DMToolkit: Tokens, NPCs, Monsters, Potions, Items

Table of Contents

Tokens Module

1. Overview

The **Tokens** module provides a library interface for managing grid-based character markers, or "tokens," commonly used in virtual tabletops. It includes searching, creating, editing, and viewing token entries.

---

2. Tokens.jsx – Main View

Purpose:
Renders the token library UI including:

- Searchable list of token cards
- Conditional rendering of TokenForm
- Modal-style detail view using TokenDetail

Key State:

const [showForm, setShowForm] = useState(false);

const [searchTerm, setSearchTerm] = useState("");

const [selectedToken, setSelectedToken] = useState(null);

Mocking:
Populates mock token data using an array including tokenTemplate and hardcoded entries for display purposes only.

Search Filter:
Filters token cards by token.name.includes(searchTerm.toLowerCase())

Actions:

- + New Token toggles TokenForm
- Clicking a TokenCard opens the TokenDetail modal

---

3. TokenForm.jsx – Token Creation

Purpose:
Handles input for creating new tokens using controlled inputs.

Form Fields:

- Name, Display Name, HP, Max HP
- Initiative, Size (width, height), Rotation
- Image URL, Notes
- Visibility toggle (checkbox)

Data Flow:
onSubmit(formData) is called on form submission with the filled data.

Special Logic:

- width and height update nested formData.size
- isVisible toggled via checkbox
- Uses defaultValues from Tokens.jsx (mock data for now)

---

4. TokenCard.jsx – Summary Display

Purpose:
Displays a compact, clickable token preview.

Displayed Fields:

- Image, Display Name
- HP / Max HP
- Size in tiles

Actions:

- Entire card clickable via onClick() from props (triggers detail view)

---

5. TokenDetail.jsx – Full Info Modal

Purpose:
Displays full token information as a modal overlay.

Features:

- Basic attributes (Size, HP, Initiative, Rotation, Visibility)

- Status Conditions (if any)
- Active Effects (with icons and duration)
- Token Notes

Conditional Sections:

- Hides unused fields (e.g., effects/status conditions) to keep UI clean

---

6. Interactivity Summary

Component	State	Action	Outcome
Tokens.jsx	showForm	Toggle New Token	Opens TokenForm
Tokens.jsx	selectedToken	Select card	Opens TokenDetail
TokenForm	Form submit	onSubmit(formData)	Closes form (for now)
TokenCard	Click	onClick()	Triggers setSelectedToken()
TokenDetail	Close	onClose()	Hides modal

---

7. Mock Data Usage

- Pulled from tokenTemplate JSON and extended manually
- No dynamic add/edit/delete yet – form submit is stubbed

---

8. Future Integration Notes

- handleTokenSubmit should eventually:
    - Push new token into a stateful array or backend
    - Add backend calls (e.g., POST to campaign-specific token library)
- Add unique token IDs if user-generated tokens are implemented
- Implement “Edit” and “Delete” handlers on TokenCard

Items Module

1. Overview

The **Items** module allows DMs to create, browse, and manage magical and non-magical equipment in a campaign. It includes support for game mechanics like attunement, charges, and item effects.

---

2. Items.jsx – Main View

Purpose:
Displays a searchable list of items and controls the creation flow.

Key Features:

- Campaign-aware item listing via useOutletContext().currentCampaign
- Toggle to show or hide ItemForm
- Item filtering via a search term
- Detailed view via ItemDetail

Key State:

const [showForm, setShowForm] = useState(false);

const [searchTerm, setSearchTerm] = useState("");

const [selectedItem, setSelectedItem] = useState(null);

Item Source:
Currently uses static mock data via itemTemplate.content (repeated 10× as a stub).

UI Features:

- + New Item toggles ItemForm
- Filtering applies item.name.includes(searchTerm)
- Clicking an ItemCard opens ItemDetail

---

3. ItemForm.jsx – Item Creation

Purpose:
Handles input and validation for item creation.

