CARD GAME (ΞΕΡΉ)_2025

Μάθημα: Ανάπτυξη Διαδικτυακών Συστημάτων και Εφαρμογών

Ομάδα:

Γιάννης Καλογριάς (185434)

Γούλας Γεώργιος (185372)

Ενότητα 1. Εισαγωγή

Η παρούσα εργασία αφορά την ανάπτυξη ενός διαδικτυακού multiplayer παιχνιδιού καρτών με τίτλο «Ξερή», βασισμένου στο ομώνυμο παραδοσιακό ελληνικό παιχνίδι. Η υλοποίηση ακολουθεί αρχιτεκτονική client–server, με backend σε PHP και βάση δεδομένων MySQL/MariaDB, ενώ το frontend είναι browser-based και επικοινωνεί με τον server μέσω RESTful API με ανταλλαγή δεδομένων σε μορφή JSON. Στόχος του project είναι η ορθή υλοποίηση της λογικής του παιχνιδιού, η διαχείριση της σειράς παιξίματος και ο αξιόπιστος συγχρονισμός της κατάστασης μεταξύ δύο παικτών σε κατανεμημένο περιβάλλον.

Ενότητα 2. Γενική Αρχιτεκτονική Συστήματος

Το σύστημα διαχωρίζεται σε τρία βασικά επίπεδα. Το επίπεδο παρουσίασης υλοποιείται μέσω του αρχείου index.html και παρέχει το γραφικό περιβάλλον του παιχνιδιού, επιτρέποντας στον χρήστη να δημιουργεί ή να συμμετέχει σε παιχνίδια και να αλληλεπιδρά με αυτά. Το επίπεδο επιχειρησιακής λογικής υλοποιείται στο backend μέσω επιμέρους PHP αρχείων, καθένα από τα οποία εξυπηρετεί συγκεκριμένη λειτουργία του παιχνιδιού. Το επίπεδο αποθήκευσης δεδομένων υλοποιείται μέσω σχεσιακής βάσης δεδομένων, στην οποία αποθηκεύεται πλήρως η κατάσταση κάθε παιχνιδιού, εξασφαλίζοντας συνέπεια και ανθεκτικότητα σε ταυτόχρονες προσβάσεις.

Ενότητα 3. Δημιουργία Παιχνιδιού

Η ροή του παιχνιδιού ξεκινά με τη δημιουργία νέου παιχνιδιού από τον πρώτο παίκτη. Η διαδικασία αυτή υλοποιείται στο αρχείο game_init.php, το οποίο δέχεται αίτημα POST με το αναγνωριστικό του χρήστη. Αρχικά πραγματοποιείται σύνδεση με τη βάση δεδομένων μέσω της συνάρτησης getDB(), η οποία ορίζεται στο αρχείο db.php. Στη συνέχεια καλείται η

συνάρτηση createDeck() από το αρχείο deck.php, η οποία δημιουργεί μία πλήρη τράπουλα 52 φύλλων. Η τράπουλα ανακατεύεται και τέσσερα φύλλα αφαιρούνται από αυτήν ώστε να τοποθετηθούν στο τραπέζι. Ακολούθως δημιουργείται νέα εγγραφή στον πίνακα games με κατάσταση waiting, ενώ ο πρώτος παίκτης καταχωρείται στον πίνακα game_players με κενό χέρι και αρχικό σκορ μηδέν. Όλες οι ενέργειες εκτελούνται εντός συναλλαγής, ώστε να διασφαλίζεται η ακεραιότητα των δεδομένων.

Ενότητα 4. Συμμετοχή Δεύτερου Παίκτη και Έναρξη Παιχνιδιού

Η συμμετοχή του δεύτερου παίκτη πραγματοποιείται μέσω του αρχείου game_join.php. Το σύστημα ελέγχει ότι το παιχνίδι βρίσκεται σε κατάσταση αναμονής και ότι υπάρχει ήδη ένας παίκτης συνδεδεμένος. Στη συνέχεια ο δεύτερος παίκτης προστίθεται στον πίνακα game_players και πραγματοποιείται το αρχικό μοίρασμα τεσσάρων φύλλων σε κάθε παίκτη, αφαιρώντας τα αντίστοιχα φύλλα από την τράπουλα. Ο πρώτος παίκτης που θα ξεκινήσει το παιχνίδι ορίζεται βάσει της χρονικής στιγμής εισόδου, χρησιμοποιώντας το πεδίο joined_at. Τέλος, η κατάσταση του παιχνιδιού ενημερώνεται σε active και η νέα κατάσταση της τράπουλας αποθηκεύεται στη βάση δεδομένων.

Ενότητα 5. Συγχρονισμός και Παρακολούθηση Κατάστασης Παιχνιδιού

Κατά τη διάρκεια του παιχνιδιού, κάθε client πραγματοποιεί περιοδικά αιτήματα προς τον server για την ανάκτηση της τρέχουσας κατάστασης. Η λειτουργία αυτή υλοποιείται στο αρχείο game_state.php, το οποίο λαμβάνει ως παραμέτρους το game_id και το user_id. Το αρχείο ανακτά τα δεδομένα του παιχνιδιού από τον πίνακα games, το χέρι και το σκορ του παίκτη από τον πίνακα game_players, 1 καθώς και το σκορ του αντιπάλου. Στη συνέχεια επιστρέφει στον client όλες τις απαραίτητες πληροφορίες για την ενημέρωση του γραφικού περιβάλλοντος. Η επιλογή του μηχανισμού polling κρίνεται επαρκής, καθώς το παιχνίδι είναι turn-based και δεν απαιτεί αυστηρό συγχρονισμό πραγματικού χρόνου.

Ενότητα 6. Εκτέλεση Κίνησης Παίκτη

Η εκτέλεση μιας κίνησης υλοποιείται στο αρχείο game_move.php. Όταν ένας παίκτης επιλέξει να παίξει ένα φύλλο, το σύστημα ελέγχει αρχικά ότι το παιχνίδι βρίσκεται σε ενεργή κατάσταση και ότι είναι η σειρά του συγκεκριμένου παίκτη. Στη συνέχεια ανακτάται το χέρι του παίκτη και ελέγχεται η εγκυρότητα του φύλλου. Το φύλλο αφαιρείται από το χέρι και πραγματοποιείται έλεγχος πιασίματος μέσω της συνάρτησης isCapture(), η οποία ορίζεται στο αρχείο game_rules.php. Σε περίπτωση πιασίματος,

υπολογίζονται οι πόντοι μέσω της συνάρτησης calculatePoints(), ενώ ελέγχεται και η περίπτωση Ξερής με τη συνάρτηση isXeri(), αποδίδοντας το αντίστοιχο bonus. Σε διαφορετική περίπτωση, το φύλλο προστίθεται στο τραπέζι. Στο τέλος της διαδικασίας ενημερώνεται το σκορ του παίκτη, το χέρι του και ο επόμενος παίκτης που έχει τη σειρά.

