# **WebQuiz実装計画書 (完全版 v9)**

## **1. 概要**

本ドキュメントは、Node.js/Socket.IOサーバーとReactクライアントで構成されるリアルタイムWebクイズアプリケーションの実装仕様を定義する。データ構造、状態遷移、Socket.IOイベント、ゲームフロー、エラーハンドリングについて網羅的な仕様を定めることを目的とする。**v9では、状態遷移の記述を、アプリケーション全体のフローから詳細な内部状態へと階層的に再構成した。**

## **2. データ構造の設計**

### **サーバー全体のデータ構造 (概念)**

サーバーは、ルーム情報とは独立して、接続している全プレイヤーの情報を管理する。

{  
 "players": {  
 "socket\_id\_1": { "name": "プレイヤーA" },  
 "socket\_id\_2": { "name": "プレイヤーB" }  
 },  
 "rooms": {  
 "a1b2c": {  
 /\* Roomオブジェクト \*/  
 }  
 }  
}

### **Room (ルーム)**

クイズゲームのセッション全体を管理するオブジェクト。

{  
 "id": "a1b2c",  
 "hostId": "socket\_id\_of\_host",  
 "state": "waiting",  
 "players": {  
 "socket\_id\_1": {  
 "id": "socket\_id\_1",  
 "name": "プレイヤーA",  
 "score": 0  
 }  
 },  
 "gameData": {  
 "questions": [ /\* Questionオブジェクトの配列 \*/ ],  
 "currentQuestionIndex": 0,  
 "questionState": "idle",  
 "answeredPlayerIds": [],  
 "activeAnswer": null  
 }  
}

* **state**: ルームの全体的な状態 (waiting, playing, finished)。
* **gameData.questionState**: 1問ごとのライフサイクルにおける詳細な状態 (idle, presenting, reading, answering, result)。
* **gameData.answeredPlayerIds**: その問題で既にお手つきになったプレイヤーのIDリスト。
* **gameData.activeAnswer**: **誰かが回答権を得た時にのみ生成される一時的なオブジェクト。** それ以外の時はnull。  
  // activeAnswerオブジェクトの構造  
  {  
   "playerId": "socket\_id\_of\_buzzer\_winner",  
   "submittedChars": ["と", "う"]  
  }

### **Question (問題)**

クイズの問題情報を保持するオブジェクト。

{  
 "id": "q001",  
 "text": "日本の首都はどこでしょう？",  
 "answer\_data": [  
 { "char": "と", "choices": ["と", "ち", "の", "そ"] },  
 { "char": "う", "choices": ["う", "あ", "お", "え"] },  
 { "char": "き", "choices": ["き", "さ", "し", "け"] },  
 { "char": "ょ", "choices": ["ょ", "ゅ", "ゃ", "っ"] },  
 { "char": "う", "choices": ["う", "ぬ", "め", "ろ"] }  
 ]  
}

## **3. 状態遷移**

本アプリケーションの状態は、クライアント側のUI状態と、サーバー側のルーム状態の2つに大別される。

### **3.1. アプリケーション全体のフロー**

ユーザーの視点から見た画面の遷移は以下の通り。

| **現在の画面** | **トリガー** | **次の画面** | **備考** |
| --- | --- | --- | --- |
| (接続直後) | - | 名前登録 |  |
| 名前登録 | registerPlayer成功 | ホーム（ロビー） |  |
| ホーム（ロビー） | createRoom / joinRoom 成功 | ルーム待機 |  |
| ルーム待機 | leaveRoom / 接続断 | ホーム（ロビー） |  |
| ルーム待機 | ホストがstartGame | ゲーム画面 |  |
| ゲーム画面 | ゲーム終了 (gameFinished) | 全体結果画面 |  |
| ゲーム画面 | ホスト切断 (roomClosed) | ホーム（ロビー） | 強制的にロビーに戻される。 |
| 全体結果画面 | (自動タイマー) | ルーム待機 | ゲーム終了後、同じメンバーで待機状態に戻る。 |

