Kickoff (Playbook) Documentation: Public API

Author	Corey Humeston
Description	Outlines current public API for usage.
Disciplines	
Last Edited At	@October 17, 2024 10:24 PM
Last Edited By	Corey Humeston
Stakeholders	Corey Humeston
Tags	
Type	

1. `searchKickoffs`

- Description: Gives us the ability to search kickoffs byIDs to see vital information within the kickoff.
    - `searchKickoffs(input: SearchKickoffsInput!): SearchKickoffsResponse!`
- Input:

```
input SearchKickoffsInput {
    after: String
    first: Int
    filters: KickoffFilters
    sortBy: KickoffSortType
}

input KickoffFilters {
    byIDs: [String]
}

enum KickoffSortType {
    DEFAULT
    CREATED_AT_ASC
    CREATED_AT_DESC
    UPDATED_AT_ASC
    UPDATED_AT_DESC
}
```

- Response:

```
type SearchKickoffsResponse {
    edges: [KickoffEdge]
    pageInfo: PageInfo
    totalCount: Int
}

type PageInfo {
  endCursor: String
  hasNextPage: Boolean
}

type KickoffEdge {
    node: Kickoff
    cursor: String!
}

type Kickoff {
    id: ID!
    name: String
    slateID: String
    difficulty: Int # 0 - Free, 1 - Common+, 2 - Rare+, 3 - Lege
    slots: [KickoffSlot]
    submissionDeadline: Time
    status: KickoffStatus
    winnerDapperID: String # Not usable currently. Ignore this f
    gamesStartAt: Time
    completedAt: Time
    createdAt: Time
    updatedAt: Time

    # --- associated entity
    numParticipants: Int
}

type KickoffSlot {
    id: ID!
    createdAt: Time
    updatedAt: Time
    stats: [KickoffStat]
    requirements: [KickoffSlotRequirements]
    slotOrder: Int
}

type KickoffStat {
    id: String
    stat: KickoffStatistic
    valueNeeded: Int
    valueType: KickoffValueType
    groupV2: String
}
```

```
enum KickoffStatistic {
    TOUCHDOWNS
    TACKLES
    SACKS
    PASSES_ATTEMPTED
    PASSES_SUCCEEDED_YARDS
    RECEPTIONS
    RECEPTIONS_YARDS
    RUSHES
    RUSHING_YARDS
    TACKLES_SOLO
    TACKLES_ASSISTED
    TACKLES_FOR_LOSS
    EXTRA_POINTS_SUCCEEDED
    FIELD_GOALS_SUCCEEDED
    FIELD_GOALS_SUCCEEDED_YARDS_LONGEST
    FUMBLES_FORCED
    INTERCEPTIONS
    PASSES_RATING
    PASSES_SUCCEEDED
    PASSES_SUCCEEDED_YARDS_LONGEST
    PASSES_SUCCEEDED_PERCENTAGE
    PASSES_SUCCEEDED_THIRTY_PLUS_YARDS
    RECEPTIONS_YARDS_AVERAGE
    RECEPTIONS_YARDS_LONGEST
    RECEPTIONS_THIRTY_PLUS_YARDS
    PASSES_TARGETED_AT
    YARDS_AFTER_CATCH
    RUSHING_YARDS_AVERAGE
    RUSHING_YARDS_LONGEST
    RUSHES_TEN_PLUS_YARDS
    RUSHES_TWENTY_PLUS_YARDS
    TOUCHDOWNS_PASSES
    TOUCHDOWNS_PASSES_YARDS_LONGEST
    TOUCHDOWNS_RECEPTIONS
    TOUCHDOWNS_RECEPTIONS_YARDS_LONGEST
    TOUCHDOWNS_RUSHING
    TOUCHDOWNS_RUSHING_YARDS_LONGEST
    RUSHES_FIFTEEN_PLUS_MILES_PER_HOUR
    RUSHES_TWENTY_PLUS_MILES_PER_HOUR
    PASSES_DEFENDED
}

enum KickoffValueType {
    INDIVIDUAL
    CUMULATIVE
}

type KickoffSlotRequirements {
    editionFlowIDs: [Int]
    playerIDs: [String]
    playTypes: [PlayType]
    setIDs: [ID]
    teamIDs: [ID]
    tiers: [EditionTier]
    series: [ID]
    seriesFlowIDs: [Int]
    playerPositions: [PlayerPosition]
    badgeSlugs: [BadgeSlug]
    combinedBadgeSlugs: [BadgeSlug]
}

enum KickoffStatus {
    SCHEDULED
    OPEN
    STARTED
    FINISHED
    PROCESSED
}
```

