ChatGPT

Magic: The Gathering Fundamentals and MTG-App Development Guide

Game Overview and Objective

Magic: The Gathering (MTG) is a turn-based strategy card game where **players duel using decks of cards** representing magical spells, creatures, and artifacts. Each player typically starts with 20 life points (40 in some formats like Commander) and the goal is to reduce opponents' life to 0 or achieve another win condition [1] [2]. Players draw cards from their **library** (deck) into their **hand**, play cards to the **battlefield**, and use strategic timing and resources (mana) to outmaneuver opponents. A player loses if their life hits 0 (or less), if they must draw from an empty library, or if they accumulate 10 poison counters, among other conditions [3]. Some cards or formats introduce alternate win/loss conditions (for example, a special card might say "you win the game" under certain conditions, or in Commander format, 21 damage from a single commander causes a loss [4]). The game can also end in a draw in rare cases (for instance, if all players lose simultaneously) [5] [6].

In MTG, **two or more players** can play. While the **default duel** is 1v1, the game supports multiplayer free-for-all and team variations. The last player or team standing wins once others have lost [2] [7]. Each game is highly dynamic, with thousands of unique cards and interactions, which makes comprehensive rules enforcement a complex but critical aspect of any digital implementation.

Core Rules and Turn Structure

MTG gameplay is structured into **turns**, with each turn broken into well-defined phases and steps. Each player's turn consists of **five phases** in order [8] [9]:

- **Beginning Phase** – divided into the Untap, Upkeep, and Draw steps. Players ready their tapped cards (untap), then upkeep triggers occur, then the active player draws a card [10].
- **Pre-combat Main Phase** – the active player may play most card types (creatures, sorceries, artifacts, etc.) and take actions. This is one of the two main phases where most gameplay actions happen.
- **Combat Phase** – broken into steps: Beginning of Combat, Declare Attackers, Declare Blockers, Combat Damage, and End of Combat [11]. Players attack with creatures and assign combat damage in this phase.
- **Post-combat Main Phase** – a second main phase after combat where the player can again cast spells or play a land, similar to the first main phase.
- **Ending Phase** – divided into End Step and Cleanup. End-step triggers happen, then in Cleanup the active player discards down to maximum hand size and damage wears off creatures [12].

Each phase/step has specific rules. Notably, **players get priority** (the opportunity to act) at most steps. Spells and abilities use a **"stack"** mechanism (last-in, first-out resolution order) and players alternate priority to add responses [9]. A phase or step ends only when the stack is empty and all players pass priority in succession with no further actions [9] [13]. This turn structure and priority system is crucial for implementing the timing of spells and abilities correctly.

Example: In a turn, you untap your cards, upkeep triggers go on the stack, then you draw for turn. In your main phase you may cast a creature spell (which goes on the stack). If the opponent wants to respond (say with a Counterspell), they must do so before you move to the combat phase. After combat, you get another main phase, then end the turn where end-step effects (like "at end of turn" triggers) occur. The rules ensure both players have windows to respond at appropriate times, which the app's game engine must manage carefully.

Card Types and Zones

Every card in Magic has a **card type** that determines how it is played and what rules apply to it [14]. The standard card types are:

- **Creature** – Permanent cards that attack and block to deal damage. Creatures have power/toughness and usually can only be played on your main phase when the stack is empty (unless they have Flash) [15].
- **Land** – Permanent resource cards that produce **mana** (the energy needed to cast spells). You may play one land per turn during a main phase (lands are not cast and do not use the stack) [15].
- **Artifact** – Permanent cards representing objects or equipment. Often they can be played by any deck (colorless) and may have activated abilities.
- **Enchantment** – Permanent cards representing ongoing magical effects (e.g. buffs, rule changes). **Auras** are a subtype of enchantment that attach to other cards.
- **Planeswalker** – Permanent cards representing powerful allies. They have loyalty points and abilities that can be activated once per turn, and they can be attacked by creatures.
- **Sorcery** – Spell cards that produce a one-time effect. They can only be cast on your turn during a main phase when the stack is empty (slow speed) [15].
- **Instant** – Spell cards that produce a one-time effect, like sorceries, but can be cast at any time you have priority (even on opponents' turns). Instants are the primary way to respond on the stack.
- **Tribal** – (Rare) A card type that exists to grant creature-like subtypes to non-creatures (obsolete in recent sets).
- **Battle** – A newer card type introduced in 2023 (e.g. *Siege* subtype) which has its own mechanics; like Planeswalkers, battles can be attacked and have effects when defeated.
- **Conspiracy, Scheme, Plane, Vanguard, etc.** – Special card types used only in certain casual modes (these won't typically appear in normal decks, but the app may consider them if supporting formats like Archenemy or Planechase in the future).

The game is played in several **zones** where cards and objects can exist [16]:

- **Library** – Your deck of cards. Players draw from their library. If a library is empty when a player needs to draw, that player loses [17].
- **Hand** – Cards you've drawn and can play. By rule, a player must discard down to 7 cards in hand during the cleanup step (if over 7).
- **Battlefield** – The play area where permanents (creatures, lands, artifacts, enchantments, planeswalkers, etc.) reside once played. This is where most game interaction happens (creatures fight, etc.).

- **Graveyard** – Discard pile for each player where used spells (instants/sorceries) and destroyed or discarded cards go. Some cards interact with the graveyard (e.g. “flashback” spells can be cast from there, or creatures can be revived from there).
- **Exile** – A zone representing cards set aside from the game. Exiled cards are usually out of play unless an effect brings them back. It’s a "removed from game" area, often used for long-term bans or delayed effects.
- **Stack** – A transient zone where spells and abilities wait to resolve. When a spell is cast or an ability triggers, it goes on the stack; players can respond (adding their own spells/abilities on top). The stack resolves last-in-first-out [16] . The engine must implement this faithfully so that, for example, a counterspell can neutralize a spell on the stack.
- **Command Zone** – A special zone mostly used in Commander format (and some specific card effects). A player’s Commander starts here and can be cast from this zone [18] . Certain other cards (like Emblems or Planechase cards) also reside in command zone. The app must handle the unique rules of the command zone (e.g., increasing cost each time a commander is cast from there, commander not going to graveyard but back to command if it dies, etc. [19]).
- **Hand, Library, Graveyard, Exile, Battlefield, Stack, and Command** cover the primary zones of play [16] . Cards also briefly go to the **discard pile** (which *is* the graveyard in MTG terminology) or **in play** (which is the battlefield).