### **3.2. サーバー側ルーム状態 (Room.state)**

Roomオブジェクトのstateプロパティの遷移。

| **現在の状態** | **トリガーイベント** | **次の状態** | **説明** |
| --- | --- | --- | --- |
| (存在しない) | createRoom | waiting | ルームが生成され、待機状態になる。 |
| waiting | ホストがstartGame | playing | ゲームが開始される。 |
| playing | ゲーム終了条件を満たす | finished | 全ての問題が終了し、結果表示状態になる。 |
| finished | (自動タイマー) | waiting | 全体結果表示後、再び待機状態に戻る。 |
| waiting/playing/finished | ホストが切断 | (ルーム削除) | ルームは即座に削除される。 |

### **3.3. 1問ごとのライフサイクル (gameData.questionState)**

Room.stateがplayingの間に、gameData.questionStateは1問ごとに以下のライフサイクルを繰り返す。

| **現在のサブ状態** | **トリガー** | **次のサブ状態** | **説明** |
| --- | --- | --- | --- |
| idle | (ゲーム進行ロジック) | presenting | **新しい問題のライフサイクル開始。** |
| presenting | (自動) | reading | 問題の開始合図表示後、問題読み上げへ。 |
| reading | プレイヤーがbuzz | answering | 早押し成功。activeAnswerが生成される。 |
| reading | (時間切れ) | result | 誰も早押しせず問題終了。**ライフサイクル完了。** |
| answering | submitCharacter (正解/続きあり) | answering | 次の文字の選択肢を提示。 |
| answering | submitCharacter (全問正解) | result | 正解！activeAnswerは破棄。**ライフサイクル完了。** |
| answering | submitCharacter (不正解/他者あり) | reading | 回答権を失う。activeAnswerは破棄され、問題読み上げ再開。 |
| answering | submitCharacter (不正解/他者なし) | result | 全員お手つき。activeAnswerは破棄。**ライフサイクル完了。** |
| result | (自動タイマー) | idle | 結果表示後、次の問題の準備へ移行。 |

## **4. Socket.IO イベント定義**

### **4.1. クライアント → サーバー (C → S)**

| **イベント名** | **データ (Payload)** | **説明** |
| --- | --- | --- |
| registerPlayer | { playerName: string } | プレイヤー名をサーバーに登録する。 |
| createRoom | (なし) | 新しいルームの作成を要求する。サーバーは送信元IDからプレイヤーを特定する。 |
| joinRoom | { roomId: string } | 既存のルームへの参加を要求する。 |
| leaveRoom | { roomId: string } | 現在のルームから退室する。 |
| startGame | { roomId: string } | **ホストのみ**がゲームの開始を要求する。 |
| buzz | { roomId: string } | 早押しボタンを押したことを通知する。 |
| submitCharacter | { roomId: string, selectedChar: string } | 回答者が選択肢から**1文字**選択したことを通知する。 |
| requestNextQuestion | { roomId: string } | （任意）結果表示後、次の問題へ進むよう要求する。 |