Ενότητα 7. Διαχείριση Γύρων και Ολοκλήρωση Παιχνιδιού

Το σύστημα υποστηρίζει αυτόματο μοίρασμα νέων φύλλων όταν και οι δύο παίκτες έχουν αδειάσει τα χέρια τους και υπάρχουν διαθέσιμα φύλλα στην τράπουλα. Όταν η τράπουλα αδειάσει πλήρως και δεν υπάρχουν άλλα φύλλα στα χέρια των παικτών, το παιχνίδι ολοκληρώνεται. Τα εναπομείναντα φύλλα του τραπεζιού αποδίδονται στον παίκτη που πραγματοποίησε το τελευταίο πιάσιμο και η κατάσταση του παιχνιδιού ενημερώνεται σε finished.

Ενότητα 8. Λειτουργία Frontend και Αλληλεπίδραση με τον Χρήστη

Η διεπαφή του χρήστη υλοποιείται μέσω ενός ενιαίου αρχείου HTML με ενσωματωμένο CSS και JavaScript. Ο ρόλος του frontend είναι η απεικόνιση της κατάστασης του παιχνιδιού και η αποστολή ενεργειών του χρήστη προς το backend μέσω HTTP αιτημάτων. Η επικοινωνία με τον server υλοποιείται μέσω της γενικής συνάρτησης api(url, data) , η οποία χρησιμοποιεί τη μέθοδο fetch για την αποστολή POST αιτημάτων με δεδομένα σε μορφή JSON:

```
function api(url, data) {
    return fetch(url, {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(data)
    }).then(r => r.json());
}
```

Η συγκεκριμένη συνάρτηση χρησιμοποιείται από όλες τις ενέργειες του παιχνιδιού, εξασφαλίζοντας ενιαίο και επαναχρησιμοποιήσιμο τρόπο επικοινωνίας με το backend.

8.1 Δημιουργία Παιχνιδιού από τον Πρώτο Παίκτη

Η δημιουργία νέου παιχνιδιού πραγματοποιείται μέσω της συνάρτησης createGame() . Ο χρήστης εισάγει το αναγνωριστικό του και καλείται το αρχείο game_init.php . Σε περίπτωση επιτυχίας, αποθηκεύεται το game_id ,

ενημερώνεται η κατάσταση του παιχνιδιού και ξεκινά ο μηχανισμός συγχρονισμού:

```
function createGame() {
    currentUserId = +userId.value;

    api('api/game_init.php', { user_id: currentUserId })
        .then(res => {
            if (res.game_id) {
                currentGameId = res.game_id;
                gameId.value = res.game_id;

                status.innerText = 'Waiting for second player...';
                startPolling();
            }
        });
}
```

Η κλήση της startPolling() εξασφαλίζει ότι ο client θα ελέγχει περιοδικά την κατάσταση του παιχνιδιού μέχρι την είσοδο του δεύτερου παίκτη.

8.2 Συμμετοχή Δεύτερου Παίκτη

Η είσοδος δεύτερου παίκτη υλοποιείται μέσω της συνάρτησης user_id και το joinGame() , η οποία αποστέλλει το game_id στο backend. Μετά την επιτυχή συμμετοχή, ξεκινά και πάλι ο συγχρονισμός της κατάστασης:

```
function joinGame() {
    currentUserId = +userId.value;
    currentGameId = +gameId.value;

    api('api/game_join.php', {
        user_id: currentUserId,
        game_id: currentGameId
    }).then(res => {
        if (res.success) {
            startPolling();
        }
    });
}
```

8.3 Ανάκτηση και Συγχρονισμός Κατάστασης Παιχνιδιού

Η τρέχουσα κατάσταση του παιχνιδιού ανακτάται περιοδικά μέσω της συνάρτησης loadState(), η οποία πραγματοποιεί αίτημα GET προς το endpoint game_state.php, περνώντας ως παραμέτρους το game_id και το user_id. Η απάντηση επιστρέφεται σε μορφή JSON και, εφόσον το πεδίο success είναι true, το frontend ενημερώνει ανάλογα το περιβάλλον χρήστη. Αν το παιχνίδι βρίσκεται σε κατάσταση waiting, εμφανίζεται μήνυμα αναμονής δεύτερου παίκτη. Διαφορετικά, όταν το παιχνίδι έχει ξεκινήσει, καλείται η updateUI(data) ώστε να ενημερωθούν τα φύλλα του τραπεζιού, το χέρι του παίκτη και το σκορ.

```
function loadState() {
    fetch(`api/game_state.php?game_id=${currentGameId}&user_id=${currentUserId}`)
        .then(r => r.json())
        .then(data => {
            if (!data.success) return;

            if (data.status === 'waiting') {
                status.innerText = 'Waiting for second player...';
                return;
            }

            updateUI(data);
        });
}
```

8.4 Εκτέλεση Κίνησης Παίκτη

Η εκτέλεση κίνησης παίκτη πραγματοποιείται μέσω της συνάρτησης playCard(card), η οποία ενεργοποιείται όταν ο χρήστης επιλέξει ένα φύλλο από το χέρι του. Η συνάρτηση ανακτά τα userId και gameId από τα αντίστοιχα πεδία της σελίδας και στέλνει POST αίτημα στο endpoint game_move.php, μαζί με το φύλλο που παίχτηκε. Αν ο server επιστρέψει σφάλμα (π.χ. δεν είναι η σειρά του παίκτη ή το φύλλο δεν υπάρχει στο χέρι), εμφανίζεται μήνυμα στον χρήστη. Αν επιστραφεί ένδειξη λήξης παιχνιδιού (game_over), εμφανίζεται αντίστοιχο μήνυμα. Σε κάθε άλλη περίπτωση, το UI ενημερώνεται άμεσα με τα νέα δεδομένα μέσω της updateUI(res).

```
function playCard(card) {
    const userId = +document.getElementById('userId').value;
    const gameId = +document.getElementById('gameId').value;

    api('api/game_move.php', {
        game_id: gameId,
        user_id: userId,
        card: card
    }).then(res => {
        if (res.error) {
            alert(res.error);
        } else if (res.game_over) {
            alert("🏁 Game Over!");
        } else {
            updateUI(res);
        }
    });
}
```

Ο μηχανισμός αυτός βασίζεται σε polling (μέσω της περιοδικής κλήσης της loadState()), το οποίο είναι επαρκές για turn-based παιχνίδι, καθώς οι ενέργειες γίνονται διαδοχικά και η κατάσταση χρειάζεται συχνή, αλλά όχι «πραγματικού χρόνου» ενημέρωση όπως σε παιχνίδια άμεσης δράσης.