Understanding card types and zones is fundamental for the rules engine. For instance, **permanents** (lands, creatures, artifacts, enchantments, planeswalkers, battles) reside on the battlefield until removed, while **spells** (instants and sorceries) exist only on the stack and then go to graveyard on resolution [20] . The rules for each card type (e.g. when you can play them, how many you can have, etc.) need to be enforced by the app to match tabletop Magic behavior.

Spells, Abilities, and the Stack

When a player plays a card, it often becomes a **spell on the stack** (except lands and some special actions). The **stack** is the game’s mechanism to handle **simultaneous effects and responses**. It’s last-in, first-out: the last spell or ability added resolves first [21] . Both players can add to the stack in response to others’ actions, enabling interactive gameplay (countering spells, casting reactions, etc.). The app’s engine must manage this ordering and allow players (or AI) to respond during the proper **priority** windows.

Abilities on cards come in three main types:

- **Activated Abilities** – These are abilities a player can activate by paying a cost (often written as “Cost: Effect”). For example, a creature might have “{T}: Deal 1 damage to any target.” Activating it puts that ability on the stack. Costs can include tapping the card, paying mana, discarding, etc. [22] . The engine should allow activation only when the player has priority and the cost can be paid.
- **Triggered Abilities** – These abilities automatically **trigger** and go to the stack when a certain event happens or condition is met. They are written with phrases like “When [condition], [effect]” or “At the beginning of [step], [effect]” [23] . For example, “When this creature dies, draw a card” will trigger and go on stack when that event occurs. The game must detect triggers at the appropriate time (often at start of phases or when some action occurs) and enqueue them to stack in the proper order (usually Active Player’s triggers first, then others – APNAP order).
- **Static Abilities** – These are continuous effects that are simply “always on” as long as the card is in the proper zone. They don’t use the stack. Examples include “creatures you control get +1/+1” on an

enchantment, or keywords like **flying** that continually modify game rules for that card. The engine must continuously apply static abilities and handle interactions (some static abilities create replacement or prevention effects too).

Keyword Abilities (like Flying, Trample, Vigilance, etc.) are a shorthand for common abilities defined in the rules [24]. For instance, "Flying" means the creature can only be blocked by other creatures with flying or "reach." These keywords need to be coded into the engine's rules – either via data or logic – because they affect gameplay (e.g., check flying status during the block assignment step). Many keywords have detailed rules (e.g., "Deathtouch" means any amount of damage from that source is enough to destroy a creature, "Lifelink" causes the controller to gain life equal to damage dealt, etc.). A **comprehensive list of keyword abilities and their rules** can be found in the official rulebook and MTG glossary [25] [26]. The app should include a **Key Terms** or glossary (and indeed we maintain `doc/KEY_TERMS.md` with these definitions for quick reference [16]).

The engine must also implement **state-based actions** – automatic checks that happen whenever a player would get priority, to enforce rules like creatures with 0 toughness die, players with 0 life lose, too many copies of a legendary permanent forces one to go, etc. [27]. State-based actions do not use the stack; they are handled as soon as relevant conditions are true. This is crucial for maintaining rule integrity (for example, if a spell reduces a creature's toughness to 0, the creature should die *before* anyone can do anything else) [27].

All of these mechanics – priority, the stack, activated/triggered abilities, state-based actions – form the backbone of MTG's rules engine. A successful MTG app needs to carefully implement them to ensure the digital gameplay matches the paper game. **Testing complex interactions** (for instance, multiple triggers happening at once, or replacing effects like "if this would die, exile it instead") will be important to validate the engine.

Winning, Losing, and Multiplayer Considerations

A standard duel ends when one player wins and the other loses. Common **win conditions** include the opponent's life total reaching 0 (or less) or the opponent being forced to draw from an empty library (mill win) [17]. **Poison counters** can also kill a player: 10 or more poison counters means that player loses [28]. Some cards explicitly say a player wins or loses the game (the engine should handle those effects as per rules [29]).

Multiplayer games add complexity to win/loss handling. In free-for-all (e.g., four-player Commander), the game continues until only one player remains; when a player loses, they leave the game but others continue. In team games, if all members of a team lose, that team loses collectively [30] [31]. The app needs to handle players leaving the game (their cards usually leave with them, except certain cases in Commander, etc.). The **Comprehensive Rules 104** section details all conditions (the app's logic can reference those conditions, see examples above and also Commander's unique rule below).

Commander (EDH) specific: In addition to the normal rules, Commander has the **"21 Commander damage"** rule – if a player is dealt 21 or more **combat damage** by the *same commander* during the game, that player loses [4]. The app should track commander damage separately for each commander-player pair. Also, each player starts at 40 life in Commander [32], and decks are 100 cards singleton (no duplicates aside from basic lands) [33] . These format-specific rules must be enforced when that format is chosen (e.g., deck validation should flag any Commander deck with duplicates or wrong size).

Two-Headed Giant (2HG): This is a popular team format where two players share a life total (starting at 30 or 40 depending on rules) and take turns simultaneously. The team's life total being 0 or less causes that team to lose. They also share a turn and phases. The app's turn manager must support shared turn structures for such modes (e.g., both heads declare attackers together). In 2HG, teams share life but not other resources (each player still has their own library, hand, etc., but they attack/block as a team) [34] .

Other multiplayer variants like **Emperor (3v3)**, **Archenemy (1 vs many)**, **Star (free-for-all with specific win rules)**, etc., each have additional rules. Emperor, for instance, has range of influence and the Emperor's loss causes team loss [35] [31] . Archenemy uses **Scheme** cards and all other players form a team against the one "boss" player. While implementing every variant is complex, the app is planned to support **"any and every format"** eventually, so the architecture should be flexible to accommodate these rule tweaks.