- Usage: Searching a kickoff will give you all the vital information to view the progress, details, and requirements needed to enter slots for this “game”. We can think of each Kickoff being a “game” that has X requirements and X amount of slots.
- Example query:

```
QUERY ----
query SearchKickoffs($input: SearchKickoffsInput!) {
  searchKickoffs(input: $input) {
    edges {
        node {
            id
            name
            slateID
            difficulty
            slots {
                id
                stats {
                    id
                    stat
                    valueNeeded
                    valueType
                    groupV2
                }
                requirements {
                    playerPositions
                }
            }
            submissionDeadline
            status
            gamesStartAt
            completedAt
        }
        cursor
        }
        totalCount
    }
}

GRAPHQL VARIABLES ----
{
    "input": {
        "after": "",
        "first": 0,
        "filters": {
            "byIDs": ["08558f32-3b10-4325-8b54-37d450a195a1"]
        }
    }
}
```

2. `getKickoffEligiblePlayers`

- Description: This endpoint is the main point of contact for getting what players can be used for each slot within a kickoff. Also tracks injuries, stats, and eligibility per slot.
    - For authenticated users — needs header token via `x-id-token` (available within Headers > X-Id-Token)
    - For unauthenticated users — no need for token, but can only view "Free" kickoffs (difficulty 0).
    - `getKickoffEligiblePlayers(input: GetKickoffEligiblePlayersInput!): GetKickoffEligiblePlayersResponse!`
- Input:

```
input GetKickoffEligiblePlayersInput {
    kickoffID: String!
}
```

- Response:

```
type GetKickoffEligiblePlayersResponse {
    eligiblePlayers: [KickoffEligiblePlayer]
}

type KickoffEligiblePlayer {
    playerGame: PlayerGame
    injury: Injury
    gameStartsAt: Time
    eligibleSlots: [EligibleSlots]
}

type PlayerGame {
    playerID: String
    gameID: String
    teamID: ID
    fullName: String
    firstName: String
    lastName: String
    position: PlayerPosition
    statAveragesBySeason: [KickoffStatDisplayBySeason]
}

type Injury {
    id: String
    playerID: String
    statusV2: KickoffInjuryStatusV2
    startDate: Time
}

type EligibleSlots {
    slotID: String!
    eligibility: KickoffPlayerEligibility!
    eligibleMoments: [MomentNFT]
```

```
}

type KickoffStatDisplayBySeason {
    stat: KickoffStatistic!
    value: Float
    season: Int
}

type MomentNFT {
  id: ID!
  ownerAddress: String!
  serialNumber: Int!
  flowID: UInt64!
  distributionFlowID: Int!
  editionFlowID: Int!
  packNFTFlowID: UInt64!

  # --- associated entity
  edition: Edition
  owner: UserProfile
  badges: [Badge]
  momentNFTListing: MomentNFTListing
  lockExpiresAt: Time

  # --- challenge related
  nftsUsageData: [NftUsageData!]
}

type Edition {
    id: ID!
    flowID: Int!
    series: Series!
    seriesFlowID: Int!
    set: Set!
    setFlowID: Int!
    play: Play!
    playFlowID: Int!
    maxMintSize: Int
    currentMintSize: Int
    tier: EditionTier
    description: String
    assetType: EditionAssetType
    assetVersion: String
    momentNFTListings(input: SearchMomentNFTListingsInput): Sear

    # MomentNFTs within the Edition owned by the authenticated u
    authenticatedUserOwnedMomentNFTs(input: SearchMomentNFTsInpu

    # Moment State
    numMomentsOwned: Int
    numMomentsInPacks: Int
    numMomentsUnavailable: Int
    numMomentsBurned: Int
    numMomentsLocked: Int

    badges: [Badge]
    isLocked: Boolean!

    evolutionStatus: EditionEvolutionStatus
    evolution: EditionEvolution
    currentStep: Int
    targetStep: Int
    completedEvolutions: Int
    justEvolved: Boolean!
    isOpenEdition: Boolean!
}

type Series {
  id: ID!
  flowID: Int!
  name: String!
  active: Boolean!
  closed: Boolean!
  offChainMetadata: OffChainSeriesMetadata
}

type OffChainSeriesMetadata {
  description: String
}

type Set {
  id: ID!
  name: String!
  description: String!
  flowID: Int!
  totalCompletedByUsers: Int!
  totalEditions: Int!
  flowSeriesNumber: Int!
  isOpen: Boolean!
  isHidden: Boolean!
  userData: SetUserData
  assetURL: String
  editions: [Edition]
  offChainMetadata: OffChainSetMetadata
  createdAt: Time!
  updatedAt: Time
  openedAt: Time
  closedAt: Time
  setSeriesData: [SetSeriesData]
  assetURLs: [SetSeriesTierAssets]
}

type OffChainSetMetadata {
  description: String
}

type SetSeriesData {
  seriesFlowID: Int
  totalEditionsInSetSeries: Int
}

type SetUserData {
  totalOwned: Int
  startedAt: Time
  completedAt: Time
  seriesUserData: [SetSeriesUserData]
}

type SetSeriesUserData {
  totalOwned: Int
  seriesFlowID: Int
  startedAt: Time
  completedAt: Time
}

type SetSeriesTierAssetsUrls {
    mobileURL:  String
    desktopURL: String
    videoURL:   String
}

type SetSeriesTierAssets {
  seriesFlowID: Int
  tier: String
  assetsURLs: SetSeriesTierAssetsUrls
}

type Play {
    id: ID!
    flowID: Int
    metadata: PlayMetadata
    badges: [Badge]
}
```

```
type PlayMetadata {
    # Off-chain metadata
    state: PlayState
    league: String
    playType: String @deprecated (reason: "Prefer playTypeV2")
    playTypeV2: PlayType
    videos: [PlayVideo]
    audio: [PlayAudio]
    images: [PlayImage]

    # On-chain metadata
    classification: PlayClassification
    week: String
    season: String
    description: String

    # Player Data
    playerID: ID
    playerFullName: String
    playerFirstName: String
    playerLastName: String
    playerPosition: String @deprecated (reason: "Prefer playerPo
    playerPositionV2: PlayerPosition
    playerNumber: String
    playerWeight: String
    playerHeight: String
    playerBirthdate: String
    playerBirthplace: String
    playerRookieYear: String
    playerDraftTeam: String
    playerDraftYear: String
    playerDraftRound: String
    playerDraftNumber: String
    playerCollege: String

    # Game Data
```

