■ Queen's Blood Game

Complete API & Socket.IO Documentation

Version 1.0

Generated: November 18, 2025

■ Table of Contents

■ 1. Authentication Endpoints

1.1 Register

POST /api/v1/auth/register

Request:

```
{ "email": "player@example.com", "username": "player123", "password": "securePassword123",
"displayName": "Pro Player" }
```

Response (200):

```
{ "success": true, "message": "Registration successful", "data": { "user": { "uid": "uid_abc123",
"username": "player123", "email": "player@example.com", "displayName": "Pro Player",
"isEmailVerified": false, "isOnline": false } } }
```

1.2 Login

POST /api/v1/auth/login

Request:

```
{ "email": "player@example.com", "password": "securePassword123" }
```

Response (200):

```
{ "success": true, "data": { "user": { "username": "player123", ... }, "accessToken":
"eyJhbGc...", "refreshToken": "eyJhbGc..." } }
```

■ 2. Lobby System (HTTP)

2.1 Create Lobby

POST /api/v1/lobbies

Headers: Authorization: Bearer {accessToken}

Request:

```
{ "lobbyName": "Epic Battle Arena", "mapId": "map_id_123", "isPrivate": false, "gameSettings": {
"turnTimeLimit": 60, "allowSpectators": false } }
```

Response (201):

```
{ "success": true, "data": { "_id": "lobby_id_123", "lobbyName": "Epic Battle Arena", "status":
"waiting", "host": { "username": "player1", ... }, "players": [ {...} ], "playerCount": 1,
"maxPlayers": 2 } }
```

2.2 Get Public Lobbies

GET /api/v1/lobbies/public?status=waiting

Response: List of public lobbies with pagination

2.3 Join Lobby

POST /api/v1/lobbies/{lobbyId}/join

```
Request: { "password": null } Response: Updated lobby with 2 players
```

2.4 Select Deck

PUT /api/v1/lobbies/{lobbyId}/deck

```
Request: { "deckId": "deck_id_456" } Response: Updated lobby with player's deck selected
```

2.5 Select Character

PUT /api/v1/lobbies/{lobbyId}/character

```
Request: { "characterId": "char_id_789" } Response: Updated lobby with character selected
```

2.6 Start Game (Host Only)

POST /api/v1/lobbies/{lobbyId}/start

Redis Operation:

```
redis.setex('game:game_123', 7200, JSON.stringify({ gameId: 'game_123', status: 'dice_roll',
players: { ... }, board: [ /* 3x10 grid */ ] }));
```

■ 3. Lobby Socket Events

Client → Server Events

3.1 Join Lobby Room

```
socket.emit('lobby:joined', { lobbyId: 'lobby_123' }); Server Actions: - Joins socket to room
'lobby_123' - Broadcasts 'lobby:state:update' to all
```

3.2 Select Deck (Socket)

```
socket.emit('lobby:select:deck', { deckId: 'deck_id_456' }); Server validates deck ownership and
broadcasts update
```

3.3 Select Character (Socket)

```
socket.emit('lobby:select:character', { characterId: 'char_id_789' }); Server validates inventory
and broadcasts update
```

3.4 Start Game (Socket)

```
socket.emit('lobby:start:game', {}); Server creates game in Redis and broadcasts:
socket.on('game:started', (data) => { // { gameId: 'game_456', lobbyId: 'lobby_123' }
window.location.href = '/game/' + data.gameId; });
```

Server → Client Events

```
socket.on('lobby:state:update', (lobbyState) => { // Full lobby state with all players // Updated
after any change }); socket.on('game:started', ({ gameId, lobbyId }) => { // Game has been created,
redirect to game }); socket.on('lobby:kicked', ({ message }) => { // You were kicked from lobby });
```

■ 4. Game System (HTTP)

4.1 Get Active Game

GET /api/v1/games/active/{gameId}

Redis Read: redis.get('game:game_123')

Response:

```
{ "gameId": "game_123", "status": "playing", "phase": "playing", "currentTurn": "user_id_1", "me":
{ "position": "left", "hand": [ /* player cards */ ], "totalScore": 45 }, "opponent": { "position":
"right", "handCount": 5, "totalScore": 38 }, "board": [ /* 3x10 grid */ ] }
```

4.2 Get Game History

GET /api/v1/games/me/history?page=1&limit;=10

MongoDB Read: db.games.find({ 'players.userId': userId })

4.3 Get User Stats

GET /api/v1/games/me/stats

```
Response: { "totalGames": 50, "wins": 30, "losses": 20, "winRate": 60.0 }
```