### **4.2. サーバー → クライアント (S → C)**

| **イベント名** | **データ (Payload)** | **説明** |
| --- | --- | --- |
| playerRegistered | { playerName: string, playerId: string } | プレイヤー名の登録が成功したことを本人に通知する。 |
| roomListUpdate | [ { id, playerCount, state }, ... ] | **ロビーの全クライアント**にルーム一覧を送信。 |
| joinedRoom | { room: Room, playerId: string } | **参加者本人**にルーム情報と自身のIDを送信。 |
| roomUpdated | { room: Room } | **ルーム内の全プレイヤー**に更新されたルーム情報をブロードキャスト。 |
| roomClosed | { roomId: string, reason: string } | ルームが解散したことを通知する。（例: reason: "ホストが切断しました。"） |
| gameStarted | { room: Room } | **ルーム内の全プレイヤー**にゲーム開始を通知。 |
| newQuestion | { question: Question, questionIndex: number, room: Room } | **ルーム内の全プレイヤー**に新しい問題を出題。 |
| readingStarted | { room: Room } | **ルーム内の全プレイヤー**に問題読み上げ開始を通知。 |
| buzzerResult | { winnerId: string, room: Room } | **ルーム内の全プレイヤー**に早押し成功者を通知。 |
| nextChoice | { choices: string[] } | **回答者本人**に、回答すべき次の文字の選択肢を送信。 |
| answerResult | { playerId: string, isCorrect: boolean, isFinal: boolean, correctAnswer?: string[] } | **ルーム内の全プレイヤー**に回答試行の結果を通知。 |
| scoreUpdated | { players: Player[] } | **ルーム内の全プレイヤー**に更新されたスコア情報を通知。 |
| gameFinished | { room: Room } | **ルーム内の全プレイヤー**にゲーム終了と最終結果を通知。 |
| errorOccurred | { code: string, message: string } | **特定のクライアント**にエラーを通知。 |

## **5. ゲームフローシーケンス**

### **5.1. 接続とプレイヤー名登録**

1. **クライアント**がサーバーに接続する。
2. クライアントのUIは「名前登録画面」を表示する。
3. ユーザーが名前を入力し、送信する。
4. **クライアント**はregisterPlayerイベントをサーバーに送信する。
5. **サーバー**は名前を検証し、送信元のsocket.idに関連付けて保存する。
6. **サーバー**はplayerRegisteredイベントを**送信元のクライアント**に返す。
7. クライアントはplayerRegisteredを受け取り、プレイヤー情報を内部に保存して「ホーム画面（ロビー）」へ遷移する。

### **5.2. ルーム作成からゲーム開始まで**

1. **クライアントA**が createRoom を送信。
2. **サーバー**は送信元のsocket.idからプレイヤー情報（名前など）を取得し、Roomを作成。クライアントAをホストとして追加する。
3. **サーバー**は**クライアントA**に joinedRoom を送信。
4. **サーバー**は**ロビーの全クライアント**に roomListUpdate をブロードキャスト。
5. **クライアントB**が joinRoom を送信。
6. **サーバー**は送信元のsocket.idからプレイヤー情報を取得し、指定されたRoomに追加する。
7. **サーバー**は**クライアントB**に joinedRoom を送信。
8. **サーバー**は**ルーム内の全プレイヤー**（AとB）に roomUpdated をブロードキャスト。
9. \*\*ホスト（クライアントA）\*\*が startGame を送信。
10. **サーバー**はRoom.stateをplayingに変更し、gameStarted をブロードキャスト。その後、最初の問題のライフサイクルを開始する（5.3へ）。

### **5.3. 問題回答サイクル (詳細版)**

**▼ 新しい問題のライフサイクル開始 ▼**

1. **サーバー**はgameData内のquestionStateをpresentingに設定し、answeredPlayerIdsを空にし、activeAnswerをnullに設定する。更新されたRoomオブジェクトを含むnewQuestionをブロードキャスト。
2. **クライアント**は「問題！」などの開始合図を表示。
3. 数秒後、**サーバー**はquestionStateをreadingに設定し、readingStartedをブロードキャスト。
4. **クライアント**は問題文の表示を開始し、早押しボタンを有効化する。

