ChatGPT

토너먼트 관리 시스템 기능 상세 기획

1. 참가자 관리 (Participant Management)

기능 요약: 대회 참가자들의 정보를 등록/수정/삭제하고 현재 참가 상태(참여 중 또는 탈락)를 관리합니다. 각 참가자의 칩 스택(칩 보유량) 정보를 실시간으로 추적하여 전체 칩 합계와 평균 스택 등을 계산합니다.

UI 요소: - **참가자 목록 테이블:** 이름, 연락처, 칩 개수, 상태(예: 참여 중, 탈락), 테이블/좌석 등이 열로 표시됩니다. - **참가자 등록/편집 폼:** 신규 참가자 등록 및 기존 정보 수정은 모달 폼으로 구현합니다. - **상태 필터:** 참가자 상태별(전체, 참여 중, 탈락)로 목록을 필터링하는 기능을 제공합니다. - **요약 표시:** 대시보드 상단이나 테이블 하단에 현재 남은 참가자 수와 전체 칩 합계, 평균 스택 등을 실시간으로 보여줍니다.

DB 구조: Firestore 컬렉션 경로는 `/tournaments/{tournamentId}/participants/{participantId}` 아래에 각 참가자 문서를 저장합니다. 문서 필드는 다음과 같습니다: - `name` (이름) - `phone` (전화번호) - `chipCount` (현재 칩 개수) - `status` (현재 상태, 예: `"active"` 또는 `"busted"` 탈락 시) - `seat` (좌석 번호) - `table` (테이블 번호)

실현 계획: 1. **CRUD 기능 구현:** React를 사용하여 참가자 목록 및 등록/편집 UI를 만들고, Firestore와 연동하여 생성/읽기/수정/삭제가 가능하도록 합니다. 2. **탈락자 처리:** 참가자가 탈락하여 `status`를 `"busted"`로 변경하면, Firestore 트리거 또는 클라우드 함수를 활용하여 남은 **playerCount**(참가자 수)를 감소시키는 로직을 연결합니다. 이로써 참가자 수 요약이 자동 갱신되도록 합니다. 3. **칩 스택 계산:** 참가자 추가/탈락 또는 chipCount 변경 시마다 전체 칩 합계와 **평균 스택**(= 총 칩 / 남은 참가자 수)을 재계산하는 로직을 구현합니다. 이러한 통계는 화면의 요약 영역에 실시간으로 표시되며, Firestore의 실시간 업데이트 리스너를 통해 자동 갱신되도록 합니다.

2. 테이블/좌석 자동 배정 (Table & Seat Assignment)

기능 요약: 참가자 인원에 따라 자동으로 테이블과 좌석을 배정하는 기능입니다. 기본적으로 한 테이블에 최대 9명의 플레이어가 앉도록 하고, 참가자 탈락 등 인원 변화 시 테이블을 재배치하거나 통합(**테이블 축소**)하여 균형을 맞춥니다.

UI 요소: - **테이블 현황 목록:** 각 테이블 번호별로 좌석 배치와 해당 좌석의 선수 이름을 보여주는 UI를 구성합니다. - **자동 배정 버튼:** 참가자 등록 후 자동 좌석 배정 기능을 제공하여 클릭 시 모든 참가자를 무작위로 테이블과 좌석에 앉힙니다. - **수동 Drag & Drop 이동:** 테이블 간에 참가자를 수동으로 이동시킬 수 있도록 드래그앤드롭 인터페이스를 지원합니다 (라이브러리 예: React-beautiful-dnd). - **테이블 상태 표시:** 각 테이블에 현재 앉은 인원 수를 표시하고, 비어있는 좌석은 별도 표시(예: 회색 좌석 아이콘)하여 한눈에 좌석 여유를 파악할 수 있게 합니다.

DB 구조: Firestore에서 `/tournaments/{tournamentId}/tables/{tableId}` 컬렉션에 테이블 정보를 저장합니다. 각 테이블 문서에는 다음과 같은 데이터를 갖습니다: - `players`: 해당 테이블에 앉아있는 참가자들의 ID 목록 (또는 participant 객체 목록). - `seats`: 좌석 번호 및 할당 상태 (예: 좌석 1~9, 각 좌석에 매핑된 참가자ID 또는 비어있음을 표시). - `dealerId`: 현재 딜러로 지정된 딜러의 ID (있다면). - `status`: 테이블 상태 (예: "active" = 사용 중, "closed" = 통합되어 닫힌 테이블 등).

실현 계획: 1. **최초 배정 알고리즘:** 참가자 등록이 완료되면, 무작위로 테이블과 좌석을 할당하는 알고리즘을 구현합니다. 예를 들어 참가자 리스트를 섞은 후 순서대로 테이블 1부터 N까지 9명씩 분배합니다. 좌석 번호도 1번부터 순차 배정하거나 랜덤하게 섞어 부여합니다. 2. **자동 리밸런싱:** 참가자가 탈락하거나 테이블별 인원 불균형이 발생할 때 **테이블 리밸런싱 알고리즘**을 개발합니다. 예를 들어 가장 많이 남은 테이블에서 한 명을 빼서 가장 적은 테이블에 옮기는 식으

로 자동 재배치합니다. 이를 통해 모든 테이블의 플레이어 수 차이가 1명 이내가 되도록 유지합니다. (※ 이 기능은 2차 버전에서 구현 검토하며, 단계적으로 고도화합니다.) 3. **Drag & Drop 수동 조정:** 자동 배정된 좌석을 운영자가 수동으로 옮기고 싶을 때를 대비하여, UI에서 참가자 카드를 드래그하여 다른 테이블/좌석으로 옮길 수 있게 합니다. Drag 이벤트 발생 시 해당 참가자의 `table`과 `seat` 값을 업데이트하고 Firestore에 반영하여 실시간으로 변경 사항이 전파됩니다. 4. **테이블 축소:** 남은 참가자가 줄어들어 테이블을 줄일 필요가 있을 경우(예: 남은 인원이 하나의 테이블 수용 인원 미만일 때), 자동으로 가장 높은 번호 테이블을 닫고 거기의 플레이어들을 다른 테이블로 이동시키는 로직을 추가합니다. 이 때 이동되는 플레이어의 좌석도 빈 좌석에 자동 할당하거나, 운영자가 확인 후 좌석을 조정할 수 있게 합니다.