Priority in Multiplayer: Turn order proceeds clockwise around the table. In free-for-all, priority passes in turn order. The app should correctly handle priority and responses in multiplayer – e.g., after the active player acts, each other player in seat order gets a chance to respond (Active Player, then Non-Active players in turn order – the "APNAP" rule). This ensures the simulation is accurate to tabletop play.

In summary, the app must enforce not only 1v1 rules but also consider how wins/losses and phases work in **multiplayer and team games**. We should design the rules engine to be data-driven or configurable for different formats (e.g., allow setting life totals, team shared resources, custom win conditions for a game mode, etc.). **Testing multiplayer scenarios** (like 2HG turn cycles, or a player losing in a 4-player game) will be important as these can be tricky edge cases.

Popular Formats and Deck Construction Rules

There are many ways to play MTG, known as **formats**, each with its own deck construction rules and card pool [36] [37] . Our application aims to support *all major formats*, so agents should be aware of their basics and how to validate decks for each:

- **Standard** – A *Constructed* tournament format that uses the most recent sets (approximately the last 2 years of card sets, rotating out older sets annually) [38] . Decks are minimum 60 cards, with up to 15-card sideboard. **Standard decks** can only include cards from legal Standard sets (the app can retrieve the current Standard card pool via an API or a static list). Also, except for basic lands, no more than 4 copies of a card are allowed in a deck [39] . Standard has its own banned list (cards deemed too strong are banned by Wizards and must be excluded by deck validation) [40] [41] . The app should allow selecting Standard format and then enforce card legality and count restrictions accordingly.

- **Commander (EDH)** – A very popular casual format, now officially supported. A Commander deck is **100 cards singleton** (each card except basic lands must be unique) [42] . One card is designated as the **Commander** (a legendary creature or special Planeswalker), and it sits in the command zone. Commander decks must follow a **color identity** rule: no card in the deck can have colors outside the commander's colors [43] . Players start at 40 life, and it's usually a 4-player free-for-all. Commander also has a **banned list** maintained by the format's Rules Committee (and some cards are banned specifically in Commander, like certain extremely powerful or multiplayer-problematic cards). The app's **deck builder** should enforce: exactly 100 cards total including commander, commander must be identified, color identity rules (e.g., if your commander is mono-red, you cannot have a blue card in the deck), singleton rule, and banned cards exclusion. The game engine must implement Commander-specific rules: command zone casting with additional costs (+2 mana each time you recast the commander) [44], commander damage tracking [4], and the option to "return commander to command zone" when it would go to graveyard/exile [19].

- **Modern** – A constructed format allowing cards from all sets since 8th Edition core set and Mirrodin (2003) onward. 60 card minimum, 4-of rule, its own banned list. No rotation (it's non-rotating eternal format). The app's format data should include which sets are legal in Modern (or use Scryfall legality flags). Deck validation must ban restricted cards (Modern currently only has banned, no one-of restricted like Vintage).

- **Pioneer** – Another non-rotating format, newer than Modern, which allows sets from Return to Ravnica (2012) onward. Also 60 card decks, 4-of rule, banned list. Similar handling as Modern but with a different starting set.

- **Legacy** – Eternal format with an even larger card pool (all sets, except a relatively large banned list). 60 card decks, 4-of rule. Legacy allows some older powerful cards that Modern bans, but still bans the most broken ones.

- **Vintage** – The largest card pool (almost every MTG card ever, except a few that are banned mainly for external reasons like ante or dexterity cards). Vintage uniquely has a **restricted list** instead of banning many cards – certain very powerful cards (like Black Lotus) are **restricted to 1 copy per deck** in Vintage [45]. The app's deck checker should, if Vintage format, allow one copy of restricted cards and four of others. Vintage is mostly of interest to hardcore players; support if possible.

- **Historic** – A *digital-only* format on MTG Arena that includes all cards available on Arena that are not in Standard [46]. This includes some special digital-only cards and anthologies. If we support Historic, we need to get card legality info from a source (likely Scryfall or Wizards Arena announcements) since it's an evolving set. Historic has its own banned list as well [47]. It's similar to an eternal format for Arena. The app could use the Scryfall "legalities" field for each card (which lists if a card is legal in historic) to validate decks.

- **Pauper** – A format where **only Common rarity cards** are allowed [48]. It can be played with any set's commons (including downshifts that were common on Magic Online). Deck construction is 60 cards, 4-of rule, but *all cards must have been printed as a common at some point*. The app should implement a check for card rarity legality when Pauper is selected (e.g., via card database info: many databases tag rarity, and whether a card was ever common). Pauper also has a small banned list of a few commons that proved too strong (e.g., [Astrolabe] was banned).

- **Others**: There are many other formats (Oathbreaker, Brawl, Explorer, Historic Brawl, Canadian Highlander, etc.). Initially, focus on the above popular ones. **Brawl** is like Standard Commander (60 card singleton with a Commander, using Standard-legal cards). **Oathbreaker** is a casual format similar to Commander but 60 cards including a Planeswalker as an Oathbreaker and a signature spell.

The app's **deck builder and validator** should be designed to easily incorporate format rules: - A data structure for each format (name, card pool or legality source, min/max deck size, allowed number of copies, life total, sideboard rules, banned/restricted lists, special rules). - Use this to validate deck submissions. For example, if format is Pauper and a card in deck is of rarity Rare, flag it as illegal. - The app can leverage online sources for banlists and card legality. **Scryfall's API** provides a `legalities` field per card (e.g., `"legalities": {"standard": "not_legal", "modern": "legal", "commander": "banned", ...}`) which can be used to check if a card is legal in a given format [49]. This is very useful for automation. The app could periodically update banlist info from there or have a config that is updated when bans happen.

Deck Legality Checks to implement: - Card count (min 60 for most constructed, exactly 100 for Commander, etc.). - Singleton rule if applicable (Commander, Brawl, etc.). - Commander color identity and other commander-specific deck building rules. - No banned cards (the app should have a list per format). - Restricted cards count (if Vintage, max 1 of each restricted). - For formats like Pauper or Artisan (commons only, or commons/uncommons only), verify rarities from card data. - Sideboard size (if format uses sideboards, typically 15 max in constructed formats). - Commander sideboards usually not used except in certain variants (some people play with a "wishboard" or Companion sideboard – if supporting Companion mechanic, that's another zone/deck to consider).

