Voyna of Meme - Card Game Server API Documentation

Version: 1.0.0 **Base URL:** `http://localhost:3000/api/v1` **Documentation Generated:** 2025-11-25

Table of Contents

Authentication & Security

Authentication Methods

1. **JWT Access Token** (expires in 15 minutes)
   - Sent via HTTP-only cookie: `accessToken`
   - OR via Authorization header: `Bearer <token>`
2. **JWT Refresh Token** (expires in 7 days)

  - Sent in response body
  - Used to obtain new access tokens

Token Payload Structure

```
{
  "userId": "507f1f77bcf86cd799439011",
  "uid": "123456",
  "email": "user@example.com",
  "username": "johndoe"
}
```

Socket.IO Authentication

- Pass access token via:
  - `socket.handshake.auth.token`
  - OR `socket.handshake.query.token`

Security Headers

- **Helmet** security middleware enabled
- **CORS** configured for frontend origin
- **HTTP-only cookies** for access tokens
- **Argon2** password hashing

HTTP REST API

All responses follow a standardized format:

```
{
  "success": true,
  "message": "Operation successful",
  "data": { ... }
}
```

Error responses:

```
{
  "success": false,
  "message": "Error description",
  "errors": [ ... ]
}
```

Authentication Endpoints

1. Register New User

Endpoint: `POST /api/v1/auth/register` **Access:** Public **Description:** Create a new user account

Request Body:

```
{
  "email": "user@example.com",
  "username": "johndoe",
  "password": "MyP@ssw0rd!",
  "displayName": "John Doe"
}
```

Validation Rules:

- `email` : Valid email address, required
- `username` : 3-20 alphanumeric characters, lowercase, required
- `password` : 8-128 characters, must contain uppercase, lowercase, number, special character, required
- `displayName` : 2-30 characters, required

Success Response (201 Created):

```
{
  "success": true,
  "message": "Registration successful! Please check your email to verify your account before logging in.",
  "data": {
    "user": {
```

```json
        "uid": "123456",
        "username": "johndoe",
        "email": "user@example.com",
        "displayName": "John Doe",
        "profilePic": null,
        "isEmailVerified": false,
        "isOnline": false,
        "stats": {
          "winRate": 0,
          "totalGames": 0,
          "wins": 0,
          "losses": 0
        },
        "createdAt": "2025-11-25T10:30:00.000Z"
      }
    }
  }
```

Error Response (400 Bad Request):

```json
{
  "success": false,
  "message": "Validation failed",
  "errors": [
    {
      "field": "username",
      "message": "Username must be at least 3 characters long"
    }
  ]
}
```

---

2. Login

Endpoint: `POST /api/v1/auth/login` **Access:** Public **Description:** Authenticate user and receive tokens

Request Body (Option 1 - Email):

```json
{
  "email": "user@example.com",
  "password": "MyP@ssw0rd!"
}
```

Request Body (Option 2 - Username):

```
{
  "username": "johndoe",
  "password": "MyP@ssw0rd!"
}
```

Success Response (200 OK):

```
{
  "success": true,
  "message": "Login successful!",
  "data": {
    "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}
```

Note: Access token is set in HTTP-only cookie named `accessToken`

Error Responses:

```
// 401 Unauthorized - Invalid credentials
{
  "success": false,
  "message": "Invalid email or password"
}
```

```
// 401 Unauthorized - Email not verified
{
  "success": false,
  "message": "Please verify your email before logging in"
}
```

---

3. Refresh Access Token

Endpoint: `POST /api/v1/auth/refresh` **Access:** Public **Description:** Get new access token using refresh token

Request Body:

```
{
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Success Response (200 OK):

```
{
  "success": true,
  "message": "Token refreshed successfully",
  "data": {
    "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}
```

Note: New access token is set in HTTP-only cookie

---

4. Verify Email (API Method)

Endpoint: `POST /api/v1/auth/verify-email` **Access:** Public **Description:** Verify email address using token

Request Body:

```
{
  "token": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6"
}
```

Success Response (200 OK):

```
{
  "success": true,
  "message": "Email verified successfully! You can now login to access all
features.",
  "data": {
    "user": {
      "uid": "123456",
      "username": "johndoe",
      "email": "user@example.com",
      "displayName": "John Doe",
      "profilePic": null,
      "isEmailVerified": true,
      "isOnline": false,
      "stats": {
        "winRate": 0,
        "totalGames": 0,
        "wins": 0,
        "losses": 0
      },
      "createdAt": "2025-11-25T10:30:00.000Z"
    }
  }
}
```

---

5. Verify Email (Link Method)

Endpoint: `GET /api/v1/auth/verify-email?token=xxx` **Access:** Public **Description:** Verify email via email link (returns HTML page)

Query Parameters:

- `token` : Verification token from email

Success Response: HTML success page

Error Response: HTML error page

---

6. Resend Verification Email

Endpoint: `POST /api/v1/auth/resend-verification` **Access:** Public **Description:** Resend email verification link

Request Body:

```
{
  "email": "user@example.com"
}
```

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Verification email sent! Please check your inbox.",
  "data": null
}
```

---

7. Logout

Endpoint: `POST /api/v1/auth/logout` **Access:** Private (requires authentication) **Description:** Logout user and clear access token

Request Headers:

```
Authorization: Bearer <access_token>
```

OR use cookie authentication

Request Body: None

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Logout successful!",
  "data": null
}
```

---

8. Get Current User

Endpoint: `GET /api/v1/auth/me` **Access:** Private (requires authentication) **Description:** Get authenticated user's profile

Success Response (200 OK):

```json
{
  "success": true,
  "message": "User profile retrieved successfully",
  "data": {
    "user": {
      "uid": "123456",
      "username": "johndoe",
      "email": "user@example.com",
      "displayName": "John Doe",
      "profilePic": "https://example.com/avatar.jpg",
      "isEmailVerified": true,
      "isOnline": true,
      "stats": {
        "winRate": 65.5,
        "totalGames": 42,
        "wins": 27,
        "losses": 15
      },
      "createdAt": "2025-11-25T10:30:00.000Z"
    }
  }
}
```

---

9. Check Authentication Status

Endpoint: `GET /api/v1/auth/status` **Access:** Public (optional authentication) **Description:** Check if user is authenticated

Success Response - Authenticated (200 OK):

```json
{
  "success": true,
  "message": "User is authenticated and verified",
  "data": {
    "isAuthenticated": true,
    "isVerified": true,
    "user": {
      "uid": "123456",
      "username": "johndoe",
      "email": "user@example.com",
      "displayName": "John Doe",
      "profilePic": null,
      "isEmailVerified": true,
      "isOnline": true,
      "stats": {
        "winRate": 65.5,
        "totalGames": 42,
        "wins": 27,
        "losses": 15
      },
      "createdAt": "2025-11-25T10:30:00.000Z"
    }
  }
}
```

Success Response - Not Authenticated (200 OK):

```
{
  "success": true,
  "message": "User is not authenticated",
  "data": {
    "isAuthenticated": false,
    "isVerified": false,
    "user": null
  }
}
```

User Endpoints

1. Search Users

Endpoint: `GET /api/v1/users/search?query=john` **Access:** Private **Description:** Search for users by username

Query Parameters:

- `query` : Search term (username substring)

Success Response (200 OK):

```
{
  "success": true,
  "message": "Users found",
  "data": {
    "users": [
      {
        "userId": "507f1f77bcf86cd799439011",
        "uid": "123456",
        "username": "johndoe",
        "displayName": "John Doe",
        "profilePic": "https://example.com/avatar.jpg",
        "isOnline": true,
        "stats": {
          "winRate": 65.5,
          "totalGames": 42,
          "wins": 27,
          "losses": 15
        }
      }
    ],
    "count": 1
  }
}
```

Deck Endpoints

1. Create Deck

Endpoint: `POST /api/v1/decks` **Access:** Private **Description:** Create a new deck (max 10 decks per user)

Request Body:

```
{
  "deckTitle": "My Starter Deck",
  "cards": [
    {
      "cardId": "507f1f77bcf86cd799439011",
      "position": 0
    },
    {
      "cardId": "507f1f77bcf86cd799439012",
      "position": 1
    },
    {
      "cardId": "507f1f77bcf86cd799439013",
      "position": 2
    }
    // ... (15-30 cards total)
  ],
  "isActive": false
}
```

Validation Rules:

- `deckTitle` : 1-50 characters, required
- `cards` : Array of 15-30 cards, required
  - `cardId` : Valid MongoDB ObjectId, must exist in user's inventory
  - `position` : Integer 0-29, optional
- `isActive` : Boolean, optional (default: false)
- Duplicate cards are allowed

Success Response (201 Created):

```
{
  "success": true,
  "message": "Deck created successfully",
  "data": {
    "deck": {
      "deckId": "507f1f77bcf86cd799439020",
      "deckTitle": "My Starter Deck",
      "userId": "507f1f77bcf86cd799439011",
      "isActive": false,
      "cardCount": 20,
      "createdAt": "2025-11-25T12:00:00.000Z",
      "updatedAt": "2025-11-25T12:00:00.000Z",
      "cards": [
        {
          "cardId": "507f1f77bcf86cd799439011",
          "position": 0,
          "name": "Fire Dragon",
          "power": 5,
          "rarity": "epic",
          "cardType": "attack",
          "cardImage": "https://example.com/cards/fire-dragon.jpg",
          "pawnRequirement": 2,
          "cardInfo": {
            "description": "Powerful dragon card",
            "effectRange": 3
          },
          "pawnLocations": [
            {"x": 0, "y": 0},
            {"x": 1, "y": 0}
          ],
          "ability": {
            "type": "boost",
            "value": 2
          }
        }
        // ... more cards
      ]
    }
  }
}
```

Error Responses:

```
// 400 Bad Request - Too few cards
{
  "success": false,
  "message": "Deck must contain at least 15 cards"
}
```

```
// 403 Forbidden - Too many decks
{
  "success": false,
  "message": "Maximum of 10 decks allowed per user"
}
```

```
// 404 Not Found - Card not in inventory
{
  "success": false,
  "message": "Card not found in your inventory"
}
```

---

2. Get All User's Decks

Endpoint: `GET /api/v1/decks` **Access:** Private **Description:** Get all decks for authenticated user

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Decks retrieved successfully",
  "data": {
    "decks": [
      {
        "deckId": "507f1f77bcf86cd799439020",
        "deckTitle": "My Starter Deck",
        "isActive": true,
        "cardCount": 20,
        "createdAt": "2025-11-25T12:00:00.000Z"
      },
      {
        "deckId": "507f1f77bcf86cd799439021",
        "deckTitle": "Legendary Deck",
        "isActive": false,
        "cardCount": 25,
        "createdAt": "2025-11-24T10:00:00.000Z"
      }
    ],
    "count": 2
  }
}
```

---

3. Get Active Deck

Endpoint: `GET /api/v1/decks/active` **Access:** Private **Description:** Get user's currently active deck

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Active deck retrieved successfully",
  "data": {
    "deck": {
      "deckId": "507f1f77bcf86cd799439020",
      "deckTitle": "My Starter Deck",
      "userId": "507f1f77bcf86cd799439011",
      "isActive": true,
      "cardCount": 20,
      "createdAt": "2025-11-25T12:00:00.000Z",
      "updatedAt": "2025-11-25T12:00:00.000Z",
      "cards": [ /* full card details */ ]
    }
  }
}
```

Success Response - No Active Deck (200 OK):

```
{
  "success": true,
  "message": "No active deck found",
  "data": {
    "deck": null
  }
}
```

---

4. Get Deck by ID

Endpoint: `GET /api/v1/decks/:deckId` **Access:** Private (must own deck) **Description:** Get specific deck details

URL Parameters:

- `deckId` : MongoDB ObjectId of the deck

Success Response (200 OK):

```
{
  "success": true,
  "message": "Deck retrieved successfully",
  "data": {
    "deck": {
      "deckId": "507f1f77bcf86cd799439020",
      "deckTitle": "My Starter Deck",
      "userId": "507f1f77bcf86cd799439011",
      "isActive": true,
      "cardCount": 20,
      "createdAt": "2025-11-25T12:00:00.000Z",
      "updatedAt": "2025-11-25T12:00:00.000Z",
      "cards": [ /* full card details */ ]
    }
  }
}
```

---

5. Update Deck

Endpoint: `PUT /api/v1/decks/:deckId` **Access:** Private (must own deck) **Description:** Update deck title, cards, or active status

Request Body (all fields optional):

```
{
  "deckTitle": "Updated Deck Name",
  "cards": [
    {
      "cardId": "507f1f77bcf86cd799439011",
      "position": 0
    }
    // ... (15-30 cards)
  ],
  "isActive": true
}
```

Success Response (200 OK):

```
{
  "success": true,
  "message": "Deck updated successfully",
  "data": {
    "deck": { /* updated deck details */ }
  }
}
```

---

6. Set Active Deck

Endpoint: `PATCH /api/v1/decks/:deckId/activate` **Access:** Private (must own deck) **Description:** Set deck as active (deactivates all other decks)

Request Body: None

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Deck activated successfully",
  "data": {
    "deck": {
      "deckId": "507f1f77bcf86cd799439020",
      "deckTitle": "My Starter Deck",
      "isActive": true,
      "cardCount": 20,
      "createdAt": "2025-11-25T12:00:00.000Z",
      "updatedAt": "2025-11-25T14:30:00.000Z",
      "cards": [ /* full card details */ ]
    }
  }
}
```

---

7. Delete Deck

Endpoint: `DELETE /api/v1/decks/:deckId` **Access:** Private (must own deck) **Description:** Delete a deck

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Deck deleted successfully",
  "data": {
    "deletedDeckId": "507f1f77bcf86cd799439020",
    "deletedAt": "2025-11-25T15:00:00.000Z"
  }
}
```

Inventory Endpoints

1. Get User Inventory

Endpoint: `GET /api/v1/inventory` **Access:** Private **Description:** Get user's card and character inventory

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Inventory retrieved successfully",
  "data": {
    "inventory": {
      "userId": "507f1f77bcf86cd799439011",
      "cards": [
        {
          "cardId": "507f1f77bcf86cd799439050",
          "quantity": 3,
          "acquiredAt": "2025-11-25T10:00:00.000Z",
          "name": "Fire Dragon",
          "power": 5,
          "rarity": "epic",
          "cardType": "attack"
        }
        // ... more cards
      ],
      "characters": [
        {
          "characterId": "507f1f77bcf86cd799439060",
          "acquiredAt": "2025-11-25T10:00:00.000Z",
          "name": "Warrior",
          "rarity": "legendary",
          "ability": {
            "name": "Power Strike",
            "description": "+2 power to all attack cards"
          }
        }
        // ... more characters
      ]
    }
  }
}
```

---

2. Add Card to Inventory

Endpoint: `POST /api/v1/inventory/cards` **Access:** Private **Description:** Add card to user's inventory (admin/gacha use)

Request Body:

```
{
  "cardId": "507f1f77bcf86cd799439050",
  "quantity": 1
}
```

Success Response (200 OK):

```
{
  "success": true,
  "message": "Card added to inventory",
  "data": {
    "inventory": { /* updated inventory */ }
  }
}
```

---

3. Remove Card from Inventory

Endpoint: `DELETE /api/v1/inventory/cards/:cardId` **Access:** Private **Description:** Remove card from inventory

Success Response (200 OK):

```
{
  "success": true,
  "message": "Card removed from inventory",
  "data": null
}
```