}

```
    teamID: ID
    gameNflID: String
    gameDate: String
    homeTeamName: String
    homeTeamID: ID
    homeTeamScore: String
    awayTeamName: String
    awayTeamID: ID
    awayTeamScore: String
    gameTime: String
    gameQuarter: String
    gameDown: String
    gameDistance: String
    teamName: String
    fieldPosition: String
}

type PlayVideo {
    type: PlayVideoType!
    url: URL!
    # videoLength is the length of the video in milliseconds
    videoLength: Int
    hasAudio: Boolean!
}

type PlayAudio {
    url: URL!
    narrator: PlayAudioNarrator
}

type PlayAudioNarrator {
    profilePicture: String
    name: String
    position: String
    organization: String
}
```

```
type PlayImage {
    type: PlayImageType!
    url: String!
}

input SearchMomentNFTListingsInput {
  after: String
  first: Int
  filters: MomentNFTListingFilters
  sortBy: MomentNFTListingsSortType
}

input MomentNFTListingFilters {
  byIDs:[String]
  byNFTFlowIDs: [Int]
  byEditionFlowIDs: [Int]
  byListingFlowIDs: [UInt64]
  minPrice: PriceInput
  maxPrice: PriceInput
}

input PriceInput {
    value: PriceScalar
    currency: Currency
}

type SearchMomentNFTListingsResponse {
  edges: [MomentNFTListingEdge]
  pageInfo: PageInfo
  totalCount: Int
}

type MomentNFTListingEdge {
  node: MomentNFTListing
  cursor: String!
```

```
}

type MomentNFTListing {
  id: ID!
  nftFlowID: UInt64!
  momentNFT: MomentNFT
  priceV2: Price
  createdAt: Time!
  updatedAt: Time!
  listingFlowID: UInt64!
}

type Price {
    value: PriceScalar
    currency: Currency
}

type PageInfo {
  endCursor: String
  hasNextPage: Boolean
}

input SearchMomentNFTsInputV2 {
  after: String
  first: Int
  filters: MomentNFTFilters
  sortBy: MomentNFTSortType
}

input MomentNFTFilters {
  byOwnerDapperIDs: [String]
  byIDs: [String]
  byFlowIDsV2: [UInt64]
  bySeriesFlowIDs: [Int]
  byOwnerFlowAddresses: [String] # After implemented this one, w
  byPlayTypes: [PlayType]
  byPlayClassification: [PlayClassification]
  bySetFlowIDs: [Int]
  bySetIDs: [String]
  byEditionFlowIDs: [Int]
  byPlayerIDs: [String]
  bySeries: [ID]
  byTeamIDs: [String]
  byTiers: [EditionTier]
  byPlayerPositions: [PlayerPosition]
  byBadgeSlugs: [BadgeSlug]
  byCombinedBadgeSlugs: [BadgeSlug]
  byIsOpenEdition: Boolean
  byIsLocked: Boolean
  bySerialNumbers: [Int]
  byPlayFlowIDs: [Int]
  byPlayerFullNames: [String]
  byTeamNames: [String]
  bySeasons: [String]
  byLeagues: [String]
  bySetNames: [String]
  byLeaderboardID: String
}

type SearchMomentNFTsResponseV2 {
  edges: [MomentNFTEdge]
  pageInfo: PageInfo
  totalCount: Int
}

type MomentNFTEdge {
  node: MomentNFT
  cursor: String!
}

enum MomentNFTSortType {
  DEFAULT
  CREATED_AT_ASC
  CREATED_AT_DESC
  UPDATED_AT_ASC
  UPDATED_AT_DESC
  ACQUIRED_AT_ASC
  ACQUIRED_AT_DESC
  SERIAL_NUMBER_ASC
  SERIAL_NUMBER_DESC
  LISTED_PRICE_ASC
  LISTED_PRICE_ASC_NULLS_FIRST
  LISTED_PRICE_ASC_NULLS_LAST
  LISTED_PRICE_DESC
  LISTED_PRICE_DESC_NULLS_FIRST
  LISTED_PRICE_DESC_NULLS_LAST
}

enum BadgeSlug {
    ALL_DAY_DEBUT,
    CHAMPIONSHIP_YEAR,
    FIRST_SERIAL,
    LOW_SERIAL,
    PLAYER_NUMBER,
    ROOKIE_MINT,
    ROOKIE_YEAR,
    CRAFTED_REWARD,
    CHALLENGE_REWARD,
    DYNAMIC_MOMENT,
    PERFECT_SERIAL,
    HALL_OF_FAME,
}

enum MomentNFTListingsSortType {
  CREATED_AT_ASC
  CREATED_AT_DESC
  UPDATED_AT_ASC
  UPDATED_AT_DESC
  PRICE_ASC
  PRICE_DESC
  SERIAL_NUMBER_ASC
  SERIAL_NUMBER_DESC
}

enum PlayImageType {
    PLAY_IMAGE_TYPE_NIL
    PLAY_IMAGE_TYPE_CROPPED_ASSET
}

enum PlayClassification {
    PLAYER_GAME
    TEAM_GAME
    PLAYER_MELT
    TEAM_MELT
}

enum PlayState {
    PLAY_STATUS_DRAFT
    PLAY_STATUS_PUBLISHED
    PLAY_STATUS_QA
    PLAY_STATUS_CURATED
}

enum PlayType {
    BLOCK,
    BLOCKED_KICK,
    FIELD_GOAL,
    FORCED_FUMBLE,
    FUMBLE_RECOVERY,
    INTERCEPTION,
    KICK_RETURN,
    PASS,
    PASS_DEFENSE,
    PLAYER_MELT,
    PRESSURE,
    PUNT,
    PUNT_RETURN,
    RECEPTION,
    RUSH,
    SACK,
    SAFETY,
    STRIP_SACK,
    TACKLE,
    TWO_POINT_ATTEMPT,
    TEAM_MELT,
}

enum PlayVideoType {
    PLAY_VIDEO_TYPE_NIL
    PLAY_VIDEO_TYPE_VERTICAL
    PLAY_VIDEO_TYPE_SQUARE
}

enum EditionTier {
    COMMON,
    UNCOMMON,
    LEGENDARY,
    RARE,
    ULTIMATE,
}

enum EditionAssetType {
    STATIC
    DYNAMIC
}

enum KickoffStatistic {
    TOUCHDOWNS
    TACKLES
    SACKS
    PASSES_ATTEMPTED
    PASSES_SUCCEEDED_YARDS
    RECEPTIONS
    RECEPTIONS_YARDS
    RUSHES
    RUSHING_YARDS
    TACKLES_SOLO
    TACKLES_ASSISTED
    TACKLES_FOR_LOSS
    EXTRA_POINTS_SUCCEEDED
    FIELD_GOALS_SUCCEEDED
    FIELD_GOALS_SUCCEEDED_YARDS_LONGEST
    FUMBLES_FORCED
    INTERCEPTIONS
    PASSES_RATING
    PASSES_SUCCEEDED
    PASSES_SUCCEEDED_YARDS_LONGEST
    PASSES_SUCCEEDED_PERCENTAGE
    PASSES_SUCCEEDED_THIRTY_PLUS_YARDS
    RECEPTIONS_YARDS_AVERAGE
    RECEPTIONS_YARDS_LONGEST
    RECEPTIONS_THIRTY_PLUS_YARDS
    PASSES_TARGETED_AT
    YARDS_AFTER_CATCH
    RUSHING_YARDS_AVERAGE
    RUSHING_YARDS_LONGEST
    RUSHES_TEN_PLUS_YARDS
    RUSHES_TWENTY_PLUS_YARDS
    TOUCHDOWNS_PASSES
    TOUCHDOWNS_PASSES_YARDS_LONGEST
    TOUCHDOWNS_RECEPTIONS
    TOUCHDOWNS_RECEPTIONS_YARDS_LONGEST
    TOUCHDOWNS_RUSHING
    TOUCHDOWNS_RUSHING_YARDS_LONGEST
    RUSHES_FIFTEEN_PLUS_MILES_PER_HOUR
    RUSHES_TWENTY_PLUS_MILES_PER_HOUR
    PASSES_DEFENDED
}

enum PlayerPosition {
    DB,
    DL,
    K,
    LB,
    OL,
    P,
    QB,
    RB,
    TE,
    WR,
    MELT,
}

enum KickoffInjuryStatusV2 {
    OUT
    Q
    IR
    D
    P
    SUS
    RETURNED
    UNKNOWN
}

enum KickoffPlayerEligibility {
    ELIGIBLE
    ELIGIBLE_SUBMITTED
    INELIGIBLE_MOMENT_OWNERSHIP
    INELIGIBLE_GAME_STARTED
}
```