8.5 Ενημέρωση Γραφικού Περιβάλλοντος

Η ενημέρωση της διεπαφής πραγματοποιείται μέσω της συνάρτησης updateUI(data) , η οποίαανανεώνει το τραπέζι, το χέρι του παίκτη, το σκορ και την ένδειξη σειράς:

```
function updateUI(data) {
    renderCards('table', data.table, false);
    renderCards('hand', data.hand, data.your_turn);

    score.innerText = data.your_score;
    status.innerText = data.your_turn
        ? '🟢 Your turn'
        : '⏳ Opponent turn';
}
```

Η απόδοση των φύλλων υλοποιείται μέσω της συνάρτησης renderCards() , η οποία δημιουργεί δυναμικά HTML στοιχεία και ενεργοποιεί τη δυνατότητα επιλογής μόνο όταν είναι η σειρά του παίκτη.

Ενότητα 9. Δημιουργία Νέου Παιχνιδιού (game_init.php)

Το αρχείο game_init.php αποτελεί το σημείο εκκίνησης κάθε νέου παιχνιδιού. Είναι υπεύθυνο για τη δημιουργία της αρχικής κατάστασης του παιχνιδιού, την παραγωγή και ανακάτεμα της τράπουλας, καθώς και την καταχώρηση του πρώτου παίκτη στη βάση δεδομένων. Η λειτουργία του εκτελείται μέσω HTTP POST αιτήματος και επιστρέφει δεδομένα σε μορφή JSON.

9.1 Λήψη και Έλεγχος Δεδομένων Αιτήματος

Το σύστημα αναμένει ως είσοδο το user_id του παίκτη που επιθυμεί να δημιουργήσει νέο παιχνίδι. Σε περίπτωση που το πεδίο αυτό απουσιάζει, η εκτέλεση διακόπτεται άμεσα και επιστρέφεται μήνυμα σφάλματος. Με τον τρόπο αυτό διασφαλίζεται η εγκυρότητα του αιτήματος.

```
$data = json_decode(file_get_contents("php://input"), true);

if (!isset($data['user_id'])) {
    echo json_encode(["error" => "Invalid request"]);
    exit;
}

$userId = (int)$data['user_id'];
```

Το σύστημα αναμένει ως είσοδο το user_id του παίκτη που επιθυμεί να δημιουργήσει νέο παιχνίδι. Σε περίπτωση που το πεδίο αυτό απουσιάζει, η εκτέλεση διακόπτεται άμεσα και επιστρέφεται μήνυμα σφάλματος. Με τον τρόπο αυτό διασφαλίζεται η εγκυρότητα του αιτήματος.

9.2 Έναρξη Συναλλαγής Βάσης Δεδομένων

$db->beginTransaction();

Η χρήση συναλλαγής (transaction) διασφαλίζει ότι όλες οι ενέργειες που ακολουθούν θα εκτελεστούν ατομικά. Σε περίπτωση αποτυχίας σε οποιοδήποτε στάδιο, η βάση δεδομένων επανέρχεται στην προηγούμενη συνεπή κατάστασή της.

9.3 Δημιουργία Παιχνιδιού στο Backend

Αρχικά, ορίζεται ο τύπος περιεχομένου της απάντησης σε JSON και φορτώνονται τα απαραίτητα αρχεία για τη σύνδεση με τη βάση δεδομένων και τη διαχείριση της τράπουλας:

```php
header('Content-Type: application/json');

require_once 'db.php';
require_once 'deck.php';

$db = getDB();
```

Η είσοδος δεδομένων πραγματοποιείται μέσω JSON payload, το οποίο αποκωδικοποιείται και ελέγχεται ως προς την εγκυρότητά του. Αν δεν υπάρχει το user_id , το αίτημα απορρίπτεται:

```php
$data = json_decode(file_get_contents("php://input"), true);

if (!isset($data['user_id'])) {
    echo json_encode(["error" => "Invalid request"]);
    exit;
}

$userId = (int)$data['user_id'];
```

Στη συνέχεια ξεκινά συναλλαγή βάσης δεδομένων, ώστε όλες οι ενέργειες που ακολουθούν να εκτελεστούν ατομικά και με ασφάλεια:

$db->beginTransaction()

9.4 Δημιουργία και Αρχικοποίηση Τράπουλας

Η τράπουλα δημιουργείται μέσω της συνάρτησης createDeck() και ανακατεύεται με τη shuffleDeck() . Τα πρώτα τέσσερα φύλλα αφαιρούνται από την τράπουλα και τοποθετούνται στο τραπέζι, σύμφωνα με τους κανόνες του παιχνιδιού Ξερή: Η χρήση της array_splice() διασφαλίζει ότι τα φύλλα αφαιρούνται οριστικά από την τράπουλα και δεν μπορούν να ξαναμοιραστούν.

```php
/* 1️⃣ Δημιουργία τράπουλας */
$deck = createDeck();
shuffleDeck($deck);

/* 2️⃣ 4 φύλλα στο τραπέζι */
$tableCards = array_splice($deck, 0, 4);
```

9.5 Δημιουργία Εγγραφής Παιχνιδιού

Ακολούθως δημιουργείται νέα εγγραφή στον πίνακα games . Η κατάσταση του παιχνιδιού ορίζεται σε waiting , ενώ αποθηκεύονται σε μορφή JSON τόσο η εναπομείνασα τράπουλα όσο και τα φύλλα του τραπεζιού:

```php
/* 3️⃣ Δημιουργία παιχνιδιού */
$stmt = $db->prepare("
    INSERT INTO games (status, current_player, deck, table_cards)
    VALUES ('waiting', NULL, ?, ?)
");
$stmt->execute([
    json_encode($deck),
    json_encode($tableCards)
]);

$gameId = $db->lastInsertId();
```

Η αποθήκευση της κατάστασης σε JSON επιτρέπει την εύκολη ανάκτηση και ενημέρωση του παιχνιδιού χωρίς πολύπλοκες συσχετίσεις πινάκων.