The UI should provide feedback if a deck violates any rule (e.g., highlight illegal cards, show an error "This deck has 5 copies of Lightning Bolt but only 4 are allowed").

Data Sources for Cards and Rules

To power the app with **all card data, images, and rules text**, we rely on external databases and APIs. Fortunately, the MTG community provides rich resources:

- **Scryfall API** – A comprehensive Magic card REST API that provides details for every MTG card ever printed. Scryfall offers card names, types, rules text, oracle rulings, set information, high-quality images, etc. It is free to use (with some rate limits) and very robust. For example, you can query a card by name or ID and get a JSON with all its attributes and even direct image links [49]. Scryfall also provides endpoints for bulk data (e.g., one huge file of all cards) and for **rulings** (Oracle rulings are clarifications or errata on cards) [50] [51]. Our app already uses Scryfall for images and card info – e.g., the `ScryfallClient` is integrated (we load images by card ID instead of scraping them manually [52]). We should continue to leverage Scryfall for up-to-date card data. Notably:
- **Card search**: Scryfall's `/cards/search` endpoint allows complex queries (name, type, text, etc.) which can power our search bar [53]. We might use Scryfall's query language to implement advanced filtering in the app.
- **Card details**: `/cards/:id` or `/cards/named?exact=...` gives full details. This includes Oracle text, which we need for rules text display, and **rulings** which we can fetch via `/cards/:id/rulings` [50].
- **Images**: Scryfall provides URLs for images at various sizes (small, normal, large, etc.) in the card data [49]. We can use those URLs to display card art (which we are doing, but ensure we respect their

terms: e.g., do not alter images, credit properly under their MIT license). The app can cache images for performance.

- **Legalities**: As mentioned, each card's legality in Standard/Modern/Commander/etc is given, which we can use for deck validation logic.
- **Bulk data**: Scryfall also has bulk JSON files (like one for all Oracle cards, one for all rulings, etc.) [51]. For offline or initial data seeding, we could download these and populate a local database.

- **Official MTG API (magicthegathering.io)** – An open Magic API run by the community that provides card and set data in JSON [54]. For instance, hitting `https://api.magicthegathering.io/v1/cards` returns cards (up to 100 per page) [55] [56]. It also has endpoints for sets, types, subtypes, etc. [57]. This API is free and has generous rate limits (5000 requests/hour) [58]. However, it may not be as up-to-date or comprehensive as Scryfall (Scryfall tends to update with new sets very quickly and include promotional cards, tokens, etc.). Still, it's a good resource and can serve as a backup or an additional source. They also provide **SDKs in multiple languages** (Python, JavaScript, Java, .NET, etc.) to simplify integration [59]. For example, the **mtg-sdk-python** library [60] can fetch cards easily without writing raw HTTP calls. We can incorporate such SDKs if it speeds up development (but be mindful of adding dependencies). Given we have already some data pipeline in the app (perhaps using MTGJSON or Scryfall directly), we can choose the best source for each use-case. The MTG API does not provide images or pricing (those are not official, Scryfall covers images/prices).

- **MTG JSON** – An open-source project that aggregates all card data from multiple sources and provides it in bulk JSON files [61]. MTGJSON is updated daily and includes card details, set lists, etc., in a convenient format. One of their products is an "AllPrintings.json" which contains every card printing ever, with all its data. This can be downloaded and used to populate a local database or for offline use. The advantage of MTGJSON is you get **complete data in one go** (it even includes things like banlist info, maybe, and prices via their partner TCGPlayer). The downside is the files are large (AllPrintings.json is quite huge, tens of MBs). But for an app that wants *all sets content available offline*, using MTGJSON to build our database might be ideal [62]. In fact, our `DATA_SOURCES.md` (planned) likely notes integrating MTGJSON for the card database. MTGJSON also has a **GraphQL API** (for Patreon supporters) and supports CSV/SQL dumps [63] [64]. For a free solution, we can consume their JSON dumps. The app could, for example, on first run, download the MTGJSON data to create a local SQLite database of cards. Then use Scryfall API for any dynamic or missing pieces (like up-to-the-day rulings or images). This hybrid approach ensures fast search (local DB) and completeness.

- **Gatherer** – Wizards of the Coast's official card database (web only, no official API). It has all cards and official Oracle text rulings. We generally don't need to scrape Gatherer because Scryfall and MTGJSON replicate that data. But if needed, Gatherer could be referenced for ruling clarifications (though Scryfall also includes Oracle rulings from Gatherer). Gatherer is also often slower and harder to integrate, so it's mentioned mostly for completeness.

- **Moxfield / Archidekt** – These are popular deck building websites. **Moxfield** in particular allows deck creation and has an unofficial API endpoint for fetching deck list by ID (as noted by users, hitting `api2.moxfield.com/v2/decks/all/<deckid>/export` gives a deck's card list [65]). However, Moxfield's API is *private*, with no official documentation [66]. They have said they sometimes assist on a case-by-case basis but no open API. We can still integrate with Moxfield by allowing users to import decks via copy-pasting decklist, or perhaps by parsing a Moxfield deck URL (they provide public deck

CSV/JSON export through that endpoint, but Cloudflare may complicate it). **Archidekt** has a public API for decks, and there's also **TappedOut** (older) and **Deckstats** – but focusing: Moxfield is widely used for Commander decks. So an import feature could use the deck ID approach (with user manually providing an ID or URL). If this proves unreliable, we can let users paste a deck in text form (Moxfield and others allow copying a deck to clipboard).

- **Rules Reference** – For embedding the rules in-app (as documentation), we have a few options:

- Use the **Comprehensive Rules** document (Wizards provides it in text form). It's 200+ pages of rules – too much for a user-facing doc, but we can include it as a reference file. Perhaps better is to have a simplified rules summary (which we are constructing in `RULES.md`). Wizards' site provides a **Basic Rules PDF** for beginners [67], which could be summarized or linked.
- **MTG Wiki (Fandom)** is a great resource for explanations of mechanics and can be used as reference for writing our docs (as we've done with citations). However, in-app we wouldn't link to wiki (to avoid requiring internet), but we can extract needed parts into our documentation.