---

4. Add Character to Inventory

Endpoint: `POST /api/v1/inventory/characters` **Access:** Private **Description:** Add character to user's inventory

Request Body:

```
{
  "characterId": "507f1f77bcf86cd799439060"
}
```

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Character added to inventory",
  "data": {
    "inventory": { /* updated inventory */ }
  }
}
```

Friend System Endpoints

1. Get Friend List

Endpoint: `GET /api/v1/friends` **Access:** Private **Description:** Get authenticated user's friends

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Friends retrieved successfully",
  "data": {
    "friends": [
      {
        "userId": "507f1f77bcf86cd799439012",
        "uid": "234567",
        "username": "janedoe",
        "displayName": "Jane Doe",
        "profilePic": "https://example.com/jane.jpg",
        "isOnline": true,
        "stats": {
          "winRate": 72.3,
          "totalGames": 56,
          "wins": 40,
          "losses": 16
        }
      }
    ],
    "count": 1
  }
}
```

---

2. Send Friend Request

Endpoint: `POST /api/v1/friends/requests` **Access:** Private **Description:** Send friend request to another user

Request Body:

```json
{
  "toUserId": "507f1f77bcf86cd799439012"
}
```

Success Response (201 Created):

```json
{
  "success": true,
  "message": "Friend request sent successfully",
  "data": {
    "friendRequest": {
      "requestId": "507f1f77bcf86cd799439070",
      "fromUserId": "507f1f77bcf86cd799439011",
      "toUserId": "507f1f77bcf86cd799439012",
      "status": "pending",
      "createdAt": "2025-11-25T16:00:00.000Z"
    }
  }
}
```

Error Responses:

```json
// 400 Bad Request - Already friends
{
  "success": false,
  "message": "Already friends with this user"
}
```

```
// 400 Bad Request - Request already exists
{
  "success": false,
  "message": "Friend request already pending"
}
```

---

3. Get Sent Friend Requests

Endpoint: `GET /api/v1/friends/requests/sent` **Access:** Private **Description:** Get friend requests sent by user

Success Response (200 OK):

```
{
  "success": true,
  "message": "Sent requests retrieved",
  "data": {
    "requests": [
      {
        "requestId": "507f1f77bcf86cd799439070",
        "toUser": {
          "userId": "507f1f77bcf86cd799439012",
          "username": "janedoe",
          "displayName": "Jane Doe",
          "profilePic": "https://example.com/jane.jpg"
        },
        "status": "pending",
        "createdAt": "2025-11-25T16:00:00.000Z"
      }
    ],
    "count": 1
  }
}
```

---

4. Get Pending Friend Requests

Endpoint: `GET /api/v1/friends/requests/pending` **Access:** Private **Description:** Get friend requests received by user

Success Response (200 OK):

```
{
  "success": true,
  "message": "Pending requests retrieved",
  "data": {
    "requests": [
      {
        "requestId": "507f1f77bcf86cd799439071",
        "fromUser": {
          "userId": "507f1f77bcf86cd799439013",
          "username": "bobsmith",
          "displayName": "Bob Smith",
          "profilePic": "https://example.com/bob.jpg"
        },
        "status": "pending",
        "createdAt": "2025-11-25T15:30:00.000Z"
      }
    ],
    "count": 1
  }
}
```

---

5. Accept Friend Request

Endpoint: `POST /api/v1/friends/requests/:requestId/accept` **Access:** Private
Description: Accept a friend request

URL Parameters:

- `requestId` : MongoDB ObjectId of the friend request

Success Response (200 OK):

```
{
  "success": true,
  "message": "Friend request accepted",
  "data": {
    "friend": {
      "userId": "507f1f77bcf86cd799439013",
      "username": "bobsmith",
      "displayName": "Bob Smith",
      "profilePic": "https://example.com/bob.jpg",
      "isOnline": false
    }
  }
}
```

---

6. Decline Friend Request

Endpoint: `POST /api/v1/friends/requests/:requestId/decline` **Access:** Private
Description: Decline a friend request

Success Response (200 OK):

```
{
  "success": true,
  "message": "Friend request declined",
  "data": null
}
```

---

7. Remove Friend

Endpoint: `DELETE /api/v1/friends/:friendId` **Access:** Private **Description:** Remove a friend from friend list

URL Parameters:

- `friendId` : MongoDB ObjectId of the friend (user ID)

Success Response (200 OK):

```
{
  "success": true,
  "message": "Friend removed successfully",
  "data": null
}
```

Lobby Endpoints

1. Create Lobby

Endpoint: `POST /api/v1/lobbies` **Access:** Private **Description:** Create a new game lobby

Request Body:

```json
{
  "lobbyName": "Epic Battle",
  "isPrivate": false,
  "password": null,
  "mapId": "507f1f77bcf86cd799439080",
  "gameSettings": {
    "turnTimeLimit": 60,
    "allowSpectators": true
  }
}
```

Validation Rules:

- `lobbyName` : 1-50 characters, required
- `isPrivate` : Boolean, optional (default: false)
- `password` : String, optional (required if private)
- `mapId` : Valid map ObjectId, required
- `gameSettings` : Object, optional
  - `turnTimeLimit` : Number (seconds), optional
  - `allowSpectators` : Boolean, optional

Success Response (201 Created):

```json
{
  "success": true,
  "message": "Lobby created successfully",
  "data": {
    "lobbyId": "507f1f77bcf86cd799439090",
    "lobbyName": "Epic Battle",
    "hostUserId": "507f1f77bcf86cd799439011",
    "hostDeckId": null,
    "hostCharacterId": null,
    "players": [
      {
        "userId": "507f1f77bcf86cd799439011",
        "username": "johndoe",
        "displayName": "John Doe",
        "profilePic": null,
        "deckId": null,
        "characterId": null,
        "isReady": false,
        "joinedAt": "2025-11-25T17:00:00.000Z"
      }
    ],
    "mapId": "507f1f77bcf86cd799439080",
    "isPrivate": false,
    "password": null,
    "maxPlayers": 2,
    "playerCount": 1,
    "isFull": false,
    "status": "waiting",
    "gameSettings": {
      "turnTimeLimit": 60,
      "allowSpectators": true
    },
    "createdAt": "2025-11-25T17:00:00.000Z",
    "expiresAt": "2025-11-25T17:30:00.000Z"
  }
}
```

Error Response:

```
// 400 Bad Request - Already in lobby
{
  "success": false,
  "message": "You are already in an active lobby"
}
```

---

2. Get Public Lobbies

Endpoint: `GET /api/v1/lobbies/public?status=waiting&limit=20` **Access:** Private
Description: Get list of public lobbies

Query Parameters:

- `status` : Filter by status ('waiting', 'ready', 'started'), optional
- `limit` : Number of results (default: 20), optional
- `skip` : Number to skip for pagination, optional

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Public lobbies retrieved successfully",
  "data": {
    "lobbies": [
      {
        "lobbyId": "507f1f77bcf86cd799439090",
        "lobbyName": "Epic Battle",
        "hostUsername": "johndoe",
        "playerCount": 1,
        "maxPlayers": 2,
        "isFull": false,
        "status": "waiting",
        "mapId": "507f1f77bcf86cd799439080",
        "createdAt": "2025-11-25T17:00:00.000Z"
      }
    ],
    "count": 1,
    "total": 5
  }
}
```

---

3. Get Current Lobby

Endpoint: `GET /api/v1/lobbies/me/current` **Access:** Private **Description:** Get user's current active lobby

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Current lobby retrieved successfully",
  "data": {
    "lobbyId": "507f1f77bcf86cd799439090",
    "lobbyName": "Epic Battle",
    "hostUserId": "507f1f77bcf86cd799439011",
    "players": [ /* player details */ ],
    "status": "waiting",
    "playerCount": 2,
    "isFull": true
  }
}
```