3. 딜러 관리 및 배정 (Dealer Management & Assignment)

기능 요약: 대회를 진행할 딜러들을 사전에 등록하고, 테이블에 자동 혹은 수동으로 딜러를 배치하는 기능입니다. 또한 딜러 교대 시간에 맞춰 알림을 주거나 딜러를 교체하는 관리 기능도 포함됩니다.

UI 요소: - **딜러 리스트:** 딜러의 이름, 연락처 등의 정보를 보여주는 리스트와 신규 딜러 추가/수정/삭제 UI. - **테이블별 딜러 현황:** 각 테이블에 현재 배정된 딜러 이름 또는 아이디를 표시합니다. 딜러가 아직 지정되지 않은 테이블은 "미배정" 등으로 표시. - **딜러 배정 설정:** 드롭다운이나 드래그 방식으로 특정 딜러를 특정 테이블에 할당할 수 있는 인터페이스. 예를 들어, 딜러 이름을 테이블 박스로 드래그하여 배정하거나, 테이블 행에서 딜러를 선택하는 드롭다운. - **교대 알림:** (후속 개발) 딜러 교대 시간이 되면 관리자 화면에 알림을 띄우거나 해당 테이블 표시를 강조하여 교대 시점을 알려주는 기능.

DB 구조: 딜러 정보는 전역으로 `/staff/{staffId}` 컬렉션에 저장하고, 특정 대회에 투입된 딜러는 `/tournaments/{tournamentId}/staff/{staffId}` 하위에 배정 정보를 둘 수 있습니다. (혹은 `tournaments/{id}/tables/{tableId}`의 필드로 `dealerId`만 참조하는 현재 구조 활용) 기본 딜러 문서에는: - `name` (딜러 이름) - `phone` (연락처) - `status` (활동 상태 등, 필요시) - `assignedTable` (현재 배정된 테이블 ID, 없으면 null)

실현 계획: 1. **딜러 CRUD 구현:** React를 통해 딜러 목록 조회 및 추가/수정 폼을 제공하고 Firestore `staff` 컬렉션과 연동합니다. 관리자만 딜러 정보를 편집할 수 있도록 권한 관리도 적용합니다. 2. **테이블 자동 배정:** 대회 시작 시 등록된 딜러들을 이용해 테이블 수 만큼 자동으로 배정하는 기능을 만듭니다. 예컨대 테이블 개수와 딜러 수를 비교하여 가능한 경우 각 테이블에 한 명씩 랜덤/순차 배정합니다. (딜러 수 부족 시 일부 테이블은 공석 처리) 3. **수동 배정:** 운영자가 딜러를 원하는 테이블로 변경할 수 있도록 UI에서 지원합니다. 딜러 리스트에서 테이블 지정 필드를 수정하면 Firestore의 해당 `tables/{tableId}.dealerId`가 갱신되고 UI에 반영됩니다. 4. **딜러 교대 관리:** (추후 기능) 딜러들의 교대 시간이 일정하다면, 예를 들어 2시간마다 교대라면, 교대 5분 전 등에 관리자에게 알림을 주고 새로운 딜러로 테이블 딜러를 교체할 수 있게 합니다. 교대 시에는 현재 딜러의 `assignedTable`을 해제하고 대기 중인 딜러에게 할당을 변경합니다. 이 기능은 타이머 및 알림 시스템과 연계하여 **4단계** 개발 때 구현을 검토합니다.

4. 블라인드 타이머 및 구조표 (Blind Timer & Structure)

기능 요약: 포커 토너먼트에서 **블라인드 타이머**는 매우 중요합니다. 레벨별 블라인드(베팅 최소금/빅블라인드) 구조를 관리하고, 현재 레벨의 남은 시간을 표시하는 카운트다운 시계를 제공합니다. 레벨 종료 시 다음 블라인드로 자동 상승하고, 레벨 변경이나 휴식 시간에 알림을 주는 기능도 포함됩니다.

UI 요소: - **큰 카운트다운 시계:** 현재 레벨에서 남은 시간을 시/분/초 단위의 대형 디지털 시계로 표시합니다. (예: 00:15:30 형태로 15분 30초 남음) - **현재 블라인드 정보:** 현재 레벨 번호와 소블라인드/빅블라인드 금액, 앤티(있다면)를 화면에 크게 보여줍니다. 다음 레벨의 블라인드 정보도 작게 표시하여 미리 볼 수 있게 할 수 있습니다. - **레벨 제어 버튼:** 일시정지, 재개, 다음 레벨로 넘기기, 이전 레벨로 돌아가기 등의 수동 조정 버튼을 제공합니다. 이를 통해 비상시(예: 타이머 일시정지 필요 상황) 운영자가 개입할 수 있습니다. - **레벨 구조표 보기:** 전체 레벨 구조표(예: 레벨1:

100/200, 20min ... 레벨2: 200/400, 20min ...)를 표나 팝업으로 열람할 수 있게 합니다. 참가자들도 볼 수 있도록 별도 페이지나 모달로 제공해도 좋습니다.