9.6 Εισαγωγή Πρώτου Παίκτη

Ο πρώτος παίκτης καταχωρείται στον πίνακα game_players με αρχικό κενό χέρι και μηδενικό σκορ. Το χέρι αποθηκεύεται επίσης σε μορφή JSON:

```php
/* 4️⃣ Εισαγωγή πρώτου παίκτη */
$stmt = $db->prepare("
    INSERT INTO game_players (game_id, user_id, hand, score)
    VALUES (?, ?, '[]', 0)
");
$stmt->execute([$gameId, $userId]);
```

Με τον τρόπο αυτό διασφαλίζεται ότι το παιχνίδι μπορεί να ξεκινήσει άμεσα μόλις συνδεθεί και ο δεύτερος παίκτης.

9.7 Ολοκλήρωση Συναλλαγής και Απόκριση

Μετά την επιτυχή ολοκλήρωση όλων των ενεργειών, η συναλλαγή επικυρώνεται και επιστρέφεται στο frontend η απάντηση με το αναγνωριστικό του παιχνιδιού:Σε περίπτωση σφάλματος, η συναλλαγή αναιρείται και επιστρέφεται μήνυμα λάθους, εξασφαλίζοντας συνέπεια στη βάση δεδομένων. Σε περίπτωση σφάλματος, η συναλλαγή αναιρείται και επιστρέφεται μήνυμα λάθους, εξασφαλίζοντας συνέπεια στη βάση δεδομένων:

```php
    $db->commit();

    echo json_encode([
        "success" => true,
        "game_id" => $gameId,
        "status" => "waiting"
    ]);

} catch (Exception $e) {
    $db->rollBack();
    echo json_encode(["error" => $e->getMessage()]);
}
```

10. Διαχείριση Κίνησης Παίκτη και Λογική Παιχνιδιού (game_move.php)

Το αρχείο game_move.php αποτελεί τον πυρήνα της λειτουργικότητας του παιχνιδιού, καθώς διαχειρίζεται κάθε κίνηση παίκτη, την εφαρμογή των κανόνων της Ξερής, τον υπολογισμό πόντων, την εναλλαγή σειράς και τον έλεγχο ολοκλήρωσης του παιχνιδιού. Η εκτέλεσή του γίνεται μέσω HTTP POST αιτήματος και προστατεύεται πλήρως μέσω συναλλαγών βάσης δεδομένων.

10.1 Λήψη και Έλεγχος Εγκυρότητας Αιτήματος

Το σύστημα απαιτεί το αναγνωριστικό παιχνιδιού, το αναγνωριστικό παίκτη και το φύλλο που παίζεται. Ο έλεγχος αυτός αποτρέπει ελλιπή ή κακόβουλα αιτήματα πριν την επεξεργασία.

```php
$data = json_decode(file_get_contents("php://input"), true);

if (!isset($data['game_id'], $data['user_id'], $data['card'])) {
    echo json_encode(["error" => "Invalid request"]);
    exit;
}
```

10.2 Έλεγχος Κατάστασης Παιχνιδιού και Σειράς

```php
    /* 🔲 Φόρτωση παιχνιδιού */
    $stmt = $db->prepare("SELECT * FROM games WHERE id = ?");
    $stmt->execute([$gameId]);
    $game = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$game || $game['status'] !== 'active') {
        throw new Exception("Game not active");
    }

    if ((int)$game['current_player'] !== $userId) {
        throw new Exception("Not your turn");
    }
```

Σε αυτό το στάδιο διασφαλίζεται ότι το παιχνίδι βρίσκεται σε εξέλιξη και ότι ο παίκτης που επιχειρεί να παίξει έχει πράγματι τη σειρά του. Οποιαδήποτε απόκλιση οδηγεί σε άμεση απόρριψη της κίνησης.

10.3 Φόρτωση Παίκτη και Έλεγχος Φύλλου

```php
$hand = json_decode($player['hand'], true);
$score = (int)$player['score'];

/* Έλεγχος ότι το φύλλο υπάρχει στο χέρι */
$cardIndex = -1;
foreach ($hand as $i => $c) {
    if ($c['suit'] === $playedCard['suit'] && $c['value'] === $playedCard['value']) {
        $cardIndex = $i;
        break;
    }
}
```

Ο έλεγχος αυτός επιβεβαιώνει ότι το φύλλο που επιχειρεί να παίξει ο χρήστης υπάρχει στο χέρι του. Με αυτόν τον τρόπο αποτρέπεται οποιαδήποτε μη έγκυρη ή αλλοιωμένη κίνηση από την πλευρά του client.

10.4 Εφαρμογή Κανόνων Πιασίματος και Ξερής

```php
/* 🃏 Κανόνες πιασίματος */
if (isCapture($playedCard, $tableCards)) {

    $captured = $tableCards;
    $captured[] = $playedCard;

    $points = calculatePoints($captured);
    $score += $points;

    if (isXeri($tableCards, $playedCard)) {
    $score += 10;
}

    $tableCards = [];
```

Η λογική πιασίματος βασίζεται στις συναρτήσεις αρχείο isCapture() και isXeri() που ορίζονται στο game_rules.php . Αν το φύλλο πιάνει τη στοίβα, υπολογίζονται οι πόντοι, προστίθεται το μπόνους της Ξερής (εφόσον ισχύει) και το τραπέζι αδειάζει

10.5 Ενημέρωση Κατάστασης Παίκτη

```php
/* 🔄 Ενημέρωση παίκτη */
$stmt = $db->prepare("
    UPDATE game_players
    SET hand = ?, score = ?
    WHERE id = ?
");
$stmt->execute([
    json_encode($hand),
    $score,
    $player['id']
]);
```

Μετά την ολοκλήρωση της κίνησης, ενημερώνεται το χέρι και το σκορ του παίκτη στη βάση δεδομένων, διασφαλίζοντας συνέπεια μεταξύ server και client.