- **Comprehensive Rules parsing**: Some projects parse the comprehensive rules into JSON or HTML. If needed, we could load portions of it for context help (like if a user clicks on a keyword, show its rule). This is a stretch goal, perhaps.

- **Oracle Rulings**: These are official clarifications for specific cards. Scryfall provides those via API [50]. We noticed in testing that **card rulings** were not showing in the app UI ("rulings for the cards aren't working" as the user noted). We should fix that by using Scryfall's rulings endpoint. For example, to get rulings for a card, we can call `GET /cards/<oracle_id>/rulings` or use the card's multiverse ID. The JSON returns a list of ruling entries (date and ruling text). The app can display these in the card detail panel. This will greatly help users understand complex card interactions. We should ensure to implement a call to fetch rulings when a card detail is shown (perhaps in the background to avoid slowing UI, since rulings can be loaded lazily).

- **Pricing**: While not a core requirement, some users like to see card prices. Scryfall includes price fields (usd, eur, tix for MTGO) for many cards. MTGJSON also offers some price data via TCGPlayer. This could be a nice-to-have feature (displaying deck cost, etc.). But given it's lower priority ("very very low priority" per user notes), we can defer this.

In summary, **our strategy** for data: - Use **MTGJSON or Scryfall bulk** to populate a local database of cards covering "all sets" (requirement: support content from *all sets*). - Use **Scryfall's API** for specific needs: image fetching (already in use), card search queries (since they support powerful search syntax), rulings fetching, and perhaps price updates. - The app's `DATA_SOURCES.md` (when updated) should document how to update the card database when new sets release (likely: run a script to download new data or ingest from Scryfall's API). The architecture might involve a local SQLite database accessed via an `MTGRepository` (which we have in code). Indeed, we see mention of FTS (Full Text Search) index was implemented [68] to search card text quickly. So likely we import card data from a JSON into SQLite. Agents continuing development should maintain that pipeline (e.g., when a new MTG set comes out, update the data file and trigger re-index).

Also, **deck import/export**: The app should support importing deck lists. This can be simple text files where each line is "Quantity CardName [SetCode]". Many deck sites export in a similar plaintext format. We should ensure the app can read/write such lists (perhaps .txt or .dek files). This will let users bring in decks from external sources easily. Integration with specific APIs (like pulling a deck from Moxfield by URL) can be added for convenience.

Existing Projects and References for MTG Rules Implementation

Developing a full MTG engine is a daunting task, but we have examples of **open-source projects** that have achieved (or attempted) this. Future agents can study these projects for inspiration, algorithms, or even data (where licenses permit):

- **XMage (magefree)** – A well-known open-source project that allows playing MTG online against others or AI. XMage includes *full rules enforcement for over 28,000 unique cards (73,000+ card printings)* [69]. It supports a wide range of formats (Standard, Modern, Commander, Pauper, etc., even some custom ones) and game modes (1v1, multiplayer up to 10 players, Two-Headed Giant, etc.) [70] [71]. XMage is written in Java and has a client-server architecture. Key things to learn:
- How they represent cards and handle the comprehensive rules. Likely, XMage has a rules engine that encodes card abilities and interactions in code, and a *database of card scripts or hard-coded card classes* for each unique card. Given the number of cards, they have a system to share common mechanics.
- XMage's handling of priority, the stack, triggers, and state-based actions is proven to work for thousands of cards. Looking at their source (on GitHub under magefree/mage) could provide insight into structuring our rules engine. For example, they likely have classes like `Game`, `Player`, `Card`, `Ability`, etc., and a system for continuous effects (layer system).
- They also support **AI opponents**, which is non-trivial in Magic. Their AI might be rule-based or use heuristics (since MTG is too complex for a simple minimax). For initial versions of our app, AI can be basic (random or scripted decisions), but eventually we want a smarter AI. XMage's AI code could be a reference.
- XMage's feature list confirms they support multiplayer and even things like **tournaments** with up to 16 players and draft/sealed events [72] [73]. Our app might not need tournament structure initially, but supporting draft/sealed (limited formats) in the future would be nice (we'd need a pack generator and a pick UI for Draft). XMage's approach to drafting could be examined when we get there.

- License: XMage is GPL. That means we can't directly integrate its code into our project if our project is differently licensed, but we can study it.

- **Forge** – Another community-driven project (Card-Forge/Forge on GitHub) which is a **single-player MTG engine** with AI. Forge focuses on **casual play vs AI and a "quest mode"**. It has thousands of cards implemented and an AI that can handle many of them. Forge is written in Java as well. They implement cards via scripting (some use Groovy scripts or similar). Studying Forge can give ideas on representing card abilities in data or script form rather than hardcoding everything. Forge also has an **Android app** (the UI for mobile), so their codebase shows how the logic can be separated from UI – useful for us as we modularize our game engine separate from UI. Forge's AI might be more developed for single-player than XMage's. The project is active (lots of commits). Key features from Forge:

- “Quest mode” (like a rogue-like using MTG games to accumulate cards) – outside our scope but interesting.
- The AI can draft and play limited as well.
- They have their own card database (likely derived from MTGJSON or similar) and content downloaders for images (as discussed in user comments, Forge has an in-app downloader for card pics, likely from Scryfall or Gatherer) [74] .
- License: Forge was historically GPL as well (need to confirm). It’s open source.

- **OpenMTG (Python)** – A smaller project (hlynurd/open-mtg) aiming to provide a framework for AI agents to play MTG in Python [75] . This might not be fully complete, but since our app is Python (if we’re using PyQt or similar, which it seems we do given `.py` files in the repo), open-mtg could have some useful code or at least concepts to borrow. We might be able to integrate bits of it if license allows (if it’s MIT or such).

- **MtgPure (Haskell/functional)** – A showcase project for a purely functional approach to modeling MTG rules [76] . Likely not directly useful to integrate, but reading about their modeling might give ideas for an architecture that separates game state, effects, and uses a deterministic approach.