DB 구조: 대회의 블라인드 구조는 `/tournaments/{tournamentId}/blindStructure` 필드 또는 하위 컬렉션으로 저장합니다. 예를 들어 `blindLevels: [ { level: 1, sb: 100, bb: 200, ante: 0, duration: 20 }, { level: 2, sb: 200, bb: 400, ante: 0, duration: 20 }, ... ]` 형태의 배열로 저장하거나, 각 레벨을 도큐먼트로 관리할 수도 있습니다. 추가로 현재 진행 중인 레벨과 남은 시간을 저장하는 필드(예: `currentLevel`, `timeRemaining`)가 `tournaments/{id}` 문서에 있을 수 있습니다.

실현 계획: 1. **타이머 UI 및 수동 제어:** 우선 React로 클라이언트 사이드에서 동작하는 블라인드 타이머를 구현합니다. 운영자는 대시보드에서 현재 레벨을 시작/정지하거나 수동으로 다음 레벨로 넘어갈 수 있습니다. 레벨 업 또는 수동 조정 시 Firestore에 현재 레벨 정보와 남은 시간을 업데이트하여 참가자용 페이지에도 실시간 반영되도록 합니다. 2. **레벨 구조 관리:** UI에서 블라인드 구조표(blindStructure)를 입력하거나 편집할 수 있게 합니다. 예를 들어 대회 생성 시 기본 구조를 입력해두고, 필요시 대회 도중에도 레벨 시간을 조정하거나 블라인드 금액을 수정할 수 있도록 만듭니다. (하지만 일반적으로 대회 중 구조 변경은 드물기에 읽기 전용으로 두고, 시작 전에만 수정 가능하게 할 수도 있습니다.) 3. **자동 레벨 전환:** (후속 개발) Cloud Functions 또는 Firestore 트리거를 활용하여 타이머 종료 시 자동으로 다음 레벨로 변경되는 기능을 구현합니다. 예를 들어 서버 시계를 기반으로 정확한 카운트다운을 하고, 시간이 다 되면 `currentLevel++` 및 `timeRemaining`을 레벨 기본 duration으로 리셋하는 등의 처리를 자동화합니다. 또한 레벨 변경 시 운영자와 참가자 화면에 알림(사운드나 화면 강조)을 주도록 합니다. 4. **휴식 시간 처리:** 레벨 구조 중간중간 휴식 시간이 있다면, 해당 레벨을 특별 표시하거나 타이머가 0이 되었을 때 "Break Time - 10분 휴식" 같은 화면을 보여주고 휴식 시간 카운트다운을 표시하도록 합니다. 휴식이 끝나면 다음 레벨로 자동 재개되는 로직을 추가합니다. (※ 휴식 시간 데이터도 blindStructure에 `type: "break", duration: 10` 등의 형태로 포함하여 관리)

5. 칩/스택 계산기 (Chip & Stack Calculator)

기능 요약: 총 칩(토너먼트 내 전체 칩량)과 **평균 스택**을 자동으로 계산하여 보여주는 기능입니다. 총 칩은 참가자 수 × 시작 칩으로 계산되며, 평균 스택은 총 칩 / 남은 참가자 수로 산출됩니다. 참가자 탈락이나 Rebuy(리바이)/Add-on으로 칩 변동이 있을 때 이 수치를 즉시 갱신합니다.

UI 요소: - **대시보드 통계 영역:** 관리자용 대시보드나 블라인드 타이머 화면의 한쪽에 "남은 참가자: X명, 총 칩: Y, 평균 스택: Z"와 같은 통계 정보를 표시합니다. - **실시간 업데이트:** 참가자 수나 칩 변동이 있을 경우 이 영역의 숫자가 애니메이션과 함께 즉시 변경되어 가시성을 높입니다. - **칩 설정 입력:** (운영자용) 대회 시작 시 시작 칩 스택과 Rebuy/Add-on 칩량 등을 설정해둘 수 있는 입력란을 제공합니다. 이를 바탕으로 시스템이 총 칩을 계산합니다.

실현 계획: 1. **기본 계산 로직:** `participants` 컬렉션의 문서 개수(또는 상태가 "active"인 참가자 수)와 대회의 시작 칩 스택 정보를 이용해 총 칩과 평균 스택을 계산하는 함수를 구현합니다. 이 함수는 Firestore의 참가자 데이터 변화에 대한 리스너 안에서 호출되어 실시간으로 값이 갱신되도록 합니다. 2. **UI 반영:** 계산된 총 칩과 평균 스택 값을 대시보드/타이머 UI의 지정된 영역에 바인딩합니다. React 상태 관리나 전역 상태 (예: Redux, Context 등)를 통해 실시간 변화가 UI에 반영되도록 합니다. 3. **Rebuy/Add-on 반영:** Rebuy나 Add-on으로 인해 총 칩이 증가하는 경우를 고려합니다. 예를 들어, 한 참가자가 재구매를 하면 새로운 칩이 투입되므로 총 칩 계산식을 단순 시작칩×초기참가자수에서 변경해야 합니다. **총 칩**을 Firestore에 별도 필드로 저장하여 업데이트하거나, 매번 참가자들의 chipCount 합계를 구하여 계산할 수도 있습니다. 구현 난이도를 고려해 **참가자 chipCount 합의 실시간 집계**로 처리하면 Rebuy/Add-on도 자동 포함됩니다. (Firestore에서 집계 기능이 없으므로 클라이언트에서 합산하거나, Cloud Function으로 트리거하여 `totalChips` 필드를 관리) 4. **예외 처리:** 평균 스택은 남은 참가자가 0이 되면 계산 불가하므로 (대회 종료 시), 해당 경우 "-"로 표시하거나 대회 종료 처리 시 통계 표시 영역을 숨기는 등의 처리도 합니다.

