, hash)
    return err
}
func UpdatePasswordHash(hash, id string) (p Player, err error) {
    query := "UPDATE player SET password_hash = $1 WHERE id = $2
RETURNING *;"
    rows, err := db.Pool.Query(query, hash, id)
    if err != nil {
        return
    }
    defer rows.Close()
    if rows.Next() {
        err = rows.Scan(&p.Id, &p.Mail, &p.Name,
&p.PasswordHash, &p.IsEngine,
            &p.CreatedAt, &p.UpdatedAt)
    }
    return }
```

Б.5 Код пакету ws

Лістинг Б.5.1 – Вміст файлу client.go

```
package ws
```

```
import (
    "encoding/json"
    "time"
    "github.com/google/uuid"
    "github.com/gorilla/websocket"
)
const (
    writeWait      = 10 * time.Second
    pongWait       = 60 * time.Second
    pingPeriod     = (pongWait * 9) / 10
    maxMessageSize = 512
)
type client struct {
    id      uuid.UUID
    name    string
    isGuest bool
    hub     *Hub
    room    *Room
    send       chan []byte
    connection *websocket.Conn
}
func newClient(id uuid.UUID, name string, isGuest bool, conn
*websocket.Conn) *client {
    return &client{
        id:         id,
        name:       name,
        isGuest:    isGuest,
        send:       make(chan []byte, 256),
        connection: conn,
    }
}
func (c *client) readRoutine() {
    defer func() { c.cleanup() }()
    c.connection.SetReadLimit(maxMessageSize)
    c.connection.SetReadDeadline(time.Now().Add(pongWait))
    c.connection.SetPongHandler(func(appData string) error {
return c.connection.SetReadDeadline(time.Now().Add(pongWait)) })
    for {
        _, msg, err := c.connection.ReadMessage()
        if err != nil {
            return
        }
        c.handleMessage(msg)
    }
}
func (c *client) writeRoutine() {
    ticker := time.NewTicker(pingPeriod)
    defer func() { ticker.Stop(); c.cleanup() }()
    for {
        select {
        case msg, ok := <-c.send:
c.connection.SetWriteDeadline(time.Now().Add(writeWait))
            if !ok {
```

```
                return
            }
            if err :=
c.connection.WriteMessage(websocket.TextMessage, msg); err !=
nil {
                return
            }
        case <-ticker.C:
c.connection.SetWriteDeadline(time.Now().Add(writeWait))
            if err :=
c.connection.WriteMessage(websocket.PingMessage, nil); err !=
nil {
                return
            }
        }
    }
}
func (c *client) handleMessage(raw []byte) {
    msg := Message{}
    err := json.Unmarshal(raw, &msg)
    if err != nil { return }
    switch msg.Type {
    case CREATE_ROOM:
        data := CreateRoomDTO{}
        err := json.Unmarshal(msg.Data, &data)
        if err != nil || data.TimeControl < 1 || data.TimeBonus
< 0 || c.hub == nil ||
            (!data.IsVSEngine && c.isGuest) {
            return
        }
        r := newRoom(c.hub, c.name, data.IsVSEngine,
data.TimeControl, data.TimeBonus)
        c.hub.add(r)

    case MAKE_MOVE:
        var move MoveDTO
        err = json.Unmarshal(msg.Data, &move)
        if c.room == nil || err != nil {
            return
        }
        c.room.move <- moveEvent{client: c, move: move}
    case CHAT:
        data := ChatDTO{}
        err := json.Unmarshal(msg.Data, &data)
        if err != nil || c.room != nil {
            c.room.chat <- chatEvent{data: data, client: c}
        }
    case RESIGN:
        if c.room != nil {
            c.room.clientEvents <- clientEvent{client: c, eType:
typeResign}
        }
    case DRAW_OFFER:
```

```
        if c.room != nil {
            c.room.clientEvents <- clientEvent{client: c, eType:
typeOfferDraw}
        }
    case DECLINE_DRAW:
        if c.room != nil {
            c.room.clientEvents <- clientEvent{client: c, eType:
typeDeclineDraw}
        }
    }
}
func (c *client) cleanup() {
    c.connection.Close()
    if c.hub != nil { c.hub.unregister(c) }
    if c.room != nil { c.room.unregister(c) }
}
```

Лістинг Б.5.2 – Вміст файлу room.go