}

```
type Badge {
    id: ID
    slugV2: BadgeSlug
    title: String
    description: String
    visible: Boolean
}

type EditionEvolution {
    currentStep: Int
    targetStep: Int
    completedEvolutions: Int
    justEvolved: Boolean!
    history: [EditionEvolutionHistory]
}

type EditionEvolutionHistory {
    id: ID!
    editionFlowID: Int!
    eventTime:     Time!
    eventName:     String!
    eventDetail:   String! @deprecated(reason: "Use homeTeamID a
    homeTeamID:    ID
    awayTeamID:    ID
    gameResult:    String! @deprecated(reason: "Use homeTeamScor
    homeTeamScore: String
    awayTeamScore: String
    hasPlayed:     Boolean! @deprecated
    gamePoints:    Int
    statPoints:    Int
    currentStep:   Int
    targetStep:    Int
    videoStartTimeMs: Int
}
```

```
type UserProfile {
  id: String
  dapperID: String
  email: String
  phoneNumber: String
  username: String
  flowAddress: String
  profileImageUrl: String
  isCurrentTOSSigned: Boolean
  isMarketingAllowed: Boolean
  hasCompletedFTUE: Boolean
  signedTermsOfServices: [UserSignedTermsOfService]
  isVerified: Boolean
  verifiedUserProfile: VerifiedUserProfile
  createdAt: Time
  favoriteTeamIDs: [String]
  favoriteTeams: [Team]
  accolades: [AccoladeAndUserAccolade]
  banners: [BannerAndUserBanner]
  selectedBannerID: String  # you can provide "favorite_team_ba
  selectedBanner: Banner
  playbookSeen: String
  hasSeenUserProfile: Boolean
  hasClaimedSeasonalNFT: Boolean
  profilePins: [Pin]
  storedPins: [StoredPin]
  hasVisitedMarketPlace: Boolean
}

type UserSignedTermsOfService {
  signedAt: Time
  version: Int!
}

type VerifiedUserProfile {
  name: String
  organization: String
  jerseyNumber: Int
  createdAt: Time
  updatedAt: Time
}

type Team {
  id: ID!
  name: String!
  nflIDs: [String]!
  createdAt: Time!
  updatedAt: Time
  assetUrls: AssetUrls
  counters: TeamCounters
}

type AssetUrls {
  bannerDesktopUrl: String!
  bannerMobileUrl: String!
  logoDesktopUrl: String!
  logoMobileUrl: String!
  bannerWithLogoDesktopUrl: String!
  bannerWithLogoMobileUrl: String!
}

type TeamCounters {
  totalUniqueMoments: Int
  totalUniqueOwners: Int
}

type AccoladeAndUserAccolade {
  # --- accolade data
  id: ID!
  rewardID: ID!
  eventName: String!
  name: String!
  tier: AccoladeTier!
  status: AccoladeStatus!
  description: String
  assetUrls: AccoladeAssetUrls
  createdAt: Time!
  updatedAt: Time
  enabledAt: Time
  stoppedAt: Time
  season: String
  week: String
   # --- accolade_percentages data
  totalClaimed: Int
  percentageOfUsers: String
   # --- user_accolade data
  acquiredAt: Time
  claimedAt: Time
}

type AccoladeAssetUrls {
  iconDesktopUrl: String
  iconMobileUrl: String
  videoURL: String
}

type BannerAndUserBanner {
  # --- banner data
  id: ID!
  rewardID: ID!
  eventName: String!
  name: String!
  tier: BannerTier!
  status: BannerStatus!
  description: String
  assetUrls: BannerAssetUrls
  createdAt: Time!
  updatedAt: Time
```