10.6 Εναλλαγή Σειράς και Μοίρασμα Νέων Φύλλων

```php
if ($allEmpty && count($deck) >= 8) {
    foreach ($players as $p) {
        $newHand = array_splice($deck, 0, 4);
        $stmt = $db->prepare("UPDATE game_players SET hand = ? WHERE id = ?");
        $stmt->execute([json_encode($newHand), $p['id']]);
    }
}
```

Όταν και οι δύο παίκτες αδειάσουν τα χέρια τους και υπάρχουν διαθέσιμα φύλλα στην τράπουλα, το σύστημα μοιράζει εκ νέου τέσσερα φύλλα στον καθένα, ακολουθώντας τους κανόνες του παιχνιδιού.

10.7 Έλεγχος Τερματισμού Παιχνιδιού

```php
/* 🏁 Τέλος παιχνιδιού */
if (count($deck) === 0) {
    $stmt = $db->prepare("
        SELECT COUNT(*) FROM game_players
        WHERE game_id = ? AND JSON_LENGTH(hand) > 0
    ");
    $stmt->execute([$gameId]);
    $cardsLeft = (int)$stmt->fetchColumn();

    if ($cardsLeft === 0) {
        if (!empty($tableCards) && $game['last_capture_player']) {
            $stmt = $db->prepare("
                SELECT id, score FROM game_players
                WHERE game_id = ? AND user_id = ?
            ");
            $stmt->execute([$gameId, $game['last_capture_player']]);
            $last = $stmt->fetch(PDO::FETCH_ASSOC);

            $bonus = calculatePoints($tableCards);
            $stmt = $db->prepare("
                UPDATE game_players SET score = ?
                WHERE id = ?
            ");
            $stmt->execute([$last['score'] + $bonus, $last['id']]);
        }

        $stmt = $db->prepare("UPDATE games SET status = 'finished' WHERE id = ?");
        $stmt->execute([$gameId]);

        $db->commit();

        echo json_encode([
            "game_over" => true
        ]);
        exit;
    }
}
```

Το παιχνίδι ολοκληρώνεται όταν δεν υπάρχουν άλλα φύλλα ούτε στην τράπουλα ούτε στα χέρια των παικτών. Σε αυτή την περίπτωση αποδίδονται τυχόν τελευταίοι πόντοι και η κατάσταση του παιχνιδιού ορίζεται ως finished.

10.8 Τελική Ενημέρωση Παιχνιδιού και Απόκριση

```php
echo json_encode([
    "success" => true,
    "table" => $tableCards,
    "score" => $score,
    "next_player" => $nextPlayer
]);
```

Η τελική απόκριση επιστρέφει την ενημερωμένη κατάσταση του τραπεζιού, το σκορ του παίκτη και τον επόμενο παίκτη που έχει σειρά. Με τον τρόπο αυτό το frontend μπορεί να ανανεώσει άμεσα το περιβάλλον χρήστη.

Ενότητα 11: Διαχείριση Τράπουλας (deck.php)

Η διαχείριση της τράπουλας υλοποιείται στο αρχείο deck.php και αποτελεί βασικό υποσύστημα του παιχνιδιού, καθώς είναι υπεύθυνη για τη δημιουργία, την τυχαιοποίηση και το αρχικό μοίρασμα των φύλλων στους παίκτες. Οι συναρτήσεις που περιλαμβάνονται έχουν σχεδιαστεί με τρόπο ώστε να είναι ανεξάρτητες από τη λογική του παιχνιδιού και να μπορούν να επαναχρησιμοποιηθούν.

```php
<?php
function createDeck() {
    $suits = ['♥', '♦', '♣', '♠'];
    $values = ['2','3','4','5','6','7','8','9','10','J','Q','K','A'];

    $deck = [];
    foreach ($suits as $suit) {
        foreach ($values as $value) {
            $deck[] = ['suit' => $suit, 'value' => $value];
        }
    }
    return $deck;
}

function shuffleDeck(array &$deck): void {
    shuffle($deck);
}

function dealHands(array &$deck): array {
    $hand1 = array_splice($deck, 0, 4);
    $hand2 = array_splice($deck, 0, 4);

    return [$hand1, $hand2];
}
```

11.1 Δημιουργία τράπουλας

Η συνάρτηση createDeck() αναλαμβάνει τη δημιουργία μιας πλήρους τράπουλας 52 φύλλων. Αρχικά ορίζονται δύο πίνακες που περιέχουν τα σύμβολα (♥, ♦, ♣, ♠) και τις αξίες των φύλλων (2–10, J, Q, K, A). Στη συνέχεια, μέσω διπλής επανάληψης, δημιουργείται κάθε δυνατός συνδυασμός συμβόλου και αξίας, ο οποίος αποθηκεύεται ως συσχετιστικός πίνακας με κλειδιά value . Το τελικό αποτέλεσμα είναι ένας μονοδιάστατος πίνακας που αναπαριστά την πλήρη τράπουλα. suit και Ανακάτεμα τράπουλας Η συνάρτηση shuffleDeck() δέχεται την τράπουλα ως αναφορά και εφαρμόζει την ενσωματωμένη συνάρτηση shuffle() της PHP. Με τον τρόπο αυτό εξασφαλίζεται η τυχαία διάταξη των φύλλων πριν από την έναρξη του παιχνιδιού, στοιχείο απαραίτητο για τη δικαιοσύνη και την ορθότητα της ροής του παιχνιδιού.

11.2 Μοίρασμα αρχικών φύλλων

Η συνάρτηση dealHands() υλοποιεί το αρχικό μοίρασμα των φύλλων στους δύο παίκτες. Από την αρχή της τράπουλας αφαιρούνται τέσσερα

φύλλα για κάθε παίκτη μέσω της array_splice() , ενώ ταυτόχρονα η τράπουλα ενημερώνεται ώστε να μην περιλαμβάνει τα μοιρασμένα φύλλα. Η συνάρτηση επιστρέφει έναν πίνακα που περιέχει τα δύο χέρια, τα οποία στη συνέχεια αποθηκεύονται στη βάση δεδομένων και χρησιμοποιούνται κατά τη διάρκεια του παιχνιδιού.