Success Response - No Lobby (200 OK):

```
{
  "success": true,
  "message": "No active lobby",
  "data": null
}
```

---

4. Get Lobby by ID

Endpoint: `GET /api/v1/lobbies/:lobbyId` **Access:** Private **Description:** Get lobby details by ID

Success Response (200 OK):

```
{
  "success": true,
  "message": "Lobby retrieved successfully",
  "data": {
    "lobbyId": "507f1f77bcf86cd799439090",
    "lobbyName": "Epic Battle",
    "hostUserId": "507f1f77bcf86cd799439011",
    "players": [
      {
        "userId": "507f1f77bcf86cd799439011",
        "username": "johndoe",
        "displayName": "John Doe",
        "profilePic": null,
        "deckId": "507f1f77bcf86cd799439020",
        "characterId": "507f1f77bcf86cd799439060",
        "isReady": true,
        "joinedAt": "2025-11-25T17:00:00.000Z"
      },
      {
        "userId": "507f1f77bcf86cd799439012",
        "username": "janedoe",
        "displayName": "Jane Doe",
        "profilePic": "https://example.com/jane.jpg",
        "deckId": null,
        "characterId": null,
        "isReady": false,
        "joinedAt": "2025-11-25T17:05:00.000Z"
      }
    ],
    "mapId": "507f1f77bcf86cd799439080",
    "status": "waiting",
    "playerCount": 2,
    "maxPlayers": 2,
    "isFull": true,
    "gameSettings": {
      "turnTimeLimit": 60,
      "allowSpectators": true
    }
  }
}
```

---

5. Join Lobby

Endpoint: `POST /api/v1/lobbies/:lobbyId/join` **Access:** Private **Description:** Join an existing lobby

Request Body (for private lobbies):

```json
{
  "password": "secret123"
}
```

Success Response (200 OK):

```
{
  "success": true,
  "message": "Successfully joined lobby",
  "data": {
    "lobbyId": "507f1f77bcf86cd799439090",
    "lobbyName": "Epic Battle",
    "players": [ /* updated player list */ ],
    "playerCount": 2,
    "isFull": true
  }
}
```

Error Responses:

```
// 400 Bad Request - Lobby full
{
  "success": false,
  "message": "Lobby is full"
}
```

```
// 403 Forbidden - Wrong password
{
  "success": false,
  "message": "Incorrect password"
}
```

---

6. Leave Lobby

Endpoint: `POST /api/v1/lobbies/:lobbyId/leave` **Access:** Private **Description:** Leave a lobby

Success Response (200 OK):

```
{
  "success": true,
  "message": "Successfully left lobby",
  "data": {
    "lobbyId": "507f1f77bcf86cd799439090",
    "players": [ /* updated player list */ ],
    "playerCount": 1
  }
}
```

Success Response - Lobby Closed (200 OK):

```json
{
  "success": true,
  "message": "Lobby has been closed",
  "data": null
}
```

---

7. Select Deck in Lobby

Endpoint: `PUT /api/v1/lobbies/:lobbyId/deck` **Access:** Private **Description:** Select deck for game

Request Body:

```json
{
  "deckId": "507f1f77bcf86cd799439020"
}
```

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Deck selected successfully",
  "data": {
    "lobbyId": "507f1f77bcf86cd799439090",
    "players": [
      {
        "userId": "507f1f77bcf86cd799439011",
        "username": "johndoe",
        "deckId": "507f1f77bcf86cd799439020",
        "isReady": false
      }
    ]
  }
}
```

---

8. Select Character in Lobby

Endpoint: `PUT /api/v1/lobbies/:lobbyId/character` **Access:** Private **Description:** Select character for game

Request Body:

```
{
  "characterId": "507f1f77bcf86cd799439060"
}
```

Success Response (200 OK):

```
{
  "success": true,
  "message": "Character selected successfully",
  "data": {
    "lobbyId": "507f1f77bcf86cd799439090",
    "players": [
      {
        "userId": "507f1f77bcf86cd799439011",
        "username": "johndoe",
        "characterId": "507f1f77bcf86cd799439060",
        "isReady": false
      }
    ]
  }
}
```

---

9. Update Lobby Settings (Host Only)

Endpoint: `PUT /api/v1/lobbies/:lobbyId/settings` **Access:** Private (host only) **Description:** Update lobby settings

Request Body:

```
{
  "lobbyName": "Updated Battle Name",
  "mapId": "507f1f77bcf86cd799439081",
  "gameSettings": {
    "turnTimeLimit": 90,
    "allowSpectators": false
  }
}
```

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Lobby settings updated successfully",
  "data": {
    "lobbyId": "507f1f77bcf86cd799439090",
    "lobbyName": "Updated Battle Name",
    "mapId": "507f1f77bcf86cd799439081",
    "gameSettings": {
      "turnTimeLimit": 90,
      "allowSpectators": false
    }
  }
}
```

---

10. Kick Player (Host Only)

Endpoint: `POST /api/v1/lobbies/:lobbyId/kick` **Access:** Private (host only) **Description:** Kick a player from lobby

Request Body:

```json
{
  "playerId": "507f1f77bcf86cd799439012"
}
```

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Player kicked successfully",
  "data": {
    "lobbyId": "507f1f77bcf86cd799439090",
    "players": [ /* updated player list */ ],
    "playerCount": 1
  }
}
```

---

11. Start Game (Host Only)

Endpoint: `POST /api/v1/lobbies/:lobbyId/start` **Access:** Private (host only)
Description: Start the game

Requirements:

- Lobby must be full (2 players)
- All players must have selected deck and character
- Host-only operation

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Game started successfully",
  "data": {
    "gameId": "507f1f77bcf86cd799439100",
    "lobbyId": "507f1f77bcf86cd799439090",
    "status": "started",
    "message": "Game initialized"
  }
}
```

Error Response:

```
// 400 Bad Request - Not ready
{
  "success": false,
  "message": "All players must select deck and character before starting"
}
```

---

12. Cancel Lobby (Host Only)

Endpoint: `DELETE /api/v1/lobbies/:lobbyId` **Access:** Private (host only) **Description:** Cancel and delete lobby

Success Response (200 OK):

```
{
  "success": true,
  "message": "Lobby cancelled successfully",
  "data": null
}
```

Game Endpoints

1. Get Game by ID

Endpoint: `GET /api/v1/games/:gameId` **Access:** Private **Description:** Get completed game details

Success Response (200 OK):

```
{
  "success": true,
  "message": "Game retrieved successfully",
  "data": {
    "game": {
      "gameId": "507f1f77bcf86cd799439100",
      "lobbyId": "507f1f77bcf86cd799439090",
      "players": [
        {
          "userId": "507f1f77bcf86cd799439011",
          "username": "johndoe",
          "displayName": "John Doe",
          "finalScore": 25,
          "cardsPlayed": 15
        },
        {
          "userId": "507f1f77bcf86cd799439012",
          "username": "janedoe",
          "displayName": "Jane Doe",
          "finalScore": 22,
          "cardsPlayed": 14
        }
      ],
      "mapId": "507f1f77bcf86cd799439080",
      "totalTurns": 20,
      "status": "completed",
      "winner": "507f1f77bcf86cd799439011",
      "gameDuration": 1200,
      "startedAt": "2025-11-25T18:00:00.000Z",
      "completedAt": "2025-11-25T18:20:00.000Z"
    }
  }
}
```

---

2. Get Game History

Endpoint: `GET /api/v1/games/me/history?limit=10&skip=0` **Access:** Private
Description: Get user's game history

Query Parameters:

- `limit` : Number of results (default: 10)
- `skip` : Pagination offset (default: 0)