6. 프라이즈(상금) 자동 계산기 (Prize Payout Calculator)

기능 요약: 참가자 수와 바이인(Buy-in) 금액을 기반으로 총 상금 풀(prize pool)을 계산하고, **N위까지의 상금 분배표**를 자동으로 생성하는 기능입니다. 기본 분배 비율(예: 1위 50%, 2위 30%, 3위 20% 등)로 1차 산출하며, 운영자가 필요에 따라 수동으로 조정할 수도 있도록 합니다. 계산된 상금 구조는 참가자 실시간 페이지에도 공유되어 투명성을 제공합니다.

UI 요소: - **상금 풀 입력:** 대회 생성 시 총 상금 풀 또는 1인당 바이인 금액과 참가자 수를 입력/확인할 수 있는 필드. (Rebuy/Add-on이 있는 경우 해당 금액도 포함하여 총 상금 자동계산 가능) - **자동 분배표:** "상금 계산" 버튼 클릭 시 1위부터 N위까지 상금 분배 내역을 표 형태로 보여줍니다. 예: 1위 - 50%, 2위 - 30%, 3위 - 20% 식으로 비율 기반 계산 결과 금액을 표시. - **수동 커스터마이징:** 운영자가 특별한 분배 구조를 원할 경우 표 내의 금액을 직접 수정 가능하도록 합니다. 또는 미리 정의된 **분배 템플릿**을 선택 (예: Top 10%에게 분배, 우승상금 보너스 등). - **확정 및 저장:** 최종 확정된 상금 분배안을 저장하는 버튼. 저장 시 해당 정보를 Firestore에 기록하고, 참가자용 페이지의 "상금 구조" 섹션에 반영합니다.

실현 계획: 1. **자동 계산 로직:** 우선 간단한 **비율 기반 분배** 알고리즘을 구현합니다. 예를 들어 상위 3명에게만 상금을 준다면 1위 50%, 2위 30%, 3위 20% 등 비율을 정해 총 상금 풀에 곱해 각 순위별 상금을 산출합니다. 분배 인원 N은 참가자 수에 따라 동적으로 결정하거나, 운영자가 설정할 수 있게 합니다 (예: 100명 이상 참가시 10위까지 지급 등). 2. **UI 구현:** 상금 계산 결과를 보여주는 표를 생성하고, 각 셀에 금액을 표시합니다. 소수점이 나올 경우 반올림 처리하고, 합계 검증(모든 순위 상금의 합이 총 상금과 일치하는지)도 합니다. 필요한 경우 1원 단위 조정을 위해 일부 순위 금액을 ±1 조정할 수 있습니다. 3. **수동 조정 기능:** 각 순위 금액 필드는 기본은 읽기전용으로 보여주되, "편집 모드" 전환 시 입력 가능한 상태로 변경합니다. 운영자가 금액을 변경하면 나머지 필드에 자동으로 분산되는 로직은 복잡할 수 있으므로, 편집 모드에서는 자동 계산을 off하고 사용자가 직접 모든 금액을 조정하도록 안내합니다. 4. **결과 저장:** 결정된 payout 구조를 Firestore의 `/tournaments/{id}/payouts` 하위에 저장하거나, tournament 문서에 `payouts: [{rank:1, amount: X}, ...]` 형태로 저장합니다. 이 정보는 대회 종료 후 **결과 기록**에도 활용되고, 참가자들에게는 읽기 전용으로 제공됩니다. 5. **향후 발전:** 추후 구현 단계에서 다양한 상금 분배 **템플릿**(예: 전형적인 WSOP 스타일 분배, or 정률 분배 등)을 제공하고, UI에서 버튼 하나로 해당 비율을 적용할 수 있게 할 계획입니다.

7. 대회 기록 및 히스토리 저장 (Tournament History & Records)

기능 요약: 완료된 대회의 주요 결과와 데이터를 저장하여, 나중에 조회하거나 리포트를 출력할 수 있게 합니다. 각 대회의 **히스토리 상세 페이지**에서는 최종 순위, 상금 분배, 참가자 명단 및 탈락 순서, 대회 진행 중 주요 이벤트 등이 포함될 수 있습니다. 또한 필요에 따라 PDF 또는 Excel로 내보내는 기능을 제공하여 오프라인 보고서나 기록 보관을 돕습니다.

UI 요소: - **히스토리 목록:** 과거에 완료된 대회들의 리스트를 보여주는 화면을 구성합니다. 여기에는 대회 이름, 날짜, 참가자 수, 우승자 이름, 등 간략 정보가 표시됩니다. - **대회 상세 정보 페이지:** 특정 완료된 대회를 클릭하면 상세 페이지를 보여줍니다. 주요 내용으로 우승자를 포함한 상위 입상자 목록과 상금, 전체 참가자 목록과 성적, 대회 진행 시간, 구조(블라인드, 레벨) 등 기록을 표시합니다. - **내보내기 버튼:** 히스토리 상세 페이지에서 "PDF 다운로드" 또는 "Excel로 내보내기" 버튼을 제공하여, 해당 대회의 데이터를 정리된 양식으로 출력합니다. 예를 들어, 토너먼트 결과 리포트를 PDF로 생성하거나, Excel 파일로 생성하여 더 분석할 수 있도록 합니다.

DB 구조: 대회 종료 시 관련 데이터를 종합하여 Firestore에 기록합니다. 예를 들어 `/tournaments/{tournamentId}` 문서 내에 `completed: true` 및 `endedAt`(종료시간) 등의 필드를 업데이트하고, `/tournaments/{tournamentId}/results` 혹은 별도 `history` 컬렉션에 필요한 상세 정보를 저장합니다. 가능한 데이터 구조 예: - `tournaments/{id}` 문서: `status: "completed"`, `winner: {participantId}`, `endTime`, `duration`, `totalEntries`, `totalPrize` 등 요약 정보를 기입. - 하위 컬렉션 `participants`는 원래 존재하므로 그대로 둔다. - 새로운 하위 컬렉션 `results`: 순위별 결과를 저장 (예: rank 1~N의 participantId, name, prize 등). - `payouts` 정보도 이미 위에서 저장한 것을 활용하거나 `results`에 포함.