```
package ws
import (
    "encoding/json"
    "justchess/pkg/auth"
    "justchess/pkg/chess"
    "justchess/pkg/chess/bitboard"
    "justchess/pkg/chess/enums"
    "justchess/pkg/game"
    "log"
    "math/rand"
    "net/http"
    "github.com/google/uuid"
)
type roomStatus int
type clientEventType int
type moveEvent struct {
    client *client
    move   MoveDTO
}
type clientEvent struct {
    client *client
    eType  clientEventType
}
type chatEvent struct {
    client *client
    data   ChatDTO
}
const (
    statusOpen roomStatus = iota
    statusInProgress
    statusWhiteDisconnected
    statusBlackDisconnected
    statusOver
    typeRegister clientEventType = iota - 5
```

```
    typeUnregister
    typeResign
    typeOfferDraw
    typeDeclineDraw
)
type Room struct {
    Id                uuid.UUID  `json:"id"`
    CreatorName       string     `json:"cn"`
    Status            roomStatus `json:"s"`
    IsVSEngine        bool       `json:"e"`
    pendingDrawIssuer uuid.UUID
    hub               *Hub
    game              *chess.Game
    clients           []*client
    timeout           chan struct{}
    clientEvents      chan clientEvent
    move              chan moveEvent
    chat              chan chatEvent
}
func newRoom(h *Hub, creatorName string, isVSEngine bool,
control, bonus int) *Room {
    id := uuid.New()
    return &Room{
        Id:           id,
        CreatorName:  creatorName,
        Status:       statusOpen,
        IsVSEngine:   isVSEngine,
        hub:          h,
        game:         chess.NewGame("", control, bonus),
        clients:      make([]*client, 0),
        timeout:      make(chan struct{}),
        clientEvents: make(chan clientEvent),
        move:         make(chan moveEvent),
        chat:         make(chan chatEvent),
    }
}
func (r *Room) runRoutine() {
    for {
        select {
        case e := <-r.clientEvents:
            switch e.eType {
            case typeRegister:
                r.register(e.client)
            case typeUnregister:
                r.unregister(e.client)
                if len(r.clients) == 0 {
                    return
                }
            case typeResign:
                r.resign(e.client)
            case typeOfferDraw:
                r.offerDraw(e.client)
            case typeDeclineDraw:
```

```
                r.declineDraw(e.client)
            }
        case me := <-r.move:
            r.handleMove(me)
        case ce := <-r.chat:
            r.broadcastChat(ce.data, ce.client.name)
        case <-r.timeout:
            r.endGame()
        }
    }
}
func (r *Room) HandleNewConnection(rw http.ResponseWriter, req
*http.Request) {
    ctx := req.Context().Value(auth.Cms)
    if ctx == nil {
        rw.WriteHeader(http.StatusUnauthorized)
        return
    }
    cms := ctx.(auth.Claims)
    if !r.IsVSEngine && cms.Role == auth.RoleGuest {
        rw.WriteHeader(http.StatusUnauthorized)
        return
    }
    conn, err := upgrader.Upgrade(rw, req, nil)
    if err != nil {
        log.Printf("%v\n", err)
        return
    }
    c := newClient(cms.Id, cms.Name, cms.Role == auth.RoleGuest,
conn)
    c.room = r
    go c.readRoutine()
    go c.writeRoutine()
    r.clientEvents <- clientEvent{client: c, eType:
typeRegister}
}
func (r *Room) register(c *client) {
    for _, conn := range r.clients {
        if conn.id == c.id || r.Status == statusOver {
            close(c.send)
            return
        }
    }
    r.clients = append(r.clients, c)
    log.Printf("client %s registered\n", c.id.String())

    switch r.Status {
    case statusOpen:
        if r.IsVSEngine || len(r.clients) == 2 {
            r.startGame()
        }
    case statusWhiteDisconnected:
        if r.game.WhiteId == c.id {
```

