ChatGPT

Project Overview

We propose an **online, web-based D&D 5E platform** that combines the feel of tabletop play with digital conveniences. Its **core innovation** is an AI Dungeon Master (DM) that can both generate real-time narrative and assist a human DM. The AI will handle creative storytelling, NPCs, and rules adjudication, while the UI automates bookkeeping (like tracking hit points, spells, and notes) to reduce busywork. Players join via private "rooms" (lobbies) by code, just like meeting around a virtual table. The initial MVP will focus on the essentials – AI DM, player chat, character sheets, and a basic map or encounter view – leaving advanced community features (public lobbies, fine-tuning marketplace, etc.) for later. All D&D 5th Edition rules will be supported at launch (the most popular edition of D&D), and no payment or account system is required initially (it will be open-source and free).

Key Features

- **AI Dungeon Master (Core Engine):** A fine-tuned LLM acts as a virtual DM, improvising a unique adventure. It will balance **creativity and rule logic**: reacting to player actions in real time while fairly applying D&D 5E mechanics [1] [2]. For example, the AI might describe a dragon encounter, roleplay NPCs with distinct personalities, and then handle a combat round by rolling dice. We will use state-of-the-art language models (e.g. GPT-4 via OpenAI, Anthropic's Claude, or local models via Ollama) because these can generate rich narrative and remember context across turns [3]. Under the hood, an **agentic orchestration layer** (using frameworks like LangChain or AutoGen) will manage long-term memory and planning: it can store facts (e.g. "the rogue stole a chicken two towns ago") in a vector database and retrieve them later to keep the story consistent [4] [5]. This lets the AI pursue story goals and adapt over multiple sessions (achieving "agents" that carry context and pursue objectives [6] [5]).

- **Multiplayer Lobbies:** Players (casual to expert) can create a **private game room** and share a code or link so friends can join. The system will use real-time web technology (WebSockets, e.g. Socket.IO) so all participants see messages and updates immediately. One user (the host) starts a session, and the AI DM joins automatically as the "Game Master" in that room. The backend (Node.js or Python) will track each room's state in MongoDB (similar to GameMaster.AI's architecture [7]), storing character data, session logs, and the ongoing narrative. Within a room, players can chat with each other and the AI, view shared maps or tokens, and manage their characters together, all in sync.

- **Automated Character and Session Tools:** The UI will include **digital character sheets** and automated trackers so that dice rolls, hit points, spell uses, and conditions update instantly without manual notes. For instance, after each turn the AI can log outcomes (damage dealt, XP gained) to the sheet. This mirrors existing tools (like D&D Beyond) that streamline play. Additionally, an **AI "notetaker"** feature (inspired by GameMaster.AI's notetaker) can summarize events or keep adventure logs [8]. Visual aids (e.g. top-down battle maps, token grids) will enhance immersion. These QoL features ensure tedious bookkeeping is automated while preserving the storytelling essence.

- **Real-Time & Pre-Generated Content:** The platform prioritizes real-time interactive play, where the AI responds instantly to player input. However, it can also handle pre-written modules: a user could preload a set of encounters, NPC descriptions, or maps so the AI has context even before the session starts. Retrieval-Augmented Generation (RAG) will allow the model to pull from these custom resources (lore, world maps, character backstories) when generating text [9]. For example, if the user uploads a custom map or NPC stat block, the AI can reference those details seamlessly.

- **Fine-Tunable Campaign DMs:** We will provide a **user-friendly interface for custom AI DMs**. Users can input text (campaign lore, character histories, world setting) and even select from suggested prompts or personas. For example, one demo model could be a *Lord of the Rings*-themed DM: we'll show sample training data (Tolkien lore snippets, themed prompts) and how it shapes the AI's style. Though full fine-tuning (training new weights) is complex, we'll **simulate** it in the MVP with a stub UI: users can enter a few example scenarios or choose keywords to influence the DM. (In practice, careful prompting plus memory often achieves ~90% of the effect of training a custom model [10].) We'll also layer *persona presets* like "You are Eleniel, a wise elven DM who speaks in riddles" to illustrate how tone can be changed [11]. Ultimately, this interface will let future users create and share fine-tuned AI DMs for any setting.