실현 계획: 1. **자동 기록 저장:** 대회가 종료되는 시점을 트리거로 하여(예: 관리자가 "대회 종료" 버튼 클릭), 해당 대회의 모든 필요한 데이터를 가져와 Firestore에 저장합니다. 이때 실시간으로 쓰던 컬렉션을 그대로 둘 수도 있으나, 히스토리 용으로 복사본을 남기는 것이 데이터 무결성과 이후 통계에 유리합니다. 예를 들어 Cloud Function을 통해 `participants` 목록과 `payouts` 등을 읽어와 `history` 관련 경로에 복제 저장합니다. 2. **히스토리 UI 개발:** 새로운 메뉴/탭으로 "히스토리"를 추가하고, 과거 대회 목록을 불러옵니다. Firestore에서 `status: "completed"`인 tournament 문서들의 주요 필드를 리스트로 보여줍니다. 목록은 날짜 순 또는 이름순으로 정렬 기능을 제공하고, 검색 기능도 고려합니다 (예: 대회 이름으로 검색). 3. **상세 페이지:** 사용자가 히스토리 목록에서 하나의 대회를 클릭하면, 해당 대회의 상세 데이터(우승자, 순위표, 상금 구조, 레벨 구조, 참가자 최종 칩 카운트 등)를 보여주는 페이지로 이동합니다. 필요한 경우 Firestore에서 `results`나 `participants` 하위 데이터를 읽어서 화면에 렌더링합니다. 4. **PDF/Excel 내보내기:** SheetJS, pdfMake 등의 라이브러리를 이용하여 상세 페이지 내용을 문서화합니다. **pdfMake**를 사용하면 브라우저에서 바로 PDF를 생성할 수 있고, **SheetJS**(xlsx)는 Excel 파일 생성을 도와줍니다. "PDF 다운로드" 클릭 시 해당 대회의 상위 입상자 결과, 주요 통계를 PDF로 다운받도록 구현하고, "Excel 내보내기" 클릭 시 전체 참가자 리스트와 순위를 포함한 시트가 생성되도록 합니다. (이 기능은 5단계 이후 고도화로 계획합니다.)

8. 참가자용 실시간 대회 현황 페이지 (Real-time Player Info Page)

기능 요약: 참가자들이 자신의 휴대폰 등으로 대회 진행 상황을 실시간 조회할 수 있는 전용 페이지를 제공합니다. 이 페이지에는 현재 블라인드 레벨과 남은 시간, 현재 참가자 수, 평균 스택, 상금 구조 요약, 공지사항, 그리고 **본인의 좌석 정보** 등이 표시됩니다. 참가자는 이를 통해 진행 상황을 쉽게 파악하고, 자리 이동이나 공지가 있을 경우 즉각 확인할 수 있습니다.

UI 요소: - **모바일 최적화 레이아웃:** 참가자들은 주로 모바일 기기로 볼 것이므로 반응형 디자인으로 구성합니다. 주요 정보들을 아이콘과 함께 카드 형태로 표시하여 가독성을 높입니다. (예: 남은 인원: 50명, 💰 평균 스택: 30,000 등) - **블라인드/타이머 정보:** 현재 레벨과 블라인드 (예: "Level 5: 500/1000 (Ante 100)"), 남은 시간 타이머를 표시합니다. 참가자 페이지에서는 운영자 페이지보다 간소화된 디자인으로, 큰 글씨로 핵심만 보여줍니다. - **상금 구조:** 상위 몇위까지 얼마의 상금을 받는지 요약하여 보여줍니다. (예: "1위: 5,000,000원, 2위: 3,000,000원, ...") - **좌석 및 테이블:** 로그인이나 QR 스캔 등을 통해 본인을 인증하면, 현재 본인이 앉은 테이블 번호와 좌석 번호를 크게 표시합니다. 만약 자리 이동 공지가 있었다면 변경된 자리 정보를 실시간 반영합니다. - **공지사항 배너:** 운영자가 공지를 올리면 참가자 페이지 상단에 배너나 팝업으로 해당 내용이 표시됩니다. 예: "15분 후에 디너 브레이크 있습니다" 등의 공지. - **다크 모드 및 접근성:** 조명 환경이 어두운 카지노/대회장 환경을 고려하여 다크 모드를 지원하고, 텍스트 크기 조절 등 접근성 옵션을 제공합니다.

실현 계획: 1. **실시간 연동:** Firestore의 대회 관련 데이터에 listener를 설정하여, 블라인드 레벨이나 남은 시간, 참가자 수 등의 변화가 있을 때 참가자 페이지의 상태를 즉시 업데이트합니다. 예를 들어 `onSnapshot`으로 `tournaments/{id}` 문서의 relevant 필드(예: currentLevel, timeRemaining, playerCount 등)를 구독합니다. 2. **페이지 접근:** 참가자별 개별 로그인을 요구하지 않고, **공용 현황 페이지**로 디자인하되 본인 좌석 정보 등 민감한 부분은 인증이 필요한 형태로 만듭니다. 예를 들어 페이지 로드 시 좌석 QR코드를 스캔하거나 본인 전화번호를 입력하여 본인 확인을 거치면, Firestore에서 해당 참가자의 seat/table 정보를 찾아서 보여주는 흐름입니다. QR코드는 참가자 등록 시 각 참가자ID에 대한 QR을 생성하여 배포해두고, 스캔 시 해당 ID로 좌석 정보 조회하도록 구현할 수 있습니다. 3. **UI/UX 최적화:** 모바일 환경에서 한눈에 핵심 정보가 들어오도록 UX를 설계합니다. 여러 정보를 스크롤 없이 보기 위해 카드 레이아웃을 사용하고, 반응형으로 가로 2열 또는 1열로 재배치되게 합니다. 또한 업데이트가 자주 일어나는 타이머의 경우 클라이언트에서도 1초 간격 감소 표시를 하고, 정기적으로 서버 시간과 sync를 맞춥니다. 4. **다크모드 적용:** CSS를 통해 다크 테마를 지원하고, 참가자가 선택하거나 자동으로 환경에 맞춰 다크/라이트를 전환할 수 있게 합니다. 시각적 편의와 접근성을 위해 명도 대비를 충분히 하고, 중요한 정보는 색상 뿐만 아니라 아이콘/텍스트로도 구분되게 디자인합니다.