■ 5. Game Socket Events

Client → Server Events

5.1 Join Game

```
socket.emit('game:join', { gameId: 'game_456' }); Server Actions: 1. Redis Read:
redis.get('game:game_456') 2. Validate player authorization 3. Transform state for player
perspective 4. Emit 'game:load' with transformed state
```

5.2 Roll Dice

```
socket.emit('game:dice_roll:submit', {}); Server Response (waiting):
socket.on('game:dice_roll:wait', (data) => { // { myRoll: 4, message: 'Waiting...' } }); Server
Response (result): socket.on('game:dice_roll:result', (result) => { // { playerA_roll: 5,
playerB_roll: 3, // firstTurn: 'user_id_1' } }); Redis Operations: - Read game state - Update
player.diceRoll and hasRolled - If both rolled, determine first turn - Update phase to 'playing' -
Write back to Redis
```

5.3 Card Hover (Preview)

```
socket.emit('game:action:hover', { cardId: 'card_123', x: 5, y: 2 }); Server Actions: 1. Transform
coordinates if away player 2. Validate pawn requirement 3. Calculate pawn and effect locations 4.
Transform locations back to display coords Response: socket.on('game:action:preview', (preview) =>
{ // { // isValid: true, // pawnLocations: [{x:4,y:2}, {x:6,y:2}], // effectLocations: [{x:5,y:1}]
// } });
```

5.4 Play Card

```
socket.emit('game:action:play_card', { cardId: 'card_123', handCardIndex: 2, x: 5, y: 2 }); Server
Actions: 1. Redis Read game state 2. Transform coords if away player: x = 9 - x, y = 2 - y 3.
Validate turn, card, placement 4. Place card on board (absolute coords) 5. Add pawns to adjacent
squares 6. Calculate row scores 7. Check game end 8. Draw card for opponent 9. Switch turn 10.
Redis Write updated state 11. Broadcast to both players Response: socket.on('game:state:update',
(gameState) => { // Full updated game state // Board is transformed per player });
```

5.5 Skip Turn

```
socket.emit('game:action:skip_turn', {}); Server: Draw card for opponent, switch turn
```

5.6 Game End

```
socket.on('game:end', (result) => { // { // status: 'completed', // winnerId: 'user_id_1', //
finalScore: { // 'user_id_1': 52, // 'user_id_2': 45 // }, // coinsWon: 2 // } }); Server Actions
on Game End: 1. Save game to MongoDB 2. Update user stats 3. Delete from Redis: -
redis.del('game:game_456') - redis.del('user:game:user_1') - redis.del('user:game:user_2')
```

■■ 6. Redis Data Structure

6.1 Active Game State

Key: game:{gameId}, **TTL:** 7200 seconds

```
{ gameId: 'game_456', status: 'playing', phase: 'playing', currentTurn: 'user_id_1', turnNumber: 8, players: { 'user_id_1': { position: 'home', hand: [ /* cards */ ], deck: [ /* card IDs */ ], totalScore: 45, rowScores: [15, 20, 10] }, 'user_id_2': { ... } }, board: [ [ // Row 0 { x: 0, y: 0, card: null, owner: null, pawns: { 'user_id_1': 2 } } ] ] }
```

6.2 User Game Mapping

```
Key: user:game:{userId} TTL: 7200 seconds Value: 'game_456' Used to quickly find which game a user is in
```

6.3 Game Room Mapping

```
Key: gameroom:{gameId} TTL: 7200 seconds Value: ['user_id_1', 'user_id_2'] Tracks which users are in a game room
```

■ 7. Board Transformation System

Each player always sees themselves on the **LEFT** and opponent on the **RIGHT**. The board is flipped 180° for the away player.

7.1 Coordinate Transformation

```
For Away Player: Display → Absolute: x = width - 1 - x, y = height - 1 - y Absolute → Display: x =
width - 1 - x, y = height - 1 - y For Home Player: No transformation needed (display = absolute)
```

7.2 Example: Card Placement

```
Player B (Away) clicks (5, 2) on screen 1. Frontend: emit('play_card', { x: 5, y: 2 }) 2. Server:
Transform to absolute x = 9 - 5 = 4 y = 2 - 2 = 0 3. Server: Place at board[0][4] 4. Broadcast to
Player A: Shows at (4, 0) - opponent's card at top 5. Broadcast to Player B: Board flipped, shows
at (5, 2) - own card
```

7.3 Board State Storage

One authoritative state in Redis (absolute coordinates). Transformed when broadcasting to players.