```
            r.Status = statusInProgress
        }
    case statusBlackDisconnected:
        if r.game.BlackId == c.id {
            r.Status = statusInProgress
        }
    }
    r.broadcastRoomStatus()
}
func (r *Room) unregister(c *client) {
    for i, conn := range r.clients {
        if conn.id == c.id {
            close(c.send)
              r.clients[i] = r.clients[len(r.clients)-1]
            r.clients = r.clients[:len(r.clients)-1]
            log.Printf("client %s unregistered\n",
c.id.String())
            if len(r.clients) == 0 {
                if r.Status != statusOpen && r.Status !=
statusOver {
                    r.game.End <- struct{}{}
                    r.game.SetEndInfo(enums.Unknown, enums.None)
                }
                r.hub.remove(r)
                return
            }
            if c.id == r.game.WhiteId && r.Status != statusOver
                r.Status = statusWhiteDisconnected
            } else if c.id == r.game.BlackId && r.Status !=
statusOver {
                r.Status = statusBlackDisconnected
            }
            r.broadcastRoomStatus()
            break
        }
    }
}
func (r *Room) handleMove(m moveEvent) {
    isPlayerTurn := (r.game.Bitboard.ActiveColor == enums.White
&& r.game.WhiteId == m.client.id) ||
        (r.game.Bitboard.ActiveColor == enums.Black &&
r.game.BlackId == m.client.id)
    if r.Status == statusOpen || r.Status == statusOver ||
        (r.game.WhiteId != m.client.id && r.game.BlackId !=
m.client.id) ||
        (!r.IsVSEngine && !isPlayerTurn) {
        return
    }
    res := make(chan bool)
    r.game.Move <- chess.MoveEvent{
        ClientId: m.client.id,
        Move:     bitboard.NewMove(m.move.Destination,
m.move.Source, m.move.Type),
```

```
        Response: res,
    }
    if !<-res { return }
    lastMove := r.game.Moves[len(r.game.Moves)-1]
    data, err := json.Marshal(Message{Type: LAST_MOVE, Data:
r.serializeLastMove(lastMove, r.game.Bitboard.LegalMoves)})
    if err != nil {
        log.Printf("cannot Marshal data: %v\n", err)
        return
    }
    r.broadcast(data)
    if r.game.Result != enums.Unknown {
        r.endGame()
    }
}
func (r *Room) resign(c *client) {
    if r.Status == statusOpen || r.Status == statusOver {
        return
    }
    if c.id == r.game.WhiteId {
        r.game.End <- struct{}{}
        r.game.SetEndInfo(enums.Resignation, enums.Black)
        r.endGame()
    } else if c.id == r.game.BlackId {
        r.game.End <- struct{}{}
        r.game.SetEndInfo(enums.Resignation, enums.White)
        r.endGame()
    }
}
func (r *Room) offerDraw(c *client) {
    if r.Status == statusOpen || r.Status == statusOver {
        return
    }
    if r.pendingDrawIssuer == uuid.Nil {
        var oppId uuid.UUID
        var msg []byte
        if c.id == r.game.WhiteId {
            oppId = r.game.BlackId
            msg, _ = json.Marshal(Message{Type: CHAT, Data:
[]byte(`"White offers draw"`)})
        } else if c.id == r.game.BlackId {
            oppId = r.game.WhiteId
            msg, _ = json.Marshal(Message{Type: CHAT, Data:
[]byte(`"Black offers draw"`)})
        } else {
            return
        }
        r.broadcast(msg)
        r.pendingDrawIssuer = c.id
        msg, _ = json.Marshal(Message{Type: DRAW_OFFER, Data:
nil})
        for _, conn := range r.clients {
            if conn.id == oppId {
```