- **Tech Stack (Frontend):** The client will be a **TypeScript React app** (bundled with Vite and managed via pnpm). We'll build the UI with Shadcn UI/Radix components styled by Tailwind CSS. Radix provides accessible, composable primitives for dialogs, tabs, menus, etc., letting us quickly assemble an intuitive interface [12]. The layout will be responsive (mobile-friendly) with panels for chat, character sheets, and maps. State management (React Query or Zustand) will keep UI in sync with the server. For example, switching tabs for "Inventory" or "Spellbook" will instantly reflect the latest character data.

- **Tech Stack (Backend):** The server can be in **TypeScript (Node.js/Express)** or **Python (FastAPI)** – whichever best integrates the AI tooling. It will expose REST/WebSocket endpoints for the front end, handle game logic, and call the LLMs. We'll use **MongoDB** (e.g. Atlas) to store persistent data: player accounts (optional), saved campaigns, and game sessions [7]. When a new room is created, we generate a unique code and record the lobby data in Mongo. Each message in chat triggers a backend event that may (1) relay it to other players and (2) forward it to the AI DM agent. The server then returns the AI's response to everyone. In this way, the AI's turn is just another message in the chat flow.

- **AI Backends (LLMs):** We will support **multiple LLM providers**. By default, the host can choose to run models locally via Ollama – a framework for serving models like Llama 3, Mistral, etc., on one's own machine. If the host has an OpenAI or Anthropic API key, the system can use that instead (for example, calling GPT-4 or Claude 3). This dual approach ensures flexibility: games can be fully offline (privacy-respecting) or tap into premium models for maximum quality. (Our fine-tuned demo DMs can be distributed as Ollama model packages.) In any case, the orchestration layer will abstract these so the rest of the system just "calls the AI" without caring which engine runs it.

- **Orchestration & Memory:** The AI DM will be built as an **agentic system**. An orchestration framework (such as LangChain, AutoGen or a custom agent stack) will manage multi-step reasoning and memory. Concretely, the agent will have: a **short-term memory** (the current session's chat history and state), a **long-term memory** (stored in a vector DB like Pinecone/Weaviate) to recall past

events or lore [4] [13], and a **planner** to sequence events. For example, if players left off at a cliffhanger, the AI consults its memory to pick up that narrative thread. Tools or "executors" (like a dice-roller function) can be plugged in so the AI can perform rule-based actions. In short, the DM agent keeps context, sets goals (e.g. complete a quest), breaks them into actions, and updates memory as the game unfolds [6] [5].

- **Authentication & Deployment:** Initially, no user login or payments are required. The app will be open-source and free. We can host the frontend on a static site (Vercel or Netlify) and the backend on a free-tier server or container (Heroku, AWS, or self-hosted). Because Ollama requires local installation, we'll provide instructions for hosts who want to use local models. The web app must be secure (use HTTPS/WSS) and should scale to at least a few concurrent lobbies. We will design the server statelessly where possible (each game room's data in Mongo) so it can be scaled horizontally later.

AI Dungeon Master Details

Our AI DM is the heart of the system. It must **emulate a great human DM** by being imaginative, adaptive, and consistent. To do this, we will employ the following techniques:

- **Prompt Engineering & Personas:** We'll craft strong system prompts and examples. For instance, one prompt might say: "You are an enigmatic high-fantasy Dungeon Master. Narrate vividly, roleplay NPCs, and adapt to player choices." We may also use *persona presets* – e.g., "You are Kaelor, an ancient elven DM who loves riddles." These give the AI clear "character" guidelines [11]. During play, the AI sees recent chat (players' actions) and the prompt, then generates the DM's response. Experiments show that even without fine-tuning, models like GPT-4 can improvise rich narratives if steered well [3].

- **Memory & Retrieval:** To keep **long campaigns coherent**, we'll implement memory. Short-term context (the last few turns) is handled by the chat log. Long-term context uses RAG: we embed important world details and past events in a vector database. When needed, the AI retrieves relevant memory snippets to inform its response (e.g. *"Recall that villagers warned about bandits west of town."*). This means earlier choices influence later narration. Tools like Pinecone or Weaviate will store these memories [4]. In practice, this lets the AI recall that "the rogue stole a chicken two towns ago" and have NPCs react to it, just like a human DM would remember.