Ενότητα12: Συμμετοχή Δεύτερου Παίκτη και Έναρξη Παιχνιδιού (game_join.php)

Η συμμετοχή του δεύτερου παίκτη και η έναρξη του παιχνιδιού υλοποιούνται μέσω του αρχείου game_join.php, το οποίο δέχεται αίτημα POST σε μορφή JSON με τα πεδία game_id και user_id. Αρχικά γίνεται αποκωδικοποίηση του JSON και έλεγχος εγκυρότητας, ώστε να διασφαλιστεί ότι το αίτημα περιέχει τα απαραίτητα δεδομένα. Στη συνέχεια ξεκινά συναλλαγή (beginTransaction()), ώστε όλες οι αλλαγές στη βάση (προσθήκη παίκτη, μοίρασμα φύλλων, ενημέρωση παιχνιδιού) να εκτελεστούν ατομικά και με συνέπεια.

```php
$data = json_decode(file_get_contents("php://input"), true);

if (!isset($data['game_id'], $data['user_id'])) {
    echo json_encode(["error" => "Invalid request"]);
    exit;
}

$gameId = (int)$data['game_id'];
$userId = (int)$data['user_id'];
```

Στο επόμενο στάδιο, το σύστημα φορτώνει το παιχνίδι από τον πίνακα games και ελέγχει ότι υπάρχει και ότι βρίσκεται σε κατάσταση waiting, δηλαδή ότι δεν έχει ξεκινήσει ακόμα. Αν το παιχνίδι δεν βρεθεί ή αν έχει ήδη ξεκινήσει, επιστρέφεται σφάλμα.

```php
/* 1️⃣ Φόρτωση παιχνιδιού */
$stmt = $db->prepare("SELECT * FROM games WHERE id = ?");
$stmt->execute([$gameId]);
$game = $stmt->fetch(PDO::FETCH_ASSOC);

if (!$game) {
    throw new Exception("Game not found");
}

if ($game['status'] !== 'waiting') {
    throw new Exception("Game already started");
}
```

Έπειτα, γίνεται έλεγχος ότι υπάρχει ακριβώς ένας παίκτης ήδη καταχωρημένος στον πίνακα game_players. Αυτό εξασφαλίζει ότι το παιχνίδι βρίσκεται σε σωστή κατάσταση (έχει δημιουργηθεί από τον πρώτο παίκτη και αναμένει τον δεύτερο).

```php
/* 2️⃣ Έλεγχος αν υπάρχει ήδη παίκτης */
$stmt = $db->prepare("
    SELECT COUNT(*) FROM game_players WHERE game_id = ?
");
$stmt->execute([$gameId]);
$count = (int)$stmt->fetchColumn();

if ($count !== 1) {
    throw new Exception("Invalid game state");
}
```

Αφού επιβεβαιωθεί η ορθότητα, ο δεύτερος παίκτης προστίθεται στον πίνακα game_players με αρχικό κενό χέρι ([]) και σκορ 0. Στη συνέχεια ξεκινά το μοίρασμα: η τράπουλα ανακτάται από το πεδίο games.deck (JSON), και για κάθε παίκτη αφαιρούνται 4 φύλλα από την αρχή της τράπουλας μέσω array_splice(), τα οποία αποθηκεύονται στο hand του αντίστοιχου παίκτη.

```php
/* 3️⃣ Προσθήκη δεύτερου παίκτη */
$stmt = $db->prepare("
    INSERT INTO game_players (game_id, user_id, hand, score)
    VALUES (?, ?, '[]', 0)
");
$stmt->execute([$gameId, $userId]);

/* 4️⃣ Μοίρασμα φύλλων */
$deck = json_decode($game['deck'], true);

$stmt = $db->prepare("
    SELECT id FROM game_players WHERE game_id = ?
");
$stmt->execute([$gameId]);
$players = $stmt->fetchAll(PDO::FETCH_COLUMN);

foreach ($players as $pid) {
    $hand = array_splice($deck, 0, 4);
    $stmt = $db->prepare("
        UPDATE game_players SET hand = ? WHERE id = ?
    ");
    $stmt->execute([json_encode($hand), $pid]);
}
```

Ακολούθως, ορίζεται ποιος παίκτης θα ξεκινήσει πρώτος. Η επιλογή γίνεται με βάση το joined_at (αύξουσα σειρά), άρα πρώτος παίζει ο παίκτης που μπήκε πρώτος στο παιχνίδι (συνήθως αυτός που το δημιούργησε). Τέλος, ενημερώνεται η εγγραφή του παιχνιδιού στον πίνακα games: η κατάσταση γίνεται active, ορίζεται ο current_player και αποθηκεύεται η ενημερωμένη τράπουλα (αφού αφαιρέθηκαν τα μοιρασμένα φύλλα).

```php
/* 5️⃣ Ορισμός πρώτου παίκτη (απλά ο πρώτος που μπήκε) */
$stmt = $db->prepare("
    SELECT user_id FROM game_players
    WHERE game_id = ?
    ORDER BY joined_at ASC
    LIMIT 1
");
$stmt->execute([$gameId]);
$firstPlayer = $stmt->fetchColumn();

/* 6️⃣ Update game */
$stmt = $db->prepare("
    UPDATE games
    SET status = 'active',
        current_player = ?,
        deck = ?
    WHERE id = ?
");
$stmt->execute([
    $firstPlayer,
    json_encode($deck),
    $gameId
]);
```

Με την ολοκλήρωση όλων των παραπάνω, η συναλλαγή επιβεβαιώνεται με commit() και επιστρέφεται στο frontend JSON απάντηση που δηλώνει

επιτυχία, καθώς και ποιος παίκτης ξεκινά.

```
    $db->commit();

    echo json_encode([
        "success" => true,
        "game_id" => $gameId,
        "first_player" => $firstPlayer
    ]);

} catch (Exception $e) {
    $db->rollBack();
    echo json_encode(["error" => $e->getMessage()]);
}
```

Ενοτητα13: Ανάκτηση και Συγχρονισμός Κατάστασης Παιχνιδιού (game_state.php)

Η ανάκτηση και επιστροφή της τρέχουσας κατάστασης του παιχνιδιού υλοποιείται στο αρχείο game_state.php, το οποίο εξυπηρετεί αιτήματα GET από το frontend. Ο βασικός ρόλος του είναι να λειτουργεί ως “state provider” για τους clients, επιστρέφοντας όλα τα απαραίτητα δεδομένα ώστε ο κάθε παίκτης να ενημερώνει την οθόνη του (τραπέζι, χέρι, σκορ, σειρά). Αρχικά ορίζεται ότι η απάντηση θα είναι σε JSON και φορτώνεται η σύνδεση της βάσης μέσω του db.php.

```
header('Content-Type: application/json');

require_once 'db.php';

$db = getDB();
```