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Game history retrieved",
  "data": {
    "games": [
      {
        "gameId": "507f1f77bcf86cd799439100",
        "opponent": {
          "userId": "507f1f77bcf86cd799439012",
          "username": "janedoe",
          "displayName": "Jane Doe"
        },
        "result": "win",
        "myScore": 25,
        "opponentScore": 22,
        "gameDuration": 1200,
        "completedAt": "2025-11-25T18:20:00.000Z"
      }
    ],
    "count": 1,
    "total": 42
  }
}
```

---

3. Get Recent Games

Endpoint: `GET /api/v1/games/me/recent?limit=5` **Access:** Private **Description:** Get user's most recent games

Success Response (200 OK):

```
{
  "success": true,
  "message": "Recent games retrieved",
  "data": {
    "games": [ /* similar to history */ ],
    "count": 5
  }
}
```

---

4. Get User Statistics

Endpoint: `GET /api/v1/games/me/stats` **Access:** Private **Description:** Get user's game statistics

Success Response (200 OK):

```
{
  "success": true,
  "message": "Statistics retrieved successfully",
  "data": {
    "stats": {
      "totalGames": 42,
      "wins": 27,
      "losses": 15,
      "draws": 0,
      "winRate": 65.5,
      "averageScore": 23.4,
      "totalGameTime": 50400,
      "favoriteMap": "Classic Arena",
      "mostUsedCharacter": "Warrior"
    }
    }
  }
```

---

5. Get Leaderboard

Endpoint: `GET /api/v1/games/leaderboard?limit=100` **Access:** Private **Description:** Get global leaderboard

Query Parameters:

- `limit` : Number of top players (default: 100)

Success Response (200 OK):

```json
{
  "success": true,
  "message": "Leaderboard retrieved",
  "data": {
    "leaderboard": [
      {
        "rank": 1,
        "userId": "507f1f77bcf86cd799439011",
        "username": "johndoe",
        "displayName": "John Doe",
        "profilePic": null,
        "stats": {
          "winRate": 75.2,
          "totalGames": 125,
          "wins": 94,
          "losses": 31
        }
      },
      {
        "rank": 2,
        "userId": "507f1f77bcf86cd799439012",
        "username": "janedoe",
        "displayName": "Jane Doe",
        "profilePic": "https://example.com/jane.jpg",
        "stats": {
          "winRate": 72.3,
          "totalGames": 56,
          "wins": 40,
          "losses": 16
        }
      }
    ],
    "count": 100
  }
}
```

---

6. Get Head-to-Head Record

Endpoint: `GET /api/v1/games/head-to-head/:opponentId` **Access:** Private **Description:** Get win/loss record against specific opponent

Success Response (200 OK):

```
{
  "success": true,
  "message": "Head-to-head record retrieved",
  "data": {
    "opponent": {
      "userId": "507f1f77bcf86cd799439012",
      "username": "janedoe",
      "displayName": "Jane Doe"
    },
    "record": {
      "totalGames": 12,
      "wins": 7,
      "losses": 5,
      "draws": 0,
      "winRate": 58.3
    },
    "recentGames": [ /* last 5 games */ ]
  }
}
```

---

7. Get Active Game State

Endpoint: `GET /api/v1/games/active/:gameId` **Access:** Private **Description:** Get active game state from Redis

Success Response (200 OK):

```
{
  "success": true,
  "message": "Active game state retrieved",
  "data": {
    "gameId": "507f1f77bcf86cd799439100",
    "status": "active",
    "phase": "playing",
    "currentTurn": "507f1f77bcf86cd799439011",
    "turnNumber": 5,
    "players": {
      "507f1f77bcf86cd799439011": {
        "userId": "507f1f77bcf86cd799439011",
        "username": "johndoe",
        "position": "home",
        "hand": [ /* cards in hand */ ],
        "deckCount": 15,
        "totalScore": 12,
        "rowScores": [4, 5, 3]
      },
      "507f1f77bcf86cd799439012": {
        "userId": "507f1f77bcf86cd799439012",
        "username": "janedoe",
        "position": "away",
        "handCount": 5,
        "deckCount": 16,
        "totalScore": 10,
        "rowScores": [3, 4, 3]
      }
    },
    "board": [ /* game board state */ ]
  }
}
```

---

8. Forfeit Game

Endpoint: `POST /api/v1/games/:gameId/forfeit` **Access:** Private **Description:** Forfeit an active game

Success Response (200 OK):

```
{
  "success": true,
  "message": "Game forfeited",
  "data": {
    "gameId": "507f1f77bcf86cd799439100",
    "status": "completed",
    "winner": "507f1f77bcf86cd799439012",
    "reason": "forfeit"
  }
}
```

Gacha Endpoints

1. Single Card Pull

Endpoint: `POST /api/v1/gacha/pull/single` **Access:** Private **Description:** Pull a single card (costs 1 coin)

Request Body: None

Success Response (200 OK):

```
{
  "success": true,
  "message": "Card pulled successfully",
  "data": {
    "card": {
      "cardId": "507f1f77bcf86cd799439050",
      "name": "Fire Dragon",
      "power": 5,
      "rarity": "epic",
      "cardType": "attack",
      "cardImage": "https://example.com/cards/fire-dragon.jpg",
      "isNew": false
    },
    "remainingCoins": 9,
    "pityCounters": {
      "totalPulls": 25,
      "pullsSinceLastEpic": 0,
      "pullsSinceLastLegendary": 25
    }
  }
}
```

Error Response:

```
// 400 Bad Request - Insufficient coins
{
  "success": false,
  "message": "Insufficient coins. You need 1 coin to pull."
}
```

---

2. Multi Card Pull (10 Cards)

Endpoint: `POST /api/v1/gacha/pull/multi` **Access:** Private **Description:** Pull 10 cards (costs 10 coins, guarantees 1 epic or better)

Request Body: None

Success Response (200 OK):

```
{
  "success": true,
  "message": "10 cards pulled successfully",
  "data": {
    "cards": [
      {
        "cardId": "507f1f77bcf86cd799439050",
        "name": "Fire Dragon",
        "power": 5,
        "rarity": "epic",
        "cardType": "attack",
        "isNew": true
      },
      {
        "cardId": "507f1f77bcf86cd799439051",
        "name": "Ice Warrior",
        "power": 3,
        "rarity": "common",
        "cardType": "defense",
        "isNew": false
      }
      // ... 8 more cards
    ],
    "summary": {
      "common": 6,
      "rare": 3,
      "epic": 1,
      "legendary": 0,
      "newCards": 2
    },
    "remainingCoins": 15,
    "pityCounters": {
      "totalPulls": 35,
      "pullsSinceLastEpic": 0,
      "pullsSinceLastLegendary": 35
    }
  }
}
```

---

3. Get Gacha Info

Endpoint: `GET /api/v1/gacha/info` **Access:** Private **Description:** Get user's gacha information (coins, pity, rates)

Success Response (200 OK):

```
{
  "success": true,
  "message": "Gacha info retrieved",
  "data": {
    "coins": 25,
    "pityCounters": {
      "totalPulls": 35,
      "pullsSinceLastEpic": 8,
      "pullsSinceLastLegendary": 35
    },
    "dropRates": {
      "common": 60.0,
      "rare": 30.0,
      "epic": 9.0,
      "legendary": 1.0
    },
    "pityRules": {
      "epicPity": 10,
      "legendaryPity": 90
    },
    "costs": {
      "singlePull": 1,
      "multiPull": 10
    }
  }
}
```

Socket.IO Real-Time Events

Connection Authentication

Client Connection:

```
const socket = io('http://localhost:3000', {
  auth: {
    token: '<access_token>'
  }
});
```

OR using query parameters:

```
const socket = io('http://localhost:3000?token=<access_token>');
```

Server Authentication:

- Validates JWT token on connection
- Verifies user exists in database
- Attaches user info to socket (userId, username, displayName)
- Auto-disconnects if authentication fails

Lobby Socket Events

Client → Server Events

1. Join Lobby (Auto-Join on Connection)

Event: `lobby:joined` **Description:** Called automatically or from REST API after joining lobby

Emit:

```
socket.emit('lobby:joined', {
  lobbyId: '507f1f77bcf86cd799439090'
});
```

No direct response - Server broadcasts `lobby:state:update` to all players

---

2. Leave Lobby

Event: `lobby:leave` **Description:** Leave current lobby

Emit:

```
socket.emit('lobby:leave', {
  lobbyId: '507f1f77bcf86cd799439090' // optional
});
```

Server Response:

```
// To leaving user
socket.on('lobby:left', (data) => {
  console.log(data);
  // { message: 'Successfully left lobby' }
});

// To remaining players (broadcast)
socket.on('lobby:state:update', (lobby) => {
  console.log(lobby);
  // Updated lobby state with player removed
});

// If lobby was closed (host left)
socket.on('lobby:closed', (data) => {
  console.log(data);
  // { message: 'Lobby has been closed' }
});
```

---

3. Update Lobby Settings (Host Only)

Event: `lobby:update:settings` **Description:** Update lobby name, map, or game settings

Emit:

```
socket.emit('lobby:update:settings', {
  lobbyName: 'Updated Battle',
  mapId: '507f1f77bcf86cd799439081',
  gameSettings: {
    turnTimeLimit: 90,
    allowSpectators: false
  }
});
```

Server Response:

```
// To all players in lobby (broadcast)
socket.on('lobby:state:update', (lobby) => {
  console.log(lobby.lobbyName); // 'Updated Battle'
  console.log(lobby.gameSettings.turnTimeLimit); // 90
});

// On error
socket.on('error', (error) => {
  console.error(error);
  // { message: 'Only the host can update lobby settings' }
});
```

---

4. Select Deck

Event: `lobby:select:deck` **Description:** Select deck for the game

Emit:

```
socket.emit('lobby:select:deck', {
  deckId: '507f1f77bcf86cd799439020'
});
```

Server Response:

```
// To all players (broadcast)
socket.on('lobby:state:update', (lobby) => {
  console.log(lobby.players[0].deckId); // '507f1f77bcf86cd799439020'
});

// On error
socket.on('error', (error) => {
  console.error(error);
  // { message: 'Deck not found in your inventory' }
});
```

---

5. Select Character

Event: `lobby:select:character` **Description:** Select character for the game

Emit:

```
socket.emit('lobby:select:character', {
  characterId: '507f1f77bcf86cd799439060'
});
```

Server Response:

```
// To all players (broadcast)
socket.on('lobby:state:update', (lobby) => {
  console.log(lobby.players[0].characterId); // '507f1f77bcf86cd799439060'
});

// On error
socket.on('error', (error) => {
  console.error(error);
  // { message: 'You do not own this character' }
});
```

---

6. Kick Player (Host Only)

Event: `lobby:kick:player` **Description:** Kick a player from lobby

Emit:

```
socket.emit('lobby:kick:player', {
  playerId: '507f1f77bcf86cd799439012'
});
```

Server Response:

```javascript
// To kicked player
socket.on('lobby:kicked', (data) => {
  console.log(data);
  // { message: 'You have been kicked from the lobby' }
});

// To remaining players (broadcast)
socket.on('lobby:state:update', (lobby) => {
  console.log(lobby.playerCount); // Decreased by 1
});
```

---

7. Start Game (Host Only)

Event: `lobby:start:game` **Description:** Start the game

Emit:

```javascript
socket.emit('lobby:start:game', {});
```

Server Response:

```javascript
// To all players (broadcast)
socket.on('game:started', (data) => {
  console.log(data);
  /*
  {
    lobbyId: '507f1f77bcf86cd799439090',
    gameId: '507f1f77bcf86cd799439100',
    message: 'Game initialized, switching to game view...'
  }
  */

  // Client should now join game using gameId
});

// On error
```

```
socket.on('error', (error) => {
  console.error(error);
  // { message: 'All players must select deck and character before starting' }
});
```

---

8. Toggle Ready Status

Event: `lobby:ready:toggle` **Description:** Toggle player's ready status

Emit:

```
socket.emit('lobby:ready:toggle', {
  isReady: true
});
```

Server Response:

```
// To all players (broadcast)
socket.on('lobby:state:update', (lobby) => {
  console.log(lobby.players[0].isReady); // true
});
```

Server → Client Events

1. Lobby Reconnected

Event: `lobby:reconnected` **Description:** Sent when user reconnects to their active lobby

Receive:

```
socket.on('lobby:reconnected', (data) => {
  console.log(data);
  /*
  {
    lobbyId: '507f1f77bcf86cd799439090',
    message: 'Reconnected to lobby'
  }
  */
});
```

---

2. Lobby State Update (Master Event)

Event: `lobby:state:update` **Description:** Full lobby state broadcast to all players

Receive:

```
socket.on('lobby:state:update', (lobby) => {
  console.log(lobby);
  /*
  {
    lobbyId: '507f1f77bcf86cd799439090',
    lobbyName: 'Epic Battle',
    hostUserId: '507f1f77bcf86cd799439011',
    hostDeckId: '507f1f77bcf86cd799439020',
    hostCharacterId: '507f1f77bcf86cd799439060',
    players: [
      {
        userId: '507f1f77bcf86cd799439011',
        username: 'johndoe',
        displayName: 'John Doe',
        profilePic: null,
        deckId: '507f1f77bcf86cd799439020',
        characterId: '507f1f77bcf86cd799439060',
        isReady: true,
        joinedAt: '2025-11-25T17:00:00.000Z'
      },
      {
        userId: '507f1f77bcf86cd799439012',
        username: 'janedoe',
        displayName: 'Jane Doe',
        profilePic: 'https://example.com/jane.jpg',
        deckId: null,
        characterId: null,
        isReady: false,
        joinedAt: '2025-11-25T17:05:00.000Z'
      }
    ],
    mapId: '507f1f77bcf86cd799439080',
    isPrivate: false,
    maxPlayers: 2,
    playerCount: 2,
    isFull: true,
    status: 'waiting',
    gameSettings: {
      turnTimeLimit: 60,
      allowSpectators: true
    },
    createdAt: '2025-11-25T17:00:00.000Z',
    expiresAt: '2025-11-25T17:30:00.000Z'
  }
  */
});
```

---

3. Error

Event: `error` **Description:** Error notification

Receive:

```
socket.on('error', (error) => {
  console.error(error);
  /*
  {
    message: 'Error description'
  }
  */
});
```

Game Socket Events

Client → Server Events

1. Join Game

Event: `game:join` **Description:** Join an active game

Emit:

```
socket.emit('game:join', {
  gameId: '507f1f77bcf86cd799439100'
});
```

Server Response:

```
// Initial game state (transformed for player's perspective)
socket.on('game:load', (gameState) => {
  console.log(gameState);
  /*
  {
    gameId: '507f1f77bcf86cd799439100',
    status: 'active',
    phase: 'dice_roll',
    currentTurn: null,
    turnNumber: 0,
    me: {
      userId: '507f1f77bcf86cd799439011',
      username: 'johndoe',
      displayName: 'John Doe',
      position: 'left',  // Always LEFT for viewing player
      character: { ... },
      hand: [ ... ],     // Full hand visible to owner
      deckCount: 20,
      totalScore: 0,
      rowScores: [0, 0, 0],
      diceRoll: null,
      hasRolled: false
    },
    opponent: {
      userId: '507f1f77bcf86cd799439012',
      username: 'janedoe',
      displayName: 'Jane Doe',
      position: 'right',  // Always RIGHT for opponent
      character: { ... },
      handCount: 5,       // Only count, not cards
      deckCount: 20,
      totalScore: 0,
      rowScores: [0, 0, 0],
      diceRoll: null,
      hasRolled: false
    },
    board: [
      [  // Row 0
        {
          x: 0, y: 0,
          card: null,
          owner: null,
          pawns: {},
          special: null
        },
        // ... more squares
      ],
      // ... more rows
    ],
    settings: {
      turnTimeLimit: 60
    }
  }
  */
});