- **Cockatrice** – An open-source virtual tabletop for MTG. Unlike XMage/Forge, Cockatrice **does not enforce rules** – it’s essentially a sandbox to play with someone manually (like a simulation of the board, relying on players to follow rules). Cockatrice uses an open card database (likely from MTGJSON) and is popular for playing casual games online with any card. While it doesn’t help with rules logic, Cockatrice is relevant for UI ideas, especially in multiplayer connectivity and how to present the game zones. Our app, however, **will enforce rules** (closer to XMage/Forge), making it more complex but also more user-friendly for solo play or learning.

- **Wizards’ digital offerings** (MTG Arena, MTGO): These are proprietary, but Arena’s interface and feature set serve as a target benchmark. MTG Arena has a slick UI, animations, priority timers, and handles a subset of MTG (Standard, Historic, and limited formats, plus some digital-only mechanics). We aim to simulate similar mechanics. We can’t see their code, but we can mimic their **user experience**: e.g., how the stack is displayed, how triggers are indicated, how the combat layout looks (attackers with arrows pointing to targets, etc.). Many players will expect a certain level of polish from playing Arena.

Important: While referencing these projects and sources, ensure compliance with licenses and do not directly copy code unless license-compatible. Our project seems to be open-source as well (since all these docs are public on GitHub). If we incorporate code or algorithms from somewhere, we must attribute or ensure license allows. For data (card text, rules text), Wizards has a fairly open **Fan Content Policy** that allows using card text and images in non-commercial projects with proper attribution [77] [78] . Scryfall’s data is under an MIT-like license for card text and an encouragement to credit them for images.

We should maintain a `DATA_SOURCES.md` (already in repo) to document where data comes from (e.g., “Using Scryfall API for images and rulings [52] , using MTGJSON for card database” etc.). This keeps things transparent for future maintainers.

UI Design and User Experience Considerations

A core instruction is that **the UI should be clean, organized, and human-friendly**. We are not limited by the text-based rules – we can present information visually in ways that make the game easier to play and the application easier to use. Some guiding points:

- **Deck Builder UI**: Building decks involves browsing a large card collection and assembling a list. Our current UI has a search panel and a deck list panel. We need to avoid clutter, even though there's a lot of info to show (card filters, search results, deck contents, maybe card details on hover). A typical approach:
- Use a two-panel layout: left side for search/filter and results, right side for the deck list (with maybe a small preview of selected card). Or top-bottom split.
- Allow the search results to be displayed with card names (maybe with mana cost icons) in a list or grid, and enable drag-and-drop or double-click to add to deck.
- The deck list should show counts and perhaps group by type or cost. Possibly have collapsible sections (lands, creatures, spells) for organization.
- **Deck stats** (mana curve, color distribution) are useful but could be shown on demand (maybe a "View Stats" button or a panel that can slide out). The user noted the "view stats stuff doesn't work" and considered merging it rather than a separate tab. So perhaps integrate a small stats summary in the deck panel (e.g., show total cards, average mana cost, maybe color pie chart icon).
- Keep filters intuitive: Possibly have text search plus tick boxes or dropdowns for format, color, type, etc. (If we have advanced search with our `SearchFilters` object, ensure the UI elements are clear).
- The deck builder should also facilitate **import/export**. For example, a text box to paste a deck list or a button to load from a file. And a save function.
- **Responsiveness**: If the user adjusts the window size, panels should resize. Also, if certain areas are too crowded, consider using tabs or hideable panels. For example, one suggestion was separating search and deck view into separate tabs if one-screen is too much. However, that could slow down the building process (switching back and forth). A good compromise might be a resizable divider: the user can allocate more space to search or deck as needed.

- **Card preview**: When the user hovers or selects a card in search or in deck list, show a card image and details (maybe on a side panel or as a floating tooltip). This helps to read card text easily. Our app already loads images (with a QThread for performance) [52]; ensuring that when browsing cards, there's a quick way to see the full card is important.

- **Gameplay UI (Battlefield)**: This is challenging but crucial. It should visually represent the game zones and permanents clearly:

- Each player's battlefield perhaps in a separate area/row. A common layout is your cards on bottom half, opponent(s) on top half (like MTG Arena). For multiple opponents, maybe arrange in a circle or in separate panels.
- Life totals and other player info (hand size, library count, graveyard count, poison counters, etc.) should be visible near each player's area.
- The stack should be visible when it has spells – possibly a side panel or a center overlay that shows spells and abilities in stack order. Arena shows the stack as icons on the right; MTGO shows it as a separate window.

- **Priority indicator**: in a digital game, it's helpful to show whose turn it is and if the system is waiting for the player to take an action or pass. Perhaps a "phase bar" or highlight on the active player's panel, and a prompt like "Your turn – Main Phase (priority to you)".
- **Phases/steps**: Could have a small UI element indicating the current phase (Untap, Upkeep, etc.) and maybe a way to stop at certain phases (MTGO style "stops"). For our AI games, we might auto-resolve a lot of empty phases, but a human vs human might want to respond in upkeep, etc. Possibly a checkbox for "hold priority at X step" can be advanced feature.
- **Combat**: When entering combat, the UI could highlight possible attackers, and on declaring attacks, show arrows from attacking creatures to defending player or planeswalker. Then allow opponent to declare blockers (highlight their creatures that can block and valid block assignments).
- We should show tapped/untapped status clearly (e.g., tilt or an icon). Summoning sickness (maybe grey out creatures that cannot attack yet).
- **Hovering over cards** should show their full details (maybe zoom the card).
- **Multiple players**: If supporting up to 16 players, the UI must scroll or be selectable – realistically, such a game is hard to display simultaneously. Perhaps only some formats like Commander (4 players) are common. For >4, maybe we assume a text-based log or something. But XMage supports up to 10 in free-for-all with a zoomable table. We could consider a way to scroll through players or have a simplified representation for far players. This is a complex UI challenge and we might defer the extreme cases.
- **Game log**: A text log of actions is very useful for transparency (especially when learning or debugging). We should maintain a log panel that says things like "Player1 cast Lightning Bolt targeting Player2" or "Goblin dies due to state-based action". This can be a toggleable panel in the UI.
- **Interactions**: The user needs ways to respond with instants/abilities. Perhaps a prompt like "Respond (Yes/No)" whenever the opponent does something and you have priority with possible actions. Or simply, if you have an interrupt window, highlight cards in hand that you could cast. We may implement an *auto-pass* if no action is possible (to speed play).
- For **AI vs Player**, allow the player to confirm when they have no responses (a "Pass Priority" button). Possibly allow setting stops (like "always stop at my upkeep").