Το αρχείο απαιτεί ως είσοδο τις παραμέτρους game_id και user_id. Αν κάποια από αυτές λείπει, το request θεωρείται μη έγκυρο και επιστρέφεται σφάλμα.

```
if (!isset($_GET['game_id'], $_GET['user_id'])) {
    echo json_encode(["error" => "Invalid request"]);
    exit;
}
```

Στη συνέχεια γίνεται φόρτωση του παιχνιδιού από τον πίνακα games. Αν δεν υπάρχει αντίστοιχη εγγραφή, επιστρέφεται σφάλμα “Game not

found”. Με αυτό τον έλεγχο διασφαλίζεται ότι ο client ζητά κατάσταση για πραγματικό παιχνίδι.

```
/* 🔲 Φόρτωση παιχνιδιού */
$stmt = $db->prepare("SELECT * FROM games WHERE id = ?");
$stmt->execute([$gameId]);
$game = $stmt->fetch(PDO::FETCH_ASSOC);

if (!$game) {
    throw new Exception("Game not found");
}
```

Μετά τη φόρτωση του παιχνιδιού, αναζητούνται τα δεδομένα του παίκτη στον πίνακα game_players, συγκεκριμένα το χέρι (hand) και το σκορ (score). Αν ο χρήστης δεν ανήκει στο παιχνίδι, το σύστημα επιστρέφει σφάλμα “Player not in this game”, αποτρέποντας μη εξουσιοδοτημένη πρόσβαση στην κατάσταση παιχνιδιού.

```
$stmt = $db->prepare("
    SELECT hand, score
    FROM game_players
    WHERE game_id = ? AND user_id = ?
");
$stmt->execute([$gameId, $userId]);
$player = $stmt->fetch(PDO::FETCH_ASSOC);

if (!$player) {
    throw new Exception("Player not in this game");
}
```

Ακολούθως ανακτάται ο αντίπαλος παίκτης (μόνο το user_id και το score). Αυτό επιτρέπει στο frontend να εμφανίζει το σκορ του αντιπάλου χωρίς να αποκαλύπτεται το χέρι του, κάτι που είναι κρίσιμο για τη δικαιοσύνη του παιχνιδιού.

```
$stmt = $db->prepare("
    SELECT user_id, score
    FROM game_players
    WHERE game_id = ? AND user_id != ?
");
$stmt->execute([$gameId, $userId]);
$opponent = $stmt->fetch(PDO::FETCH_ASSOC);
```

Τέλος, το endpoint επιστρέφει ενιαία JSON απάντηση με όλα τα δεδομένα που χρειάζεται ο client. Περιλαμβάνει την κατάσταση του παιχνιδιού (status), τον τρέχοντα παίκτη (current_player) και ένα boolean your_turn, το οποίο υπολογίζεται server-side συγκρίνοντας τον current_player με το user_id. Επιπλέον επιστρέφονται τα φύλλα του τραπεζιού (table_cards) και το χέρι του παίκτη, αφού προηγουμένως γίνουν json_decode. Το σκορ του παίκτη και του αντιπάλου επιστρέφονται ως ακέραιες τιμές.

```
echo json_encode([
    "success" => true,
    "status" => $game['status'],
    "current_player" => $game['current_player'],
    "your_turn" => ((int)$game['current_player'] === $userId),

    "table" => json_decode($game['table_cards'], true),
    "hand"  => json_decode($player['hand'], true),

    "your_score" => (int)$player['score'],
    "opponent" => $opponent ? [
        "user_id" => (int)$opponent['user_id'],
        "score"   => (int)$opponent['score']
    ] : null
]);
```

Ενότητα14: Διαχείριση Σύνδεσης με τη Βάση Δεδομένων (db.php)

Η σύνδεση της εφαρμογής με τη βάση δεδομένων υλοποιείται στο αρχείο db.php, το οποίο αποτελεί το βασικό επίπεδο πρόσβασης δεδομένων για όλα τα API endpoints του backend. Κάθε αρχείο του παιχνιδιού που απαιτεί επικοινωνία με τη βάση (όπως game_init.php, game_join.php,

game_move.php και game_state.php) καλεί τη συνάρτηση getDB() ώστε να αποκτήσει ένα ενεργό αντικείμενο σύνδεσης.

Αρχικά, το db.php φορτώνει το αρχείο config.php, στο οποίο έχουν οριστεί οι σταθερές σύνδεσης (DB_HOST, DB_NAME, DB_USER, DB_PASS). Έτσι, οι παράμετροι της βάσης παραμένουν συγκεντρωμένες σε ένα σημείο, διευκολύνοντας τη συντήρηση και την αλλαγή περιβάλλοντος.

```
require_once 'config.php';
```

Στη συνέχεια, υλοποιείται η συνάρτηση getDB(), η οποία δημιουργεί σύνδεση μέσω της βιβλιοθήκης PDO. Η μεταβλητή $db δηλώνεται ως static, ώστε να διατηρεί την τιμή της μεταξύ πολλαπλών κλήσεων της συνάρτησης. Με αυτόν τον τρόπο εφαρμόζεται πρακτικά η λογική του Singleton pattern, δηλαδή η δημιουργία μίας και μοναδικής σύνδεσης που επαναχρησιμοποιείται σε όλη την εφαρμογή.

```
function getDB() {
   static $db = null;
```

Αν η σύνδεση δεν έχει δημιουργηθεί ακόμη ($db === null), τότε αρχικοποιείται με τον constructor της PDO, χρησιμοποιώντας τα στοιχεία σύνδεσης από το configuration αρχείο:

```
$db = new PDO(
    "mysql:host=".DB_HOST.";dbname=".DB_NAME,
    DB_USER,
    DB_PASS,
    [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]
);
```

χρήση PDO αποτελεί καλή πρακτική, καθώς προσφέρει αυξημένη ασφάλεια μέσω prepared statements και υποστηρίζει διαχείριση σφαλμάτων. Η επιλογή:

```
PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
```

ρυθμίζει τη σύνδεση ώστε κάθε σφάλμα βάσης να παράγει exception. Αυτό επιτρέπει στο backend να χρησιμοποιεί μηχανισμούς try/catch και συναλλαγές (transactions) για ασφαλή εκτέλεση ενεργειών, όπως συμβαίνει στα βασικά endpoints του παιχνιδιού.