■ 8. Complete Flow Example

8.1 Full Game Flow

```
PHASE 1: PRE-GAME (LOBBY) Player 1: 1. POST /auth/login 2. POST /lobbies (create lobby) 3.
emit('lobby:joined') 4. PUT /lobbies/{id}/deck 5. PUT /lobbies/{id}/character Player 2: 1. POST
/auth/login 2. POST /lobbies/{id}/join 3. emit('lobby:joined') 4. PUT /lobbies/{id}/deck 5. PUT
/lobbies/{id}/character Player 1: 6. POST /lobbies/{id}/start → Redis: Create game state →
Broadcast: game:started PHASE 2: DICE ROLL Both: 1. emit('game:join') 2. on('game:load') 3.
emit('game:dice_roll:submit') 4. on('game:dice_roll:result') PHASE 3: PLAYING Players alternate:
1. emit('game:action:hover') 2. on('game:action:preview') 3. emit('game:action:play_card') 4.
on('game:state:update') PHASE 4: GAME END 1. on('game:end') 2. MongoDB: Save completed game 3.
Redis: Delete game state
```

■ 9. Quick Reference

Endpoint/Event	Method	Purpose	Redis/DB
POST /auth/register	HTTP	Create account	MongoDB
POST /auth/login	HTTP	Login	MongoDB
POST /lobbies	HTTP	Create lobby	MongoDB
POST /lobbies/{id}/join	HTTP	Join lobby	MongoDB
lobby:joined	Socket	Join lobby room	None
lobby:start:game	Socket	Start game	Redis Write
game:join	Socket	Join game	Redis Read
game:dice_roll:submit	Socket	Roll dice	Redis R/W
game:action:play_card	Socket	Play card	Redis R/W
game:end	Socket	Game finished	MongoDB, Redis Del

Redis Keys

```
game:{gameId} - Active game (TTL: 2h) user:game:{userId} - User's current game (TTL: 2h)
gameroom:{gameId} - Users in game (TTL: 2h)
```

MongoDB Collections

```
users - User accounts lobbies - Game lobbies (temp) games - Completed games (permanent) decks -
User decks cards - All available cards characters - All characters inventories - User collections
```

■ 10. User & Search System

10.1 Search Users

GET /api/v1/users/search?username=player

Headers: Authorization: Bearer {accessToken}

```
Response: { "success": true, "data": [ { "uid": "uid_123", "username": "player123", "displayName":
"Pro Player", "profilePic": "/avatars/player.png", "isOnline": true, "lastLogin": "2024-11-18T..."
} ] }
```

■ 11. Inventory System

11.1 Get My Inventory

GET /api/v1/inventory

MongoDB Read: db.inventories.findOne({ userId })

```
Response: { "success": true, "data": { "_id": "inventory_id", "userId": "user_id", "cards": [ {
"cardId": "card_123", "name": "Dragon Strike", "power": 5, "rarity": "epic", "inventoryQuantity":
3, "acquiredAt": "2024-11-18T..." } ], "characters": [ { "characterId": "char_123", "name":
"Warrior King", "rarity": "legendary", "acquiredAt": "2024-11-18T..." } ] } }
```

11.2 Add Card to Inventory

POST /api/v1/inventory/cards

```
Request: { "cardId": "card_123", "quantity": 2 } Response: Updated inventory with new card
```

11.3 Remove Card

DELETE /api/v1/inventory/cards/{cardId}

```
Request: { "quantity": 1 } Response: Updated inventory
```

11.4 Add Character

POST /api/v1/inventory/characters

```
Request: { "characterId": "char_456" } Response: Updated inventory with new character
```

■ 12. Deck Management System

12.1 Create Deck

POST /api/v1/decks

```
Request: { "deckTitle": "My Battle Deck", "cards": [ { "cardId": "card_1", "position": 0 }, {
"cardId": "card_2", "position": 1 } ], "isActive": true } Response: { "success": true, "data": {
"deck": { "deckId": "deck_123", "deckTitle": "My Battle Deck", "userId": "user_123", "isActive":
true, "cardCount": 30, "cards": [ /* array of cards */ ] } } }
```

12.2 Get User Decks

GET /api/v1/decks

```
Response: { "decks": [ { "deckId": "deck_1", "deckTitle": "Aggro Deck", "isActive": true,
"cardCount": 30 } ], "count": 1 }
```

12.3 Get Active Deck

GET /api/v1/decks/active

Returns the currently active deck

12.4 Get Deck by ID

GET /api/v1/decks/{deckId}

Returns full deck details with all cards

12.5 Update Deck

PUT /api/v1/decks/{deckId}

```
Request: { "deckTitle": "Updated Name", "cards": [ /* new card list */ ], "isActive": true }
```