9. 공지 및 규정/구조 안내 (Announcements & Rules)

기능 요약: 대회 도중 또는 전에 운영자가 전달하고 싶은 **공지사항**을 참가자들에게 실시간으로 보여주는 기능과, 대회의 **규정**이나 **블라인드 구조표**를 안내하는 정적 정보 페이지를 제공합니다. 예를 들어 토너먼트 규칙, 예의 사항, 블라인드 상승 구조, 휴식 시간표 등을 참가자들이 언제든 확인할 수 있게 합니다.

UI 요소: - **공지 등록 UI:** 운영자 대시보드에 공지사항을 입력하여 게시할 수 있는 폼을 제공합니다. 텍스트 및 간단한 이모티콘, 강조 표시 등을 지원하고, 필요한 경우 긴 글은 링크로 대신할 수도 있게 합니다. - **참가자 페이지 배너:** 앞서 참가자용 페이지 상단에 공지사항을 띄운다고 했듯, 새로운 공지가 올라오면 상단에 눈에 띄는 배너를 표시합니다. 배너에는 공지 요약과 함께 "닫기" 또는 "자세히" 버튼이 있어 사용자가 자유롭게 볼 수 있게 합니다. - **규정/구조 안내 페이지:** 참가자용 페이지 또는 앱 내에 별도의 섹션으로 대회 구조와 규칙을 볼 수 있는 페이지를 둡니다. 여기에는 **블라인드 레벨 구조표**, **상세 대회 규정**, **예상 일정보기** 등이 포함됩니다. 이 페이지는 스크롤 가능한 문서 형식이거나 FAQ 형식으로 제작합니다.

DB 구조: 공지사항은 Firestore의 `/tournaments/{tournamentId}/announcements/{announcementId}` 컬렉션에 저장합니다. 각 공지 문서는: - `message` (공지 내용) - `createdAt` (작성 시각) - `type` (공지 유형 - 일반공지, 경고, 브레이크 안내 등 구분 가능) - `active` (현재 표시 중인지 여부, 지난 공지는 false 처리 등)

규정 및 구조 안내는 변경이 적은 정적 정보이므로 Firestore에 문서로 넣을 수도 있고, 클라이언트에 하드코딩하거나 호스팅의 정적 파일로 제공할 수도 있습니다. 유연성을 위해 `/tournaments/{id}/info/rules`, `.../blindStructure` 등을 문서로 만들어 관리할 수 있습니다.

실현 계획: 1. **공지 작성/배포:** 운영자 UI에서 공지를 작성하고 "발송"하면 Firestore `announcements` 컬렉션에 새 문서를 추가합니다. Firestore의 실시간 listener를 통해 참가자 페이지에서는 새로운 공지 문서가 추가되었음을 감지하고 화면에 배너로 띄웁니다. 일정 시간(예: 10초) 동안 배너를 보여준 뒤 자동 숨김 처리하거나, 사용자가 수동 닫을 때까지 유지합니다. 2. **공지 관리:** 운영자는 활성 공지들을 관리할 수 있어야 합니다. 예를 들어 공지 리스트를 보고 필요시 특정 공지를 종료(배너 내리기)하거나 수정/삭제할 수 있게 합니다. `active` 필드를 두어 false로 바꾸면 더 이상 참가자 화면에 보이지 않도록 구현합니다. 3. **규정/구조 제공:** 대회 시작 전에 규정과 구조를 입력하거나 업로드해둘 수 있는 인터페이스를 제공합니다. 예를 들어 텍스트 에디터나 Markdown을 입력하는 필드로 규칙을 작성할 수 있게 하거나, CSV/Excel로 블라인드 구조를 업로드하여 blindStructure에 반영하도록 할 수 있습니다. 이 데이터를 참가자 페이지의 규정 섹션에서 표시합니다. 4. **정적/동적 컨텐츠 결정:** 만약 규정이나 구조표가 모든 대회에 공통이라면 앱 빌드 시에 포함된 정적 페이지로 두고, 각 대회별 변동 사항(예: 레벨 시간)은 대회 데이터에서 가져오는 혼합 방식도 고려합니다. 반대로 대회마다 규칙이 많이 다르다면 Firestore에 모두 넣어 관리하는 편이 좋습니다. 개발 초기에는 간단히 정적 페이지로 제공하고, 추후 필요 시 동적으로 변경 가능한 방식으로 개선합니다.

10. 운영자 설정 및 보안 (Admin Settings & Security)

기능 요약: 대회 운영자를 위한 **관리자 로그인** 및 **권한 관리** 기능입니다. 운영자는 새로운 대회를 생성하거나 기존 대회를 관리할 수 있고, 일반 참가자나 딜러 등은 권한에 따라 제한된 기능만 사용하도록 제어합니다. Firebase 인증(Firebase Auth)을 통해 사용자 인증을 하고, **역할 기반 접근 제어**(Role-Based Access Control)를 적용하여 보안 규칙을 강화합니다.