- **Rule Handling:** The AI will incorporate D&D 5E rules. It should, for example, know how to handle a combat round or what a Wisdom check means. In implementation, this can be partly hand-coded: if the story requires a die roll, the system can roll and feed the result into the narrative. The AI prompt might include phrases like "(Roll d20+3 for the goblin's attack)" and then the DM's response accounts for success or failure. Importantly, we ensure fairness: the backend validates rolls or rule outcomes so players trust the system. This follows the guideline that a DM (even an AI one) must apply rules consistently [2].

- **Creative Story Generation:** Using procedural generation, the AI can invent encounters, NPCs, and quests on-the-fly. Given the current setting (e.g., a dungeon level, the party's goal, and recent events), the AI might spontaneously generate "a ruined wizard's tower with a helpful ghost" or

unique loot. This dynamism keeps each game unpredictable and engaging. Over time, we can refine the AI with additional training data or user feedback to improve coherence and creativity.

Multiplayer Flow and UI

A typical session workflow will be:

1. **Room Creation:** A user opens the app, clicks “New Game,” and sets a campaign title. The system generates a lobby code. The host can choose the AI backend (e.g. GPT-4 via API or an Ollama model).

2. **Joining:** Other players enter the code to join the lobby. They see a chat interface and their character sheet (or create one on the spot). No account is needed; players can use nicknames.

3. **Character Setup:** Each player inputs their 5E character stats. The UI shows a digital character sheet with autosave. The AI (or a “Game” assistant) can help roll stats or apply level-up features.

4. **Gameplay:** Play proceeds in turns. Typically a player types an action (e.g., “I search the bookshelf for hidden clues”) in the chat. The server relays this to all clients. Simultaneously, the AI agent receives the player’s message and the conversation context, then generates the DM’s narration (“In the dusty library, you find a secret lever behind a tome…”) as a message back to all clients. Dice rolls can be done by players (via a built-in dice roller UI) or by the AI (via its tools). The UI updates initiative trackers, health bars, and other stats automatically as needed.

5. **Fine-Tuning Interaction (Demo):** If the host wants to “tune” the DM, they open the Fine-Tuning panel and answer guided prompts (e.g., mood, key NPC traits). In the MVP this doesn’t actually train a model but shows how it will influence prompts. For example, selecting “Grimdark World” might prepend the system prompt with that style.

Throughout, the **UI is built on Radix/Shadcn** components, ensuring accessibility and rapid development [12]. Tabs or collapsible sections let players view maps, inventories, and rules reference on demand. An activity log pane streams the conversation (labeled “DM:” vs “Player:” in different colors to avoid confusion). We avoid clutter: only essential buttons (like “Roll D20” or “End Turn”) are present, with keyboard shortcuts for power users.

Architecture and Tech Stack

The high-level architecture is as follows:

- **Frontend (Client):** A React SPA (TypeScript) served to the browser. It connects via HTTPS to the backend and opens a WebSocket (wss://) for live updates. We use Tailwind CSS for styling and Radix UI primitives for widgets. Shadcn’s component library speeds up building custom UI on top of Radix. State is managed with React Query (for server data) and local React state or Zustand for UI (e.g. modal open/close).

- **Backend (Server):** A stateless API server (Node.js/Express or Python) with WebSocket support. On “/create-room” it allocates a new lobby entry in MongoDB with a unique code. On “/join” it checks the

code and adds the player to that room. The WebSocket layer listens for player messages; upon receipt, it broadcasts them and also triggers the AI agent. For AI calls, the server uses one of:

- **Ollama CLI/HTTP:** If local, it invokes `ollama run` for the chosen model.
- **OpenAI/Anthropic API:** If remote, it sends a prompt to the cloud API. The response is parsed and sent back to clients. The server also handles dice rolls and basic mechanics when needed (either automatically or by checking the AI's suggestions).
- **Database:** MongoDB stores all persistent data. Collections include *rooms* (with current state, turn order, memory embeddings), *players* (optional accounts or nicknames), *characters* (stats, inventory), and *campaigns/templates* (for preset fine-tuned DMs). This ensures sessions can be resumed. GameMaster.AI similarly uses MongoDB to save games [7].
- **AI Orchestration:** We will likely use a framework like LangChain (node/py) to structure the DM agent. This handles *retrieval* of relevant lore (from user-uploaded docs or memory DB), *chain-of-thought* if needed, and tool usage. For instance, an agent step could check the "map tool" to describe the next room. The architecture may have multiple sub-agents (one for narrative, one for combat logic, etc.) coordinating via a central memory store [6] [5].
- **Security:** While MVP is open and without logins, we'll still secure the connections (HTTPS/WSS) and ensure room codes are unguessable. If later adding accounts or paywalls, we'll integrate OAuth or a payment gateway, but that's outside MVP scope.
- **Hosting & Deployment:** The frontend is static and can be deployed on platforms like Vercel or Netlify. The backend can run on AWS/GCP/Azure or Heroku in a container. For local play, instructions will be provided to install Ollama and run the server via Docker. We'll version-control and document the stack so other developers can contribute.

MVP Roadmap

To scope the MVP, we prioritize features that demonstrate the core experience:

1. **AI DM Engine (Agent):** Develop a working agent using GPT-4 or similar that can carry out an hour-long encounter. Show creative narrative and basic rule handling.
2. **Simple UI:** Build a lobby creation/join flow, a chat window, and a character sheet panel. Include one preset battleground map. Ensure real-time updates via WebSockets.
3. **Pre-Fine-Tuned DMs:** Provide two example DMs from Day 1 – e.g. *"Tolkien DM"* (LOTR style) and *"Classic Fantasy DM"*. Embed the training data (or personas) for these as documentation so users see how they were made.
4. **Fake Fine-Tuner UI:** Implement a form where users input campaign keywords or upload a text file. This won't retrain the model yet, but it will adjust the prompt context or select a persona, hinting at future full fine-tuning.
5. **Memory Basics:** Start with a simple memory – e.g. log the last session's story and feed it as system prompt the next day. Show that the AI "remembers" the party's past choices.
6. **Rule Automation:** Automate basic mechanics (init rolls, HP tracking). Even a text-based health update ("Orc attacks for 7 damage – Alice's HP now 8") will suffice to prove concept.

7. **Open Source Release:** Ensure all code (frontend, backend, docs) is on GitHub. Include instructions for running locally with a test model (like GPT-4 with a dummy API key or a small local LLM via Ollama) so others can try it out.

Each of these will be documented in detail (with code samples, APIs, and diagrams). We will use citations to back up design choices (e.g. how LangChain is used for multi-step LLM tasks [4] [13] or how Radix/Tailwind accelerates UI dev [12]). With the MVP complete, follow-up phases can add community features, more rulesets, voice integration, and a public library of user-created DMs and campaigns.

Sources: Design principles and technical references are drawn from state-of-the-art AI Dungeon Master research and tools. For example, a recent guide notes that "prompt design, data organization, and inventive automation" are vital to simulating human DM traits [14] [2]. Likewise, GameMaster.AI – an open-source solo RPG assistant – uses GPT-3.5/4 with an "AI notetaker" [8], and our plan builds on these architectures. We will follow best practices (LLM chaining, memory storage, reactive UI) and cite their documentation and relevant literature throughout development to ensure a robust, creative platform.

---

[1] [2] [3] [4] [9] [10] [11] [14] How to Build an AI Dungeon Master for Tabletop RPGs | by Konna Giann | Apr, 2025 | Medium
https://medium.com/@kgiannopoulou4033/how-to-build-an-ai-dungeon-master-for-tabletop-rpgs-548b7dd6d1ee

[5] [6] [13] Agentic AI vs. LLMs: Understanding the Shift from Reactive to Proactive AI — Arion Research LLC
https://www.arionresearch.com/blog/agentic-ai-vs-llms-understanding-the-shift-from-reactive-to-proactive-ai

[7] [8] GitHub - deckofdmthings/GameMasterAI: An open-source AI Dungeon Master system
https://github.com/deckofdmthings/GameMasterAI

[12] Radix UI
https://www.radix-ui.com/