```
                conn.send <- msg
            }
        }
    } else if r.pendingDrawIssuer != c.id && (c.id ==
r.game.WhiteId ||
        c.id == r.game.BlackId) {
        msg, _ := json.Marshal(Message{Type: CHAT, Data:
[]byte(`"Draw accepted"`)})
        r.broadcast(msg)
        r.game.End <- struct{}{}
        r.game.SetEndInfo(enums.Agreement, enums.None)
        r.endGame()
    }
}
func (r *Room) declineDraw(c *client) {
    if r.Status == statusOpen || r.Status == statusOver ||
        c.id == r.pendingDrawIssuer || r.pendingDrawIssuer ==
uuid.Nil ||
        (c.id != r.game.WhiteId && c.id != r.game.BlackId) {
        return
    }
    msg, _ := json.Marshal(Message{Type: CHAT, Data:
[]byte(`"Draw declined"`)})
    r.broadcast(msg)
    r.pendingDrawIssuer = uuid.Nil
}
func (r *Room) broadcastChat(data ChatDTO, senderName string) {
    data.Message = `"` + senderName + `: ` + data.Message + `"`
    msg, err := json.Marshal(Message{Type: CHAT, Data:
[]byte(data.Message)})
    if err != nil { return }
    r.broadcast(msg)
}
func (r *Room) startGame() {
    r.Status = statusInProgress
    go r.game.RunRoutine(r.timeout)
    if rand.Intn(2) == 1 {
        r.game.WhiteId = r.clients[0].id
        if len(r.clients) == 2 {
            r.game.BlackId = r.clients[1].id
        } else {
            r.game.BlackId = uuid.MustParse("ccaf962b-855e-49da-
b85f-7e8bba0edae2")
        }
    } else {
        r.game.BlackId = r.clients[0].id
        if len(r.clients) == 2 {
            r.game.WhiteId = r.clients[1].id
        } else {
            r.game.WhiteId = uuid.MustParse("ccaf962b-855e-49da-
b85f-7e8bba0edae2")
        }
    }
```

```
}
func (r *Room) endGame() {
    r.Status = statusOver
    data, _ := json.Marshal(GameResultDTO{Result: r.game.Result,
Winner: r.game.Winner})
    msg, _ := json.Marshal(Message{Type: GAME_RESULT, Data:
data})
    r.broadcast(msg)
    r.hub.remove(r)
    if len(r.game.Moves) > 1 {
        err := game.Insert(r.Id.String(), *r.game)
        if err != nil {
            log.Printf("cannot store game in the db: %v, game:
%v\n", err, *r.game)
        }
    }
}
func (r *Room) broadcast(msg []byte) {
    for _, c := range r.clients {
        c.send <- msg
    }
}
func (r *Room) broadcastRoomStatus() {
    info := chess.GameInfo{}
    if r.Status == statusOpen || r.Status == statusOver {
        info = chess.GameInfo{
            WhiteTime:  r.game.WhiteTime,
            BlackTime:  r.game.BlackTime,
            Result:     r.game.Result,
            Winner:     r.game.Winner,
            Moves:      r.game.Moves,
            LegalMoves: r.game.Bitboard.LegalMoves,
        }
    } else {
        res := make(chan chess.GameInfo)
        r.game.Info <- chess.GameInfoEvent{Response: res}
        info = <-res
    }
    data, err := json.Marshal(Message{Type: ROOM_STATUS, Data:
r.serializeRoomStatus(info)})
    if err != nil {
        log.Printf("cannot Marshal data: %v\n", err)
        return
    }
    r.broadcast(data)
}
func (r *Room) serializeRoomStatus(info chess.GameInfo) []byte {
    data, err := json.Marshal(RoomStatusDTO{
        Status:     r.Status,
        White:      r.game.WhiteId,
        Black:      r.game.BlackId,
        WhiteTime:  info.WhiteTime,
        BlackTime:  info.BlackTime,
```

```
        Control:    r.game.TimeControl,
        IsVSEngine: r.IsVSEngine,
        Clients:    len(r.clients),
    })
    if err != nil {
        log.Printf("cannot Marshal room status: %v\n", err)
    }
    return data
}
func (r *Room) serializeLastMove(move chess.CompletedMove, lm
[]bitboard.Move) []byte {
    data, err := json.Marshal(LastMoveDTO{
        SAN:        move.SAN,
        FEN:        move.FEN,
        TimeLeft:   move.TimeLeft,
        LegalMoves: lm,
    })
    if err != nil {
        log.Printf("cannot Marshal last move: %v\n", err)
    }
    return data
}
```

Лістинг Б.5.3 – Вміст файлу hub.go

```
package ws
import (
    "encoding/json"
    "justchess/pkg/auth"
    "log"
    "net/http"
    "os"
    "sync"
    "github.com/google/uuid"
    "github.com/gorilla/websocket"
)
var upgrader = websocket.Upgrader{
    WriteBufferSize: 1024,
    ReadBufferSize:  1024,
    CheckOrigin:     func(r *http.Request) bool { return
r.Header.Get("Origin") == os.Getenv("DOMAIN") },
}
type Hub struct {
    sync.Mutex
    rooms   map[*Room]struct{}
    clients map[*client]struct{}
}
func NewHub() *Hub {
    return &Hub{
        rooms:   make(map[*Room]struct{}),
        clients: make(map[*client]struct{}),
    }
}
```