실현 계획: - **관리자 로그인:** Firebase Authentication의 이메일/패스워드 또는 기타 수단을 통해 관리자 계정을 로그인합니다. 처음엔 단순하게 관리자 계정을 하드코딩하거나 Firestore에 `users` 컬렉션으로 만들어 체크할 수 있지만, Firebase Auth의 Custom Claims를 이용하면 보다 안전하게 역할 부여가 가능합니다. - **권한 부여:** 관리자 계정에는 `role: "admin"`과 같은 커스텀 클레임이나 Firestore 필드로 권한을 표기하고, 딜러계정에는 `role: "dealer"` 등을 지정합니다. 이렇게 하면 클라이언트 앱에서 해당 역할에 맞는 화면이나 기능만 활성화하고, Firestore **보안 규칙**을 통해서도 권한을 검증할 수 있습니다 (예: `request.auth.token.role == 'admin'`인

경우에만 쓰기 허용 등). - **대회 생성 및 관리:** 관리자 권한이 있는 사용자만 새로운 대회를 생성할 수 있고, 진행 중인 대회의 설정(블라인드 구조 수정, 공지 작성 등)을 바꿀 수 있게 합니다. 딜러 권한 사용자는 참가자 조회나 간단한 진행 상태 확인은 가능하지만, 주요 변경 작업은 제한합니다. 일반 참가자는 기본적으로 쓰기 동작이 필요 없으므로 읽기 전용 접근만 허용합니다. - **Firebase 보안 규칙 설정:** Firestore 보안 규칙을 작성하여 `/tournaments/{id}/...` 경로에 대한 쓰기 권한을 관리자에게만 부여합니다. 예를 들어 참가자 컬렉션에 `.write` 규칙으로 `auth.token.role == 'admin'` (또는 해당 tournament의 owner UID 비교) 등을 설정하고, 읽기는 참가자도 할 수 있게 공개 범위를 조절합니다. 또한 `/staff`나 `/announcements` 등도 유사하게 관리자만 생성/수정 가능하게 제한합니다. - **기타 설정 UI:** 관리자 패널에 대회 환경 설정(예: 기본 블라인드 템플릿 선택, 재구매 허용 여부, 자동 알림 on/off 등)을 조정할 수 있는 화면을 둡니다. 이는 Firestore의 tournament 문서 필드에 저장되어 대회별로 설정 값을 가지게 합니다. 예를 들어 `allowRebuys: true/false`, `breakInterval: 120` (분 단위 휴식주기) 등의 설정을 저장하고, 다른 기능 구현 시 이 값을 참고하여 동작을 달리하도록 합니다.

11. 리바이 및 애드온 관리 (추가 제안 기능: Rebuy/Add-on Management)

기능 요약: 토너먼트 진행 중 참가자가 **Rebuy**(재구매)나 **Add-on**을 통해 칩을 추가 구매하는 경우를 관리하는 기능입니다. 리바이/애드온 횟수 제한, 추가 구매로 인한 칩량 증가 처리, 그리고 총 상금 풀 및 칩 합계 조정 등이 포함됩니다. 이 기능을 통해 운영자는 현장에서 추가 구매 상황을 시스템에 반영하고, 참가자 페이지와 통계에 즉각 반영할 수 있습니다.

UI 요소: - **추가 구매 버튼:** 참가자 목록 테이블에서 각 참가자 행에 "Rebuy" 또는 "Add-on" 버튼을 제공합니다. 운영자가 해당 참가자가 재구매할 때 버튼을 누르면 자동으로 그 참가자의 chipCount에 시작 칩만큼 추가되도록 합니다. Add-on도 유사하게 처리합니다. - **추가 구매 기록 팝업:** Rebuy/Add-on 버튼 클릭 시 확인 창을 띄워 몇 번째 리바이인지, 칩 몇 개를 추가했는지 등을 확인 및 기록합니다. 제한 횟수를 초과한 경우 경고를 표시하고 거부합니다. - **요약 정보 업데이트:** 추가 구매 발생 시 총 칩, 평균 스택, 총 상금 등이 변동되므로, 해당 수치들을 즉시 재계산하여 UI에 업데이트합니다. 또한 상금 분배도 **총 상금 풀**이 증가했다면 재계산을 고려해야 합니다.

DB 구조: 참가자의 문서에 `chipCount`를 갱신하는 것으로 칩 증가는 처리됩니다. 추가로 `rebuyCount`, `addonCount` 등의 필드를 participant 문서에 두어 해당 참가자의 재구매 횟수를 추적할 수 있습니다. 상금 풀 증액을 위해 대회 문서에 `totalBuyIns` 또는 `prizePool` 필드를 두고, Rebuy/Add-on 발생 시 해당 값을 증가시키는 방법도 고려합니다. 예를 들어: - `tournaments/{id}.prizePool`: 초기엔 참가자 수×바이인으로 설정, 리바이 발생 시 +바이인 금액씩 증가. - 또는 별도 컬렉션 `/tournaments/{id}/buyins`에 각 구매 이벤트를 기록 (누가 언제 Rebuy 했는지 로그 남기기 위함).

실현 계획: 1. **추가 구매 처리 로직:** 운영자가 Rebuy 혹은 Add-on 입력을 하면, 해당 참가자의 `chipCount`에 시작 칩 값을 더해줍니다. Firestore 트랜잭션을 사용하여 동시성 이슈 없이 증가시키고, `rebuyCount`도 +1 (또는 addonCount) 합니다. 2. **제한 규칙:** 대회 규정에 따라 Rebuy 가능 횟수나 Add-on 가능 시간 등을 설정할 수 있습니다. 예를 들어 레벨 6까지만 Rebuy 허용, 1인 최대 2회 Rebuy 등. 이러한 규칙을 `tournaments/{id}` 설정 필드로 만들어두고, UI에서 추가 구매 시 해당 조건을 검사하여 위반 시 오류 알림을 줍니다. 3. **통계 및 상금 처리:** Rebuy로 인해 **총 참가자 수**는 변하지 않지만 총 칩은 증가하므로 평균 스택 계산에 영향이 있습니다. 이전 섹션의 칩 계산 로직이 참가자 chipCount 합계로 작동한다면 자동 반영될 것입니다. 또한 상금 풀을 바이인 합계로 산정하는 경우, Rebuy 발생 시 상금 풀도 증가시켜야 합니다. 이를 위해 `prizePool` 필드를 업데이트하고, 상금 분배표가 이미 확정된 상태라면 운영자에게 재계산이 필요함을 경고하거나 자동 재계산 옵션을 제공합니다. 4. **참가자 페이지 반영:** 참가자용 실시간 페이지에서도 총 참가자 수 변화는 없지만 평균 스택 정보 등이 갱신되어야 합니다. Firestore listener로 해당 값들이 업데이트되면 자동으로 표시를 바꾸고, 혹시 본인이 Rebuy한 경우라면 자신의 칩 스택이 증가했음을 확인할 수 있도록 UI상 피드백(예: 본인 좌석 정보 카드에 highlight)도 고려합니다.