}

```
  enabledAt: Time
  stoppedAt: Time
   # --- user_banner data
  acquiredAt: Time
  claimedAt: Time
  sourceID: String
  sourceType: BannerSourceType
}

type BannerAssetUrls {
  iconDesktopUrl: String
  iconMobileUrl: String
}

type Pin {
    pinType: PinType!
    sourceId: String!
    assetUrl: String
    title: String
    subtitle: String
    name: String
    description: String
}

type StoredPin {
    pinType: PinType!
    sourceId: String!
}

type NftUsageData {
    nftID: ID!
    challengeID: ID!
    challengeCategory: ChallengeCategory!
    submissionID: ID!
    submissionStatus: ChallengeSubmissionStatus!
}
```

```
enum AccoladeTier {
  GRAPHITE,
  SILVER,
  GOLD,
  PLATINUM,
}

enum AccoladeStatus {
  ACTIVE,
  INACTIVE,
}

enum BannerTier {
  BRONZE,
  SILVER,
  GOLD,
  PLATINUM,
}

enum BannerStatus {
  ACTIVE,
  INACTIVE,
}

enum BannerSourceType {
  DEFAULT
  FAVORITE_TEAM
  REWARDS
  TEAM_NFT
}

enum EditionEvolutionStatus {
    NOT_STARTED
    IN_PROGRESS
    EVOLVING
    COMPLETED
}

enum PinType {
    ACCOLADE
    TEAM_MOMENTS_COUNT
    TEAM_LEADERBOARD_POSITION
    SET_PROGRESS
    SETS_COMPLETED
}

enum ChallengeCategory {
    PRESCRIPTIVE
    PRESCRIPTIVE_FREE
    PREDICTIVE
    PREDICTIVE_FREE
    BURN
    PRESCRIPTIVE_INSTANT
    LOCK
}

enum ChallengeSubmissionStatus {
    VALID
    MISSING_REQUIREMENTS
    BURNING_PENDING
    PREDICTIVE_PENDING
    PREDICTIVE_REJECTED
    LOCKING_PENDING
    COMPLETE
}
```