- **Yield options**: In MTGO, you can yield to triggers or pass priority automatically on certain things to streamline. We might not need that immediately, but keep it in mind if the flow is too tedious.

- **Visual and Audio feedback**: Sounds and animations make the game feel alive. We have some **assets for visuals and sounds** in the repo (the user mentioned they exist). We should use these:

- For example, play a sound when a spell is cast, or when damage happens.
- Particle effects or simple animations for events: a swirling animation for spells on stack, a slash animation when creatures deal damage, a glow when a card is highlighted, etc. Even simple effects improve UX.
- The code likely has a `visual_effects` component (there was mention of `VISUAL_EFFECTS.md` planned). We can gradually implement these, focusing on not blocking game logic. Possibly use separate threads or Qt's animation framework so that the UI can animate without halting the game state updates.

- The design should remain clear even with effects: don't obscure card text or board state. Use concise animations.

- **Scaling**: The UI should handle different screen sizes. We may allow window resizing and maybe full-screen mode for a better game view. The card images can be scaled (maybe provide a zoom slider for card size). On smaller screens, perhaps a single-click on a card brings up a larger view.

- **Clean and Logical Layout**: Avoid too many separate windows or pop-ups. The earlier agent guidance suggests we consolidated windows into an `IntegratedMainWindow` [79]. That's good – one main window with tabs or panels for Deck Builder, Collection, Game, etc., instead of multiple pop-ups. Keep navigation simple: a main menu with "Deck Editor", "Single Player Game", "Multiplayer Game", "Collection Manager", "Help/Documentation".

- **In-App Help & Documentation**: The user specifically wants all this documentation **available in the app** for players and developers to reference [80] [81]. We have implemented a `DocumentationDialog` that loads Markdown files like `RULES.md`, `KEY_TERMS.md`, `TUTORIAL.md` [80]. We must keep these up-to-date. For example, after implementing new mechanics, update RULES.md or KEY_TERMS.md with that info. The documentation should be accessible via a "Help" menu. Possibly also context-sensitive help (like right-click a keyword on a card to see definition from KEY_TERMS). The agent guidance explicitly says future agents should update these as features or rules change [80]. So, maintaining this **master guidance** (the document we're writing now) and possibly splitting parts of it into user-facing docs (tutorial for how to play, glossary, etc.) is an ongoing task. We might use parts of this in the actual `doc/RULES.md` in the repo for players.

- **Consistency**: The UI design should follow platform conventions where possible (use of standard controls, clear icons). Also ensure consistent coding style to avoid things breaking (the user mentioned wanting consistent coding logic to prevent constant breakage). In UI terms, that means using the established patterns in Qt for signals/slots, not hacking around unless necessary.

- **Future Expansion**: As new formats or mechanics are added, the UI might need expanding (e.g., if we add Planechase, we'd need a place to show the Plane card and a "planeswalk" button, etc.). Design with flexibility: e.g., a small panel for "special zone" which can show a Plane or Scheme if game mode uses it. For now, focus on core.

Maintaining and Extending the Project (Agent Best Practices)

This guide serves as a foundation for both understanding Magic: The Gathering *and* understanding our project's goals. Each future AI agent (and developer) working on the MTG-App should keep the following in mind:

- **Keep Context and Continuity**: Always refer back to this document and previous **Agent Handoff Notes** to understand what's been done and why. We have extensive session summaries and a devlog – use them. For example, before reimplementing a feature, check if it already exists or was tried before. The repository contains archived files and TODO lists [82]. **Avoid duplicate code**: search the repo if a function or logic might already exist before adding new one [81]. Consistency and not reinventing the wheel will keep the codebase maintainable.

- **Update Documentation**: As you add features or fix rules, update the relevant docs (`RULES.md` for game rules changes, `TUTORIAL.md` for how to use features, `KEY_TERMS.md` for new keywords, etc.) [80] . Also update this guidance if our understanding of MTG rules expands or changes (e.g., if Wizards issues new rules or if we decide to adjust something in the engine). The docs are visible in-app to users and also serve as reference for new contributors, so they must stay accurate.

- **Testing and Stability**: Because "all coding logic should be consistent so we don't have shit breaking all the time," focus on writing tests for new features (we have a pytest framework in place). E.g., if adding a complex card interaction, write a test scenario for it. We have integration tests (like for search flows, etc.) [83] and should add for game mechanics too. A thorough test suite will catch regressions. The agent checklist suggests adding game engine integration tests for complex interactions [84] – do that when possible.

- **Clean Code and UI**: Maintain a **professional quality** in code structure and UI polish. The user plans this to be a published software eventually [81] . So adhere to good coding standards (clear names, comments explaining non-obvious logic especially for complex rule handling, no leftover debug prints, etc.). In UI, ensure things have proper labels, no overlapping widgets, and it looks neat (respect spacing, alignment).

- **Performance**: Loading all cards (over 20k) and running a full rules engine can be heavy. We have to be mindful of performance:

- Use efficient data structures (the FTS index for searching is one example of performance optimization that was done [68]).
- When updating the UI or game state, avoid freezes (use background threads for heavy I/O like image downloads [52] or computations).
- The rules engine might need to simulate many possibilities (especially AI); optimize critical loops, and possibly implement caching (e.g., remember results of static ability calculations if nothing changed).

- However, **do not prematurely optimize** at the cost of clarity – first get correctness, then profile if needed.

- **AI Development**: We have a requirement for AI opponents. Initially, we can implement a simple AI (random or using a basic heuristic). Over time, we should improve it. Possibly integrate an existing rule-based AI from a project like Forge, or design a scoring function for game states. This is a big task (AI could use Monte Carlo simulation, etc.), but incremental improvement is fine. Always ensure AI turns can be interrupted or are not too slow (maybe add a yield if searching deep, to keep UI responsive).