Τέλος, η συνάρτηση επιστρέφει το αντικείμενο σύνδεσης:

return $db;

Με αυτόν τον τρόπο, το db.php εξασφαλίζει ενιαία, αποδοτική και ασφαλή πρόσβαση στη βάση δεδομένων, αποτελώντας θεμελιώδες στοιχείο της αρχιτεκτονικής του multiplayer παιχνιδιού.

Ενοτητα15: Σύνδεσης Βάσης Δεδομένων (config.php)

Το αρχείο config.php χρησιμοποιείται για τον ορισμό των βασικών παραμέτρων σύνδεσης της εφαρμογής με τη βάση δεδομένων. Αποτελεί το κεντρικό σημείο όπου αποθηκεύονται οι πληροφορίες που απαιτούνται για τη δημιουργία PDO σύνδεσης, όπως το όνομα του host, το όνομα της βάσης και τα στοιχεία χρήστη.

Στο συγκεκριμένο project, οι τιμές αυτές ορίζονται μέσω σταθερών (define), ώστε να είναι προσβάσιμες από όλα τα αρχεία του backend με ενιαίο τρόπο.

```php
<?php
define('DB_HOST','localhost');
define('DB_NAME','game');
define('DB_USER','root');
define('DB_PASS','');
```

Η χρήση σταθερών προσφέρει σημαντικά πλεονεκτήματα, καθώς αποφεύγεται η επανάληψη των στοιχείων σύνδεσης μέσα στον κώδικα και διευκολύνεται η μελλοντική αλλαγή περιβάλλοντος (π.χ. μετάβαση από τοπικό server σε production). Επιπλέον, η συγκέντρωση των ρυθμίσεων σε ξεχωριστό αρχείο ενισχύει την καθαρότητα και τη συντηρησιμότητα του συστήματος.

Το config.php αξιοποιείται άμεσα από το αρχείο db.php, το οποίο δημιουργεί τη σύνδεση με τη βάση δεδομένων χρησιμοποιώντας τις παραπάνω σταθερές. Έτσι επιτυγχάνεται διαχωρισμός ανάμεσα στις ρυθμίσεις (configuration) και στη λογική της εφαρμογής (business logic), σύμφωνα με καλές πρακτικές ανάπτυξης λογισμικού.

Ενοτητα16:Μοντέλο Δεδομένων και Σχεδίαση Βάσης (database.sql)

Η αποθήκευση και ο συγχρονισμός της κατάστασης του παιχνιδιού βασίζονται σε σχεσιακή βάση δεδομένων, η οποία ορίζεται στο αρχείο database.sql. Η σχεδίαση χρησιμοποιεί τρεις βασικούς πίνακες (users, games, game_players) ώστε να υποστηρίζεται multiplayer λειτουργία, αποθήκευση κατάστασης και ασφαλής συσχέτιση παικτών–παιχνιδιών.

Ο πίνακας users αποθηκεύει τους παίκτες του συστήματος. Κάθε χρήστης έχει μοναδικό αναγνωριστικό (id) και μοναδικό όνομα χρήστη (username). Παρότι η τρέχουσα υλοποίηση υποστηρίζει παιχνίδι δύο παικτών, η ύπαρξη ανεξάρτητου πίνακα χρηστών επιτρέπει μελλοντικά επέκταση (π.χ. περισσότερα παιχνίδια, προφίλ, leaderboard).

```sql
CREATE TABLE users (
id INT AUTO_INCREMENT PRIMARY KEY,
username VARCHAR(50) UNIQUE NOT NULL,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

Ο πίνακας games αναπαριστά κάθε παιχνίδι ως ανεξάρτητη οντότητα. Περιλαμβάνει την κατάσταση του παιχνιδιού (status με τιμές waiting, active, finished), τον παίκτη που έχει σειρά (current_player), την τράπουλα (deck) και τα φύλλα που βρίσκονται στο τραπέζι (table_cards). Τα πεδία deck και table_cards αποθηκεύονται σε τύπο JSON, ώστε να αποθηκεύεται άμεσα η κατάσταση χωρίς να απαιτείται επιπλέον κανονικοποίηση (π.χ. ξεχωριστός πίνακας για κάθε φύλλο). Επιπλέον, το last_capture_player χρησιμοποιείται για να αποδοθούν οι τελευταίοι πόντοι από τα φύλλα του τραπεζιού στο τέλος, σύμφωνα με τη λογική του παιχνιδιού.

```sql
CREATE TABLE games (
id INT AUTO_INCREMENT PRIMARY KEY,
status ENUM('waiting','active','finished') NOT NULL,
current_player INT NULL,
deck JSON NOT NULL,
table_cards JSON NOT NULL,
last_capture_player INT NULL,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (current_player) REFERENCES users(id),
FOREIGN KEY (last_capture_player) REFERENCES users(id));
```

Τέλος, ο πίνακας game_players υλοποιεί τη συσχέτιση πολλών-προς-πολλά μεταξύ χρηστών και παιχνιδιών, αποθηκεύοντας τους παίκτες που συμμετέχουν σε κάθε παιχνίδι. Περιέχει το hand (χέρι παίκτη) σε μορφή JSON και το score, καθώς και χρονική πληροφορία συμμετοχής (joined_at). Ο περιορισμός UNIQUE (game_id, user_id) εξασφαλίζει ότι ο ίδιος χρήστης δεν μπορεί να καταχωρηθεί δύο φορές στο ίδιο παιχνίδι, προστατεύοντας τη συνέπεια του συστήματος.

Συνολικά, η συγκεκριμένη σχεδίαση βάσης δεδομένων είναι κατάλληλη για turn-based multiplayer παιχνίδι, καθώς επιτρέπει την πλήρη

αποθήκευση της κατάστασης του παιχνιδιού (deck, table, hands, scores) και την αξιόπιστη ανάκτηση αυτής από τα endpoints του backend. Παράλληλα, οι foreign keys διασφαλίζουν αναφορική ακεραιότητα μεταξύ χρηστών, παιχνιδιών και συμμετοχών.