- Usage: Getting an eligible player will return you the injury of the player, the stats for the season for the player, and the eligibility of the slots the player can be predicted in. For authenticated users, it takes into account ownership of a `momentNFT` that corresponds to the player.

- Example query:

```
QUERY ----
query {
    getKickoffEligiblePlayers(input: {
        kickoffID: "1f276bf7-a572-4b6f-bc97-d50181009c3b"
    }) {
        eligiblePlayers {
            injury {
                id
                status
                statusV2
                startDate
                comment
            }
            playerGame {
                playerID
                gameID
                teamID
                fullName
                position
            }
            gameStartsAt
            eligibleSlots {
                slotID
                eligibility
                eligibleMoments {
                    id
                    edition {
                        id
                    }
                }
            }
        }
    }
}
```

3. `searchKickoffSubmissions`

- Description: Searching by kickoff submissions will give us every submission submitted so far by every user. This will include win status, stats for player within game, and points for the slot (either 0 or 1). This can be used to track progress during the kickoff to see live updates depending on how quickly you are querying the API.
    - `searchKickoffSubmissions(input: SearchKickoffSubmissionsInput!): SearchKickoffSubmissionsResponse!`
- Input:

```
input SearchKickoffSubmissionsInput {
    after: String
    first: Int
    filters: KickoffSubmissionFilters
    sortBy: KickoffSubmissionSortType
}

input KickoffSubmissionFilters {
    byKickoffSlateID: String
    byKickoffSlateIDs: [String]
    byKickoffID: String
    byKickoffIDs: [String]
    byDapperIDs: [String]
}

enum KickoffSubmissionSortType {
    CREATED_AT_ASC
    CREATED_AT_DESC
    UPDATED_AT_ASC
    UPDATED_AT_DESC
}
```

- Response:

```
type SearchKickoffSubmissionsResponse {
    edges: [KickoffSubmissionEdge]
    pageInfo: PageInfo
    totalCount: Int
}

type KickoffSubmissionEdge {
    node: KickoffSubmission
    cursor: String!
}

type KickoffSubmission {
    id: ID!
    dapperID: String
    user: UserProfile
    kickoffID: String
    slots: [KickoffSubmissionSlot]
    createdAt: Time
    updatedAt: Time
}

type KickoffSubmissionSlot {
    id: ID!
    kickoffSlotID: ID
    submissionID: ID
    playerID: String
    points: Int
    momentFlowID: UInt64
    momentTier: String
    serialNumber: Int
    fullName: String
    teamID: String
    requirements: [KickoffSlotRequirements]
    playerInSlot: PlayerGame
    gameStartAt: Time
    createdAt: Time
    updatedAt: Time
    winStatus: KickoffWinStatus
    stats: [KickoffSubmissionStat]
    injury: Injury
}

enum KickoffWinStatus {
    WIN
    LOSS
    PENDING
    UNSCORED
}

type KickoffSubmissionStat {
    tally: Float
    stat: KickoffStat
}
```