#### **5.3.1. 早押し成功から回答まで**

1. **プレイヤーC**がbuzzを送信。
2. **サーバー**はquestionStateをansweringに変更し、gameData.activeAnswerオブジェクトを{ playerId: "CのID", submittedChars: [] }として**生成**する。buzzerResultをブロードキャスト。
3. **サーバー**は**プレイヤーC**（activeAnswer.playerIdで識別）に最初の文字のnextChoiceを送信。
4. **プレイヤーC**がsubmitCharacterで文字を送信。
5. **サーバー**は回答を検証。
   * **正解で、まだ続きがある場合**:
     + activeAnswer.submittedCharsに正解文字を追加。
     + サーバーは**プレイヤーC**に次の文字のnextChoiceを送信。8に戻る。
   * **全問正解の場合**:
     + サーバーはスコアを加算。answerResult({ isCorrect: true, isFinal: true, ... })とscoreUpdatedをブロードキャスト。10へ進む。
   * **不正解の場合**:
     + プレイヤーCの回答権を剥奪（answeredPlayerIdsに追加）。
     + **まだ回答権を持つプレイヤーがいる場合**: answerResult({ isCorrect: false, isFinal: false, ... })をブロードキャスト。activeAnswerをnullに**リセット**し、questionStateをreadingに戻してreadingStartedで問題読み上げを再開。4に戻る。
     + **全員お手つきの場合**: answerResult({ isCorrect: false, isFinal: true, ... })をブロードキャスト。10へ進む。

#### **5.3.2. 問題の決着と次のステップ**

1. サーバーはquestionStateをresultに設定し、activeAnswerがnullであることを保証する。クライアントは1問ごとの結果を表示。  
   ▲ 現在の問題のライフサイクル完了 ▲
2. 数秒後、**サーバー**はゲーム終了条件（規定問題数、規定スコアなど）を判定する。
   * **ゲーム続行の場合**: **サーバーは次の問題の準備をし、ステップ1から新しい問題のライフサイクルを開始する。**
   * **ゲーム終了の場合**: Room.stateをfinishedに設定し、gameFinishedをブロードキャスト。

### **5.4. プレイヤーの切断処理**

以下の処理は、サーバーがクライアントとの接続断を検知した際（Socket.IOの標準disconnectイベント）に実行される。

1. **サーバー**は、切断したクライアントのsocket.idを取得する。
2. サーバーは、そのsocket.idを元に、サーバー全体のplayersリストから該当プレイヤーの情報を削除する。
3. サーバーは、そのプレイヤーが参加していたルームを検索する。
4. **ルームに参加していた場合**:
   * **ホストが切断した場合**:
     + **ルーム内の残りの全プレイヤー**にroomClosedイベントを送信し、ルームが解散したことを通知する。
     + サーバーのroomsリストから、そのルームを完全に削除する。
   * **ホスト以外のプレイヤーが切断した場合**:
     + ルームのplayersオブジェクトから、該当プレイヤーを削除する。
     + **ルーム内の残りの全プレイヤー**にroomUpdatedイベントをブロードキャストする。
   * 最後に、**ロビーにいる全クライアント**にroomListUpdateイベントをブロードキャストし、ルームリストの変更を反映させる。
5. **ルームに参加していなかった場合**（ロビーにいただけの場合）は、プレイヤーリストからの削除のみで処理は完了する。

## **6. エラーハンドリング**

クライアントへの通知が必要なエラーケースと、errorOccurred イベントで送信する内容を定義する。

| **エラーコード** | **メッセージ** | **説明** |
| --- | --- | --- |
| ROOM\_NOT\_FOUND | "ルームが見つかりません。" | 存在しないroomIdで参加しようとした。 |
| ROOM\_FULL | "このルームは満員です。" | 満員のルームに参加しようとした。 |
| INVALID\_NAME | "無効なプレイヤー名です。" | プレイヤー名が空、または長すぎる。 |
| NOT\_HOST | "ホストではありません。" | ホスト以外のプレイヤーがstartGameを送信した。 |
| ALREADY\_PLAYING | "ゲームはすでに開始されています。" | ゲーム中のルームに参加しようとした。 |
| NOT\_REGISTERED | "プレイヤー名が登録されていません。" | 名前登録をせずにルーム作成/参加を試みた。 |