- **Multiplayer & Networking**: Eventually, playing with friends implies a networked multiplayer mode. That will require a server or peer-to-peer sync. We likely are not there yet (focus on single-player and hotseat first). But keep the code architecture flexible for that – e.g., a clear separation of game state model and UI, so that in future a network layer could drive the game state from remote inputs. XMage's client-server model is an existence proof that this is doable; maybe down the line, we can emulate a lighter version.

- **User Feedback and Iteration**: As features get implemented, testing the app as a user (the human "solo tester" mentioned [85]) will reveal UI pain points and bugs. Each agent should take note of any user feedback (from the project owner or testers) and address it. For instance, if the user says "game viewer looks weird" or asks if the layout makes sense for Magic's game board, consider adjusting the design to more standard layout. Iterate on UI/UX – small improvements like tooltips, shortcuts (maybe pressing numbers 1-20 to set life, etc.) can make a big difference in usability.

- **Communication**: Leave clear comments and Handoff notes for the next agent. Document not just what changes you made, but why, and any problems encountered. If something was particularly tricky (say implementing the stack handling for nested triggers), explain the approach in comments or in the devlog. That context will prevent future agents from accidentally breaking or duplicating that logic.

By following this guidance, future contributors will have a solid understanding of Magic: The Gathering rules and the application's architecture/goals. We have outlined the **fundamentals of the game** (rules, mechanics, formats) and provided links to valuable **resources** (official rules, card databases, open-source projects) to aid development [86] [69]. Use this as a foundation to continue building the MTG-App into a fully featured, robust digital Magic experience.

References and Further Reading

(The following sources provide additional details and were referenced in compiling this guide. They include official rules documentation, wiki explanations, and relevant project repositories for deeper insights.)

- Wizards of the Coast – **MTG Comprehensive Rules** (latest version PDF/HTML) [87] [86]
- MTG Wiki – *Comprehensive Rules summary* [86], *Turn Structure* [8] [9], *Card Types* [15], *Ending the Game* (win/lose conditions) [17] [7], *Commander (EDH) Rules* [88] [89], etc.
- Scryfall API Documentation – for card data, images, rulings, and search examples [49] [50].
- MagicTheGathering.io API Docs – official MTG card API (with SDKs) [55] [59].
- MTGJSON project – aggregated MTG data for all cards/sets [61] (daily updates and downloads).
- Open-Source MTG Projects: **XMage** (magefree) [69] [70], **Forge** [90], and others like OpenMTG, which can be found on GitHub for reference implementations and ideas.
- Agent Guidance & Project Documentation (MTG-App) – previous development logs, design notes, and to-do lists within our repository [82] [80] (e.g. `AGENT_GUIDANCE.md`, `TODO.md`, `FEATURES.md`, etc.), which future agents should continue to update.

---

[1] [2] [3] [4] [5] [6] [7] [17] [28] [29] [30] [31] [35] Ending the game - MTG Wiki
https://mtg.fandom.com/wiki/Ending_the_game

[8] [9] [10] [11] [12] [13] Turn structure - MTG Wiki
https://mtg.fandom.com/wiki/Turn_structure

[14] [15] Card type - MTG Wiki
https://mtg.fandom.com/wiki/Card_type

[16] [20] [21] [22] [23] [27] KEY_TERMS.md
https://github.com/brettadin/MTG-app/blob/7ff6aea8b21def1b7b84670180f8948466e248f6/doc/KEY_TERMS.md

18 19 32 33 42 43 44 88 89 MTG Commander Format | Magic: The Gathering
https://magic.wizards.com/en/formats/commander

24 25 26 Keyword ability - MTG Wiki
https://mtg.fandom.com/wiki/Keyword_ability

34 36 37 38 39 46 47 48 Magic: The Gathering formats - Wikipedia
https://en.wikipedia.org/wiki/Magic:_The_Gathering_formats

40 41 45 Banned & Restricted | Magic: The Gathering
https://magic.wizards.com/en/banned-restricted-list

49 53 Scryfall API - PublicAPI
https://publicapi.dev/scryfall-api

50 Ruling Objects · API Documentation - Scryfall
https://scryfall.com/docs/api/rulings

51 Bulk Data Files · API Documentation · Scryfall Magic: The Gathering ...
https://scryfall.com/docs/api/bulk-data

52 68 79 80 81 82 83 84 85 AGENT_GUIDANCE.md
https://github.com/brettadin/MTG-app/blob/7ff6aea8b21def1b7b84670180f8948466e248f6/doc/AGENT_GUIDANCE.md

54 MTG Developers
https://magicthegathering.io/

55 56 57 58 59 60 MTG API Docs - Magic: The Gathering API Documentation
https://docs.magicthegathering.io/

61 62 63 64 MTGJSON · MTGJSON.com · Portable formats for all Magic: The Gathering data
https://mtgjson.com/

65 Is there a way to use an api to get decks from moxfield using a script? : r/magicTCG
https://www.reddit.com/r/magicTCG/comments/1hcaai9/is_there_a_way_to_use_an_api_to_get_decks_from/

66 FAQ - Moxfield
https://moxfield.com/help/faq

67 explain the rules like i'm 5 : r/mtgrules - Reddit
https://www.reddit.com/r/mtgrules/comments/1ek6dtn/explain_the_rules_like_im_5/

69 70 71 72 73 GitHub - magefree/mage: XMage - Magic Another Game Engine
https://github.com/magefree/mage

74 Forge and XMage: The best free and open source rules engines for ‘Magic: the Gathering’ : r/magicTCG
https://www.reddit.com/r/magicTCG/comments/w628xt/forge_and_xmage_the_best_free_and_open_source/

75 GitHub - hlynurd/open-mtg: A python implementation of Magic
https://github.com/hlynurd/open-mtg

76 mtg-pure - The Gathering" card model and game engine. - GitHub
https://github.com/thomaseding/mtg-pure

77 78 87 Rules | Magic: The Gathering
https://magic.wizards.com/en/rules

86 Comprehensive Rules - MTG Wiki
https://mtg.fandom.com/wiki/Comprehensive_Rules

90 GitHub - Card-Forge/forge: An unofficial rules engine for the world's greatest card game.
https://github.com/Card-Forge/forge