12.6 Activate Deck

PATCH /api/v1/decks/{deckId}/activate

Sets this deck as active (deactivates others)

12.7 Delete Deck

DELETE /api/v1/decks/{deckId}

```
Response: { "deletedDeckId": "deck_123", "deletedAt": "2024-11-18T..." }
```

■ 13. Friends System

13.1 Get My Friends

GET /api/v1/friends

```
Response: { "success": true, "data": { "friends": [ { "friendId": "user_id_2", "uid": "uid_456",
"username": "friend1", "displayName": "Friend One", "isOnline": true } ], "count": 1 } }
```

13.2 Send Friend Request

POST /api/v1/friends/requests

```
Request: { "toUserId": "user_id_123" } Response: Friend request sent
```

13.3 Get Sent Requests

GET /api/v1/friends/requests/sent

Returns all friend requests sent by you

13.4 Get Pending Requests

GET /api/v1/friends/requests/pending

Returns friend requests waiting for your response

13.5 Accept Friend Request

POST /api/v1/friends/requests/{requestId}/accept

Accepts the request and adds friend

13.6 Decline Friend Request

POST /api/v1/friends/requests/{requestId}/decline

Declines the friend request

13.7 Remove Friend

DELETE /api/v1/friends/{friendId}

Removes user from your friends list

■ 14. Complete Controller Reference

Controller	Endpoint	Method	Authentication
AuthController	/api/v1/auth/register	POST	Public
AuthController	/api/v1/auth/login	POST	Public
AuthController	/api/v1/auth/refresh	POST	Public
AuthController	/api/v1/auth/logout	POST	Private
AuthController	/api/v1/auth/me	GET	Private
UserController	/api/v1/users/search	GET	Private
FriendController	/api/v1/friends	GET	Private
FriendController	/api/v1/friends/requests	POST	Private
InventoryController	/api/v1/inventory	GET	Private
DeckController	/api/v1/decks	POST/GET	Private
DeckController	/api/v1/decks/:id	GET/PUT/DEL	Private
LobbyController	/api/v1/lobbies	POST	Private
LobbyController	/api/v1/lobbies/public	GET	Private
LobbyController	/api/v1/lobbies/:id/join	POST	Private
LobbyController	/api/v1/lobbies/:id/start	POST	Private (Host)
GameController	/api/v1/games/active/:id	GET	Private
GameController	/api/v1/games/me/history	GET	Private
GameController	/api/v1/games/me/stats	GET	Private

■ 15. Data Transfer Objects (DTOs)

Authentication DTOs:

```
RegisterRequestDto - User registration LoginRequestDto - User login UserResponseDto - User profile
data AuthResponseDto - Auth response with tokens RefreshTokenRequestDto - Token refresh
```

Lobby DTOs:

```
LobbyResponseDto - Full lobby details LobbyListItemDto - Lobby list item CreateLobbyDto - Create
lobby request JoinLobbyDto - Join lobby request
```

Game DTOs:

```
GameResponseDto - Completed game GameListItemDto - Game list item GameStatsDto - User statistics
ActiveGameStateDto - Active game state LeaderboardEntryDto - Leaderboard entry
```

Deck DTOs:

```
DeckResponseDto - Full deck details DeckSummaryDto - Deck list item CreateDeckRequestDto - Create
deck UpdateDeckRequestDto - Update deck DeckDeletionResponseDto - Deletion response
```

Inventory DTOs:

```
InventoryResponseDto - Full inventory InventoryCardDto - Card in inventory InventoryCharacterDto -
Character in inventory
```

Friend DTOs:

```
FriendResponseDto - Friend data FriendRequestResponseDto - Friend request UserSearchResponseDto -
User search result
```

■■ 16. Error Handling & Response Codes

16.1 HTTP Status Codes

Status Code	Name	Usage
200	OK	Successful GET/PUT/PATCH
201	Created	Successful POST (resource created)
400	Bad Request	Invalid input/validation error
401	Unauthorized	Missing/invalid token
403	Forbidden	Insufficient permissions
404	Not Found	Resource not found
500	Server Error	Unexpected server error

16.2 Error Response Format

```
{ "success": false, "message": "Error description", "errors": [ { "field": "email", "message":
"Email is required" } ] }
```

16.3 Socket Error Format

```
socket.on('error', (error) => { // { message: "Error description" } }); socket.on('game:error',
(error) => { // { message: "Game-specific error" } });
```

■ Queen's Blood Game - Complete API Documentation
For questions or issues, please contact the development team.