12. 휴식 시간 및 일정 관리 (추가 제안 기능: Break Time Scheduling)

기능 요약: 토너먼트 진행 중 정해진 **브레이크 시간(휴식 시간)**을 관리하는 기능입니다. 보통 몇 레벨마다 5~15분의 휴식이 주어지는데, 이를 시스템상에 설정하고, 타이머와 연동하여 자동으로 휴식 카운트다운을 표시하거나 공지하는 역할을 합니다. 또한 전체 예상 일정을 관리하여 대략적인 종료 시간 등을 예측하는 기능도 부가적으로 고려합니다.

UI 요소: - **브레이크 타이머 표시:** 휴식 시간이 시작되면 블라인드 타이머 대신 "Break Time: 남은 휴식 10:00"과 같이 별도의 타이머를 표시합니다. 화면 색상을 다르게 하거나 "BREAK"라는 큰 문구를 띄워 현재 휴식 중임을 명확히 합니다. - **휴식 예정 안내:** 일반 진행 중에도 다음 휴식까지 남은 시간을 표시하거나, 다음 휴식이 어느 레벨 후인지 안내하는 텍스트를 UI 어딘가에 표기합니다. 예: "다음 휴식까지 2레벨 남음" 또는 "다음 브레이크: Level 7 종료 후". - **일정/타임라인 표시:** 참가자용 정보 페이지나 운영자 대시보드에 오늘 대회 진행 예정표를 간략히 보여줄 수 있습니다. 예: "19:00 대회 시작, 21:00 휴식, 23:00 종료 예정" 등의 타임라인.

DB 구조: 휴식 시간 정보는 블라인드 레벨 구조에 포함시키는 방법이 가장 논리적입니다. 예를 들어 `blindLevels` 배열에 `{ level: 6, break: true, duration: 10 }` 같은 항목으로 휴식 레벨을 표시합니다. 또는 `breaks: [ { afterLevel: 5, duration: 10 } ]`처럼 별도 구조로 관리할 수도 있습니다. 일단은 블라인드 구조에 통합해두면 타이머 로직을 재사용하기 편합니다. 또한 예상 일정은 대회 시작 시각과 각 레벨 durations 합을 계산하여 추정할 수 있지만, 수동으로 조정하기 위해 `expectedEndTime` 같은 필드를 둘 수도 있습니다.

실현 계획: 1. **휴식 데이터 설정:** 대회 생성 시 운영자가 휴식 시간을 설정할 수 있게 합니다. 예를 들어 "Level 6 종료 후 10분 휴식"과 같은 설정을 UI에서 추가하면, blindStructure에 해당 정보를 추가합니다. 나중에 구조표를 파싱할 때 break 여부를 인식할 수 있게 합니다. 2. **타이머 연동:** 블라인드 타이머 구현 시 레벨이 휴식 레벨인지 체크하여, 휴식일 경우 시간을 **카운트다운 표시**하지만 블라인드 수치는 "-"로 표시하거나 "BREAK" 표시를 합니다. 휴식 타이머가 0이 되면 자동으로 다음 레벨 (휴식 다음 레벨)로 넘어가도록 타이머를 재개합니다. 3. **공지 및 알림:** 휴식 시작 1분 전쯤 참가자 페이지에 "곧 휴식 시작" 배너를 띄우거나, 소리 알림을 줄 수 있습니다. 휴식 시작/종료 시에도 공지사항을 자동 등록하거나, 타이머 화면 색상을 변경하는 등 시각적 효과를 줘서 모두 인지할 수 있게 합니다. 4. **일정 관리:** 예상 종료 시간이나 진행 속도를 추정하는 기능은 고급 기능이므로 추후 고려합니다. 현재 레벨과 남은 레벨, 남은 휴식 등을 계산해 "예상 종료: 23:10" 같은 정보를 운영자에게 제공하면 편리합니다. 이를 위해 각 레벨 소요시간 합산 + 휴식 합산을 더해 계산하고, 참가자 탈락 속도에 따라 동적으로 조정하는 알고리즘도 구상할 수 있습니다.

以上의 기능들을 종합적으로 구현하면, **포커 토너먼트 운영 플랫폼**으로서 기본적인 요구사항을 모두 갖추게 됩니다. 초기 버전에서는 핵심 기능인 참가자 관리, 좌석 배정, 블라인드 타이머 등에 우선 집중하고, 단계적으로 딜러 관리나 기록 저장, 부가 기능들을 확장해 나갑니다. 개발 중간중간 **Firestore** 보안 및 **실시간 동기화**를 꼼꼼히 점검하여 데이터 일관성과 참여자들의 원활한 사용 경험을 보장하는 것이 중요합니다. 이러한 계획을 바탕으로 효율적으로 개발을 진행하고, 필요시 추가 기능도 유연하게 반영하여 완성도 높은 시스템을 만들어 나갑니다.