// If in dice roll phase
socket.on('game:dice_roll:start', (data) => {
  console.log(data);
  // { message: 'Roll for first turn!' }
});

// On error
socket.on('game:error', (error) => {
  console.error(error);
  // { message: 'You are not authorized to join this game' }
});
```

---

2. Submit Dice Roll

Event: `game:dice_roll:submit` **Description:** Roll dice to determine first turn

Emit:

```
socket.emit('game:dice_roll:submit', {});
```

Server Response:

```
// If waiting for opponent
socket.on('game:dice_roll:wait', (data) => {
  console.log(data);
  /*
  {
    type: 'wait',
    myRoll: 4,
    message: 'Waiting for opponent...'
  }
  */
});

// If tie (broadcast to both)
socket.on('game:dice_roll:result', (data) => {
  console.log(data);
  /*
  {
    type: 'tie',
    rolls: {
      '507f1f77bcf86cd799439011': 4,
      '507f1f77bcf86cd799439012': 4
    },
    message: 'Tie! Roll again...'
  }
  */
});

// After tie, receive:
socket.on('game:dice_roll:start', (data) => {
  // { message: 'Roll again!' }
});

// If winner determined (broadcast to both)
socket.on('game:dice_roll:result', (data) => {
  console.log(data);
  /*
  {
    type: 'result',
    winner: '507f1f77bcf86cd799439011',
    rolls: {
      '507f1f77bcf86cd799439011': 5,
      '507f1f77bcf86cd799439012': 3
    },
    message: 'johndoe goes first!'
  }
  */
});

// Then receive updated game state
socket.on('game:state:update', (gameState) => {
  console.log(gameState.phase); // 'playing'
  console.log(gameState.currentTurn); // '507f1f77bcf86cd799439011'
});
```

---

3. Card Hover (Preview)

Event: `game:action:hover` **Description:** Preview card placement before playing

Emit:

```
socket.emit('game:action:hover', {
  cardId: '507f1f77bcf86cd799439050',
  x: 5,
  y: 1
});
```

Server Response:

```
// Only to requesting player (not broadcast)
socket.on('game:action:preview', (preview) => {
  console.log(preview);
  /*
  {
    isValid: true,
    pawnLocations: [
      { x: 5, y: 1 },
      { x: 6, y: 1 }
    ],
    effectLocations: [
      { x: 4, y: 1 },
      { x: 5, y: 0 },
      { x: 6, y: 1 }
    ],
    affectedCards: [
      { x: 4, y: 1, powerChange: +2 }
    ]
  }
  */
});

// On error
socket.on('game:error', (error) => {
  console.error(error);
  // { message: 'Invalid placement' }
});
```

---

4. Play Card

Event: `game:action:play_card` **Description:** Play a card from hand to board

Emit:

```
socket.emit('game:action:play_card', {
  cardId: '507f1f77bcf86cd799439050',
  handCardIndex: 2,
  x: 5,
  y: 1
});
```

Server Response:

```
// To all players (broadcast)
socket.on('game:state:update', (gameState) => {
  console.log(gameState);
  /*
  {
    gameId: '507f1f77bcf86cd799439100',
    phase: 'playing',
    currentTurn: '507f1f77bcf86cd799439012',  // Switched to opponent
    turnNumber: 6,
    me: {
      hand: [ ... ],  // Card removed
      deckCount: 14,  // Drew new card
      totalScore: 15,
      rowScores: [5, 7, 3]
    },
    opponent: {
      handCount: 5,
      deckCount: 15,
      totalScore: 12,
      rowScores: [4, 5, 3]
    },
    board: [
      // Updated with played card
      [
        { x: 5, y: 1, card: {...}, owner: 'me', pawns: {...} }
      ]
    ]
  }
  */
});
```

```
// If game ended
socket.on('game:end', (endData) => {
  console.log(endData);
  /*
  {
    status: 'completed',
    winnerId: '507f1f77bcf86cd799439011',
    finalScore: {
      '507f1f77bcf86cd799439011': 25,
      '507f1f77bcf86cd799439012': 22
    },
    coinsWon: 2
  }
  */
});

// On error
socket.on('game:error', (error) => {
  console.error(error);
  // { message: 'Not your turn' }
});
```

---

5. Skip Turn

Event: `game:action:skip_turn` **Description:** Skip current turn

Emit:

```
socket.emit('game:action:skip_turn', {});
```

Server Response:

```
// To all players (broadcast)
socket.on('game:state:update', (gameState) => {
  console.log(gameState.currentTurn); // Switched to opponent
  console.log(gameState.turnNumber);  // Incremented
});
```

---

6. Leave Game

Event: `game:leave` **Description:** Leave game (forfeit if active)

Emit:

```
socket.emit('game:leave', {});
```

No response - Connection terminated

Server → Client Events

1. Game Load (Initial State)

Event: `game:load` **Description:** Sent when player joins game

See example in Join Game above.

---

2. Game State Update

Event: `game:state:update` **Description:** Broadcast updated game state to all players

Receive:

```
socket.on('game:state:update', (gameState) => {
  // See full gameState structure in game:load example
  // Each player receives state transformed for their perspective
  // - Viewing player is always on LEFT
  // - Opponent is always on RIGHT
  // - Board is flipped 180° for away player
});
```

---

3. Dice Roll Events

See examples in Submit Dice Roll above.

---

4. Game End

Event: `game:end` **Description:** Game completed with final results

Receive:

```
socket.on('game:end', (endData) => {
  console.log(endData);
  /*
  {
    status: 'completed',
    winnerId: '507f1f77bcf86cd799439011',
    finalScore: {
      '507f1f77bcf86cd799439011': 25,
      '507f1f77bcf86cd799439012': 22
    },
    coinsWon: 2  // Winner gets 2 coins
  }
  */
});
```

---

5. Game Error

Event: `game:error` **Description:** Game-specific error

Receive:

```
socket.on('game:error', (error) => {
  console.error(error);
  /*
  {
    message: 'Error description'
  }
  */
});
```

Lobby List Broadcast Events

These events are broadcast to all connected clients to update public lobby lists in real-time.

1. Lobby Created

Event: `lobbyList:lobby:created` **Description:** New public lobby created

Receive:

```
socket.on('lobbyList:lobby:created', (lobby) => {
  console.log(lobby);
  /*
  {
    lobbyId: '507f1f77bcf86cd799439090',
    lobbyName: 'Epic Battle',
    hostUsername: 'johndoe',
    playerCount: 1,
    maxPlayers: 2,
    isFull: false,
    status: 'waiting',
    mapId: '507f1f77bcf86cd799439080',
    createdAt: '2025-11-25T17:00:00.000Z'
  }
  */
});
```

---

2. Lobby Updated

Event: `lobbyList:lobby:updated` **Description:** Public lobby updated (player joined/left)

Receive:

```
socket.on('lobbyList:lobby:updated', (lobby) => {
  console.log(lobby);
  /*
  {
    lobbyId: '507f1f77bcf86cd799439090',
    lobbyName: 'Epic Battle',
    playerCount: 2,  // Updated
    maxPlayers: 2,
    isFull: true,      // Updated
    status: 'waiting'
  }
  */
});
```

---

3. Lobby Deleted

Event: `lobbyList:lobby:deleted` **Description:** Lobby deleted/cancelled

Receive:

```
socket.on('lobbyList:lobby:deleted', (data) => {
  console.log(data);
  /*
  {
    lobbyId: '507f1f77bcf86cd799439090'
  }
  */
});
```

4. Lobby Started

Event: `lobbyList:lobby:started` **Description:** Lobby started (remove from public list)

Receive:

```
socket.on('lobbyList:lobby:started', (data) => {
  console.log(data);
  /*
  {
    lobbyId: '507f1f77bcf86cd799439090'
  }
  */
});
```

Data Models

User Model

```
{
  "_id": "507f1f77bcf86cd799439011",
  "uid": "123456",                // 6-digit unique ID
  "username": "johndoe",           // 3-20 chars, lowercase, unique
  "email": "user@example.com",     // Unique
  "password": "hashed_password",   // Argon2 hash
  "displayName": "John Doe",       // 2-30 chars
  "coins": 25,                     // For gacha
  "gachaPity": {
    "totalPulls": 35,
    "pullsSinceLastEpic": 8,
    "pullsSinceLastLegendary": 35
  },
  "stats": {
    "winRate": 65.5,
    "totalGames": 42,
    "wins": 27,
    "losses": 15
  },
  "gameHistory": [
    {
      "gameId": "507f1f77bcf86cd799439100",
      "opponent": "507f1f77bcf86cd799439012",
      "result": "win",
      "playedAt": "2025-11-25T18:20:00.000Z"
    }
  ],
  "isOnline": true,
  "lastLogin": "2025-11-25T10:00:00.000Z",
  "profilePic": "https://example.com/avatar.jpg",
  "isEmailVerified": true,
  "friends": ["507f1f77bcf86cd799439012"],
  "inventory": "507f1f77bcf86cd799439200",
  "createdAt": "2025-11-25T10:30:00.000Z",
  "updatedAt": "2025-11-25T18:20:00.000Z"
}
```

Deck Model

```
{
  "_id": "507f1f77bcf86cd799439020",
  "deckTitle": "My Starter Deck",
  "userId": "507f1f77bcf86cd799439011",
  "cards": [
    {
      "cardId": "507f1f77bcf86cd799439050",
      "position": 0
    }
    // ... 15-30 cards total
  ],
  "isActive": true,
  "createdAt": "2025-11-25T12:00:00.000Z",
  "updatedAt": "2025-11-25T14:30:00.000Z"
}
```

Constraints:

- User can have max 10 decks
- Deck must have 15-30 cards
- Only one deck can be active at a time
- Duplicate cards allowed

Game Lobby Model

```
{
  "_id": "507f1f77bcf86cd799439090",
  "hostUserId": "507f1f77bcf86cd799439011",
  "hostDeckId": "507f1f77bcf86cd799439020",
  "hostCharacterId": "507f1f77bcf86cd799439060",
  "players": [
    {
      "userId": "507f1f77bcf86cd799439011",
      "deckId": "507f1f77bcf86cd799439020",
      "characterId": "507f1f77bcf86cd799439060",
      "isReady": true,
      "joinedAt": "2025-11-25T17:00:00.000Z"
    },
    {
      "userId": "507f1f77bcf86cd799439012",
      "deckId": null,
      "characterId": null,
      "isReady": false,
      "joinedAt": "2025-11-25T17:05:00.000Z"
    }
  ],
  "mapId": "507f1f77bcf86cd799439080",
  "lobbyName": "Epic Battle",
  "isPrivate": false,
  "password": null,
  "maxPlayers": 2,
  "status": "waiting",  // 'waiting', 'ready', 'started', 'cancelled'
  "gameId": null,
  "gameSettings": {
    "turnTimeLimit": 60,
    "allowSpectators": true
  },
  "createdAt": "2025-11-25T17:00:00.000Z",
  "startedAt": null,
  "expiresAt": "2025-11-25T17:30:00.000Z"  // TTL: 30 minutes
}
```

Game Model (MongoDB - Completed Games)

```
{
  "_id": "507f1f77bcf86cd799439100",
  "lobbyId": "507f1f77bcf86cd799439090",
  "players": [
    {
      "userId": "507f1f77bcf86cd799439011",
      "finalScore": 25,
      "cardsPlayed": 15
    },
    {
      "userId": "507f1f77bcf86cd799439012",
      "finalScore": 22,
      "cardsPlayed": 14
    }
  ],
  "mapId": "507f1f77bcf86cd799439080",
  "totalTurns": 20,
  "status": "completed",  // 'completed', 'abandoned', 'draw'
  "winner": "507f1f77bcf86cd799439011",
  "gameDuration": 1200,  // seconds
  "finalBoardState": { ... },
  "startedAt": "2025-11-25T18:00:00.000Z",
  "completedAt": "2025-11-25T18:20:00.000Z",
  "createdAt": "2025-11-25T18:00:00.000Z"
}
```

Inventory Model

```
{
  "_id": "507f1f77bcf86cd799439200",
  "userId": "507f1f77bcf86cd799439011",
  "cards": [
    {
      "cardId": "507f1f77bcf86cd799439050",
      "quantity": 3,
      "acquiredAt": "2025-11-25T10:00:00.000Z"
    }
  ],
  "characters": [
    {
      "characterId": "507f1f77bcf86cd799439060",
      "acquiredAt": "2025-11-25T10:00:00.000Z"
    }
  ]
}
```

Friend Request Model

```
{
  "_id": "507f1f77bcf86cd799439070",
  "fromUserId": "507f1f77bcf86cd799439011",
  "toUserId": "507f1f77bcf86cd799439012",
  "status": "pending",  // 'pending', 'accepted', 'rejected', 'cancelled'
  "respondedAt": null,
  "createdAt": "2025-11-25T16:00:00.000Z"
}
```

Error Handling

Standard Error Response Format

```
{
  "success": false,
  "message": "Error description",
  "errors": [
    {
      "field": "fieldName",
      "message": "Detailed error message"
    }
  ]
}
```

HTTP Status Codes

Code	Description	Usage
200	OK	Successful GET, PUT, PATCH, DELETE
201	Created	Successful POST (resource created)
400	Bad Request	Validation errors, invalid input
401	Unauthorized	Authentication required/failed
403	Forbidden	Authenticated but not authorized
404	Not Found	Resource doesn't exist
409	Conflict	Duplicate resource (username, email)
500	Internal Server Error	Unexpected server error
503	Service Unavailable	Database connection error

Common Error Examples

Validation Error (400)

```json
{
  "success": false,
  "message": "Validation failed",
  "errors": [
    {
      "field": "username",
      "message": "Username must be at least 3 characters long"
    },
    {
      "field": "password",
      "message": "Password must contain at least one uppercase letter"
    }
  ]
}
```

Authentication Error (401)

```json
{
  "success": false,
  "message": "Invalid or expired token"
}
```

Authorization Error (403)

```json
{
  "success": false,
  "message": "You do not have permission to access this resource"
}
```

Not Found Error (404)

```json
{
  "success": false,
  "message": "Deck not found"
}
```

Duplicate Error (409)

```json
{
  "success": false,
  "message": "Username already exists"
}
```

Server Error (500)

```json
{
  "success": false,
  "message": "An unexpected error occurred. Please try again later."
}
```

Appendix

Rate Limiting

Currently no rate limiting is implemented. Consider implementing:

- Authentication endpoints: 5 requests per minute
- Gacha endpoints: 10 requests per minute
- Game actions: 30 requests per minute

Pagination

For endpoints returning lists, use:

- `limit` : Number of results (default: 10-20)
- `skip` : Number to skip (default: 0)
- `offset` : Alternative to skip

WebSocket Reconnection

Clients should implement:

- Automatic reconnection with exponential backoff
- Token refresh on 401 errors
- State recovery on reconnection

Security Best Practices

1. Always use HTTPS in production
2. Store refresh tokens securely (HttpOnly cookies or secure storage)
3. Implement CSRF protection for cookie-based auth

4. Validate all user input on client and server
5. Use Content Security Policy headers
6. Implement rate limiting

End of API Documentation