- Usage: Searching a kickoff submission will give you live updates for every submission and every slot within that submission based on your filtering criteria.
- Example query:

```
QUERY ----
query SearchKickoffSubmissions($input: SearchKickoffSubmissionsI
  searchKickoffSubmissions(input: $input) {
    edges {
        node {
            id
            dapperID
            user {
                id
            }
            kickoffID
            slots {
                id
                kickoffSlotID
                winStatus
                submissionID
                playerID
                points
                momentFlowID
                momentTier
                serialNumber
                fullName
                teamID
                requirements {
                    setIDs
                }
                playerInSlot {
                    playerID
                    gameID
                    teamID
                    fullName
                    firstName
                    lastName
                    position
                }
                gameStartAt
                createdAt
                updatedAt
                winStatus
                stats {
                    tally
                    stat {
                        id
                        stat
                        valueNeeded
                        valueType
                    }
                }
                injury {
                    id
                    playerID
                    status
                    startDate
                }
            }
            createdAt
            updatedAt
        }
        cursor
        }
        totalCount
    }
}


GRAPHQL VARIABLES ----
{
  "input": {
    "filters": {
      "byKickoffSlateID":  "926f9469-5f10-4f64-a961-7a2ba1e1333
    }
  }
}
```

4. `searchKickoffGames`

- Description: Gives us the ability to view game progress, including clock, score, and quarter.
    - `searchKickoffGames(input: SearchKickoffGamesInput!): SearchKickoffGamesResponse!`
- Input:

```
input SearchKickoffGamesInput {
    after: String
    first: Int
    filters: KickoffGameFilters
    sortBy: KickoffGameSortType
}

input KickoffGameFilters {
    byKickoffSlateID: String
    byGameDateRange: TimeRange
    byKickoffID: String
}

enum KickoffGameSortType {
    SCHEDULED_AT_ASC
    SCHEDULED_AT_DESC
    CREATED_AT_ASC
    CREATED_AT_DESC
    UPDATED_AT_ASC
    UPDATED_AT_DESC
}
```

- Response:

```
type SearchKickoffGamesResponse {
    edges: [KickoffGameEdge]
    pageInfo: PageInfo
    totalCount: Int
}
```

```
type KickoffGameEdge {
    node: KickoffGame
    cursor: String!
}

type KickoffGame {
    fixtureID: String
    status: String
    homeTeamID: String
    awayTeamID: String
    homeTeamScore: Int
    awayTeamScore: Int
    clock: String
    quarter: Int
    scheduledAt: Time
}
```

- Usage: Main usage is to keep track of current game progress within the kickoff.
- Example query:

```
QUERY ----
query SearchKickoffGames($input: SearchKickoffGamesInput!) {
  searchKickoffGames(input: $input) {
    edges {
        cursor
        node {
            fixtureID
            status
            homeTeamID
            awayTeamID
            homeTeamScore
            awayTeamScore
            clock
            quarter
            scheduledAt
        }
    }
    pageInfo {
        endCursor
        hasNextPage
    }
    totalCount
}
}


GRAPHQL VARIABLES ----
{
    "input": {
        "after": "",
        "first": 0,
        "filters": {
            "byKickoffID": "08558f32-3b10-4325-8b54-37d450a195a
        }
    }

}
```

5. `searchKickoffSlates`

- Description: Gives us the ability to view all the kickoffs within the slate. Each slate has multiple kickoffs by different difficulties.
    - `searchKickoffSlates(input: SearchKickoffSlatesInput!): SearchKickoffSlatesResponse!`
- Input:

```
input SearchKickoffSlatesInput {
    after: String
    first: Int
```

}

```
    filters: KickoffSlateFilters
    sortBy: KickoffSlateSortType
}

input KickoffSlateFilters {
    byIDs: [String]
    byStatuses: [KickoffSlateStatus]
}

enum KickoffSlateStatus {
    NOT_STARTED
    RUNNING
    FINISHED
    PROCESSED
    UNKNOWN
}

enum KickoffSlateSortType {
    START_DATE_ASC
    START_DATE_DESC
    END_DATE_ASC
    END_DATE_DESC
    CREATED_AT_ASC
    CREATED_AT_DESC
    UPDATED_AT_ASC
    UPDATED_AT_DESC
}
```

- Response:

```
type SearchKickoffSlatesResponse {
    edges: [KickoffSlateEdge]
    pageInfo: PageInfo
    totalCount: Int
}
```

```
type KickoffSlateEdge {
    node: KickoffSlate
    cursor: String!
}

type KickoffSlate {
    id: String!
    name: String
    startDate: Time
    endDate: Time
    status: KickoffSlateStatus
    kickoffs: [Kickoff]
}
```

- Usage: Searching by kickoff slates will give you all the information for individual kickoffs within that slate. Each slate usually contains 4 kickoffs, rated from difficulty 0 - 4. We can think of a “slate” as being the container that holds all of the kickoffs. Once all the kickoffs are completed, the slate is completed.
- Example query:

```
QUERY ----
query SearchKickoffSlates($input: SearchKickoffSlatesInput!) {
  searchKickoffSlates(input: $input) {
    edges {
        node {
            id
            name
            startDate
            endDate
            status
            kickoffs {
                id
                slateID
            }
        }
        cursor
        }
        totalCount
    }
}


GRAPHQL VARIABLES ----
{
    "input": {
        "after": "",
        "first": 0,
        "filters": {
            "byIDs": ["926f9469-5f10-4f64-a961-7a2ba1e13333"]
        }
    }
}
```