```
func (h *Hub) HandleNewConnection(rw http.ResponseWriter, r
*http.Request) {
    ctx := r.Context().Value(auth.Cms)
    if ctx == nil {
        log.Println("request with nil context")
        rw.WriteHeader(http.StatusUnauthorized)
        return
    }
    cms := ctx.(auth.Claims)
    conn, err := upgrader.Upgrade(rw, r, nil)
    if err != nil {
        log.Printf("request from: %s\n", r.Header.Get("Origin"))
        log.Printf("%v\n", err)
        return
    }
    c := newClient(cms.Id, cms.Name, cms.Role == auth.RoleGuest,
conn)
    c.hub = h
    go c.readRoutine()
    go c.writeRoutine()
    h.register(c)
}
func (h *Hub) register(c *client) {
    h.Lock()
    defer h.Unlock()
    for connected := range h.clients {
        if connected.id == c.id {
            return
        }
    }
    h.clients[c] = struct{}{}
    log.Printf("client %s registered\n", c.id.String())
    h.broadcastClientsCounter()
    h.send10Rooms(c)
}
func (h *Hub) unregister(c *client) {
    h.Lock()
    defer h.Unlock()
    if _, ok := h.clients[c]; !ok {
        return
    }
    delete(h.clients, c)
    log.Printf("client %s unregistered\n", c.id.String())
    h.broadcastClientsCounter()
}
func (h *Hub) add(r *Room) {
    h.Lock()
    defer h.Unlock()
    go r.runRoutine()
    h.rooms[r] = struct{}{}
    log.Printf("room %s added\n", r.Id.String())
    h.broadcastAddRoom(r)
}
```

```
func (h *Hub) remove(r *Room) {
    h.Lock()
    defer h.Unlock()
    if _, ok := h.rooms[r]; !ok {
        return
    }
    delete(h.rooms, r)
    log.Printf("room %s removed\n", r.Id.String())
    h.broadcastRemoveRoom(r.Id)
}
func (h *Hub) broadcastClientsCounter() {
    data, err := json.Marshal(ClientsCounterDTO{Counter:
len(h.clients)})
    if err != nil {
        log.Printf("cannot Marshal message: %v\n", err)
        return
    }
    msg, _ := json.Marshal(Message{Type: CLIENTS_COUNTER, Data:
data})
    for c := range h.clients {
        c.send <- msg
    }
}
func (h *Hub) broadcastAddRoom(r *Room) {
    data, err := json.Marshal(AddRoomDTO{
        Id:          r.Id,
        Creator:     r.CreatorName,
        TimeControl: r.game.TimeControl,
        TimeBonus:   r.game.TimeBonus,
    })
    if err != nil {
        log.Printf("cannot Marshal message: %v\n", err)
        return
    }
    msg, _ := json.Marshal(Message{Type: ADD_ROOM, Data: data})
    for c := range h.clients {
        c.send <- msg
    }
}
func (h *Hub) broadcastRemoveRoom(roomId uuid.UUID) {
    data, err := json.Marshal(RemoveRoomDTO{
        RoomId: roomId.String(),
    })
    if err != nil {
        log.Printf("cannot Marshal message: %v\n", err)
        return
    }
    msg, _ := json.Marshal(Message{Type: REMOVE_ROOM, Data:
data})
    for c := range h.clients {
        c.send <- msg
    }
}
```

```
func (h *Hub) send10Rooms(c *client) {
    cnt := 0
    for r := range h.rooms {
        cnt++
        if cnt == 10 {
            return
        }
        data, err := json.Marshal(AddRoomDTO{
            Id:          r.Id,
            Creator:     r.CreatorName,
            TimeBonus:   r.game.TimeBonus,
            TimeControl: r.game.TimeControl,
        })
        if err != nil {
            log.Printf("cannot Marshal message: %v\n", err)
            return
        }
        msg, _ := json.Marshal(Message{Type: ADD_ROOM, Data:
data})
        c.send <- msg
    }
}
func (h *Hub) GetRoomById(id uuid.UUID) *Room {
    h.Lock()
    defer h.Unlock()
    for r := range h.rooms {
        if r.Id == id { return r }
    }
    return nil
}
```