6. `getPlayersWeeklyStats`

- Description: Gives us the players weekly stats to showcase.
  - `getPlayersWeeklyStats(input: GetPlayersWeeklyStatsInput!): GetPlayersWeeklyStatsResponse!`
- Input:

```
input GetPlayersWeeklyStatsInput {
    playerID: String!
    statCategories: [KickoffStatistic!]!
    numberOfRounds: Int
    orderBy: KickoffStatOrderBy
}
```

```
enum KickoffStatistic {
    TOUCHDOWNS
    TACKLES
    SACKS
    PASSES_ATTEMPTED
    PASSES_SUCCEEDED_YARDS
    RECEPTIONS
    RECEPTIONS_YARDS
    RUSHES
    RUSHING_YARDS
    TACKLES_SOLO
    TACKLES_ASSISTED
    TACKLES_FOR_LOSS
    EXTRA_POINTS_SUCCEEDED
    FIELD_GOALS_SUCCEEDED
    FIELD_GOALS_SUCCEEDED_YARDS_LONGEST
    FUMBLES_FORCED
    INTERCEPTIONS
    PASSES_RATING
    PASSES_SUCCEEDED
    PASSES_SUCCEEDED_YARDS_LONGEST
    PASSES_SUCCEEDED_PERCENTAGE
    PASSES_SUCCEEDED_THIRTY_PLUS_YARDS
    RECEPTIONS_YARDS_AVERAGE
    RECEPTIONS_YARDS_LONGEST
    RECEPTIONS_THIRTY_PLUS_YARDS
    PASSES_TARGETED_AT
    YARDS_AFTER_CATCH
    RUSHING_YARDS_AVERAGE
    RUSHING_YARDS_LONGEST
    RUSHES_TEN_PLUS_YARDS
    RUSHES_TWENTY_PLUS_YARDS
    TOUCHDOWNS_PASSES
    TOUCHDOWNS_PASSES_YARDS_LONGEST
    TOUCHDOWNS_RECEPTIONS
    TOUCHDOWNS_RECEPTIONS_YARDS_LONGEST
    TOUCHDOWNS_RUSHING
    TOUCHDOWNS_RUSHING_YARDS_LONGEST
    RUSHES_FIFTEEN_PLUS_MILES_PER_HOUR
    RUSHES_TWENTY_PLUS_MILES_PER_HOUR
    PASSES_DEFENDED
}

enum KickoffStatOrderBy {
    ASC
    DESC
}
```

- Response:

```
type GetPlayersWeeklyStatsResponse {
    statsByWeek: [KickoffStatDisplayByWeek]
}

type KickoffStatDisplayByWeek {
    stat: KickoffStatistic!
    value: Float
    round: String
    opponentTeamID: String
    isHomeGame: Boolean
    gameStartAt: Time
    season: Int
}
```

- Usage: We use this currently during the eligible players screen, to sort all players by their weekly stats so users have the opportunity to see who's going to be the best player for a given slot. You can use the slots statistic we're tracking to and plug it into this API to get the average.
- Example query:

```
QUERY ----
query {
    getPlayersWeeklyStats(input: {
        playerID: "00-0030506"
        statCategories: PASSES_TARGETED_AT
        numberOfRounds: 5
        orderBy: DESC
    }) {
        statsByWeek {
            stat
            value
            round
            opponentTeamID
            isHomeGame
            gameStartAt
            season
        }
    }
}
```

7. `getPlayersLowestListingPrice`

- Description: Gives us the most accurate moment nft listings on the marketplace sorted by lowest price for an individual player.
    - `getPlayersLowestListingPrice(input: GetPlayersLowestListingPriceInput!): GetPlayersLowestListingPriceResponse!`
- Input:

```
input GetPlayersLowestListingPriceInput {
    kickoffID: String
}
```

- Response:

```
type GetPlayersLowestListingPriceResponse {
    playerPrice: [PlayerLowestListingPrice]
}

type PlayerLowestListingPrice {
    playerID: String
    price: String
    slotID: String
}
```

- Usage: We currently use this along with the eligible players query to make sure that the user is choosing the right slots for their players. However, if they don’t have the player in their inventory, this query will tell them how much it is to buy X to play within this slot. Can be ignored on kickoffs that have difficulty 0 since free to play doesn’t require spending any money.
- Example query:

```
QUERY ----
query {
    getPlayersLowestListingPrice(input: {
        kickoffID: "9c5a1570-1617-43cb-a257-681c63d31c0a"
    }) {
        playerPrice {
            playerID
            price
            slotID
        }
    }
}
```