Form Fields Include:

- Core: name, title, type, rarity
- Flags: magical, attunement required
- Mechanics: damage, properties, charges
- Metadata: weight, value, description, image
- Effects: name/description pairs
- Campaign assignment (via currentCampaign)

Data Model:

```
{
 name, title, type, rarity,
 isMagical, attunementRequired,
 damage, properties, charges,
 effects: [{ name, desc }],
 description, weight, value,
 image, campaigns: [currentCampaign]
}
```

Handlers:

- handleChange – Controlled input update
- addEffect / handleListChange – Manages nested effects array
- onSubmit(formData, currentCampaign) – Called with the final item

---

4. ItemCard.jsx – Summary Display

Purpose:
Compact card UI for a single item.

Displayed Fields:

- Image, Name
- Type & Rarity
- Tags: Magical / Attunement
- Description preview (first 100 characters)

Actions:

- Edit and Delete buttons are shown but not yet functional
- Clicking the card triggers onClick to open the detail modal

---

5. ItemDetail.jsx – Full Info Modal

Purpose:
Displays complete item information on parchment-style overlay.

Includes:

- Header: name, type, rarity
- Stat block: magic status, attunement, damage, properties, weight/value, charges
- Long-form description
- Effects list (if any)

Effect Rendering:

```
<div className={styles.entryBlock}>
  <h3>{effect.name}</h3>
  <p>{effect.desc}</p>
</div>
```

Interactions:

- Click overlay to dismiss (modal behavior)
- onClose() is passed as prop to cleanly dismiss view

---

6. Interactivity Summary

Component	State	Action	Outcome
Items.jsx	showForm	Toggle New Item	Opens ItemForm
Items.jsx	selectedItem	Select card	Opens ItemDetail
ItemForm	Form submit	onSubmit()	Closes form (stub)
ItemCard	Click	onClick()	Triggers setSelectedItem
ItemDetail	Close button	onClose()	Hides modal

---

7. Mock Data Usage

- All item entries are based on itemTemplate.content, imported statically.
- Display logic is repeated 10× to simulate multiple entries.
- No item persistence or actual CRUD functionality yet.

---

8. Future Integration Notes

- handleItemSubmit should push item into persistent state/backend.
- Edit and Delete actions need to be hooked into state or API logic.
- Attachments like item icons or conditional effects could be supported in the form.
- Consider validation or tooltips for damage/rarity/type fields to guide user entry.

Monsters Module

1. Overview

The **Monsters** module is designed to manage stat blocks for adversaries, beasts, and fantasy creatures. It supports complex data structures such as ability scores, traits, actions, and senses, and is tailored for rich game mechanics integration.

---

2. Monsters.jsx – Main View

Purpose:
Serves as the dashboard for all monsters associated with the selected campaign.

Key Features:

- Campaign context pulled via useOutletContext()
- Search bar filters monsters by name
- Toggleable MonsterForm for creation
- Clickable MonsterCard opens a MonsterDetail modal

State:

const [showForm, setShowForm] = useState(false);

const [searchTerm, setSearchTerm] = useState("");

const [selectedMonster, setSelectedMonster] = useState(null);

Data Source:
Uses monsterTemplate.content, mapped 10 times as placeholder.

---

3. MonsterForm.jsx – Monster Creation

Purpose:
Handles entry of detailed monster information, suitable for 5e-style creatures.

Field Categories:

- **Overview:** name, size, type, alignment, initiative, image, description
- **Stats:** armor class, HP, hit dice

- **Speed:** walk, fly, swim, climb, burrow
- **Ability Scores:** STR, DEX, CON, INT, WIS, CHA
- **Saves & Skills:** CON, Stealth
- **Senses:** darkvision, blindsight, etc.
- **Language, CR, Proficiency**
- **Trait/Action Lists:** list of named, described entries

Data Structure:

```
{
 name, size, type, alignment, initiative,
 image, description, armorClass, hitPoints,
 hitDice, speed: {}, abilityScores: {},
 savingThrows: {}, skills: {}, senses: {},
 languages, challengeRating, proficiencyBonus,
 traits: [{ name, desc }], actions: [{ name, desc }],
 campaigns: [currentCampaign]
}
```

Handlers:

- Generic handleChange for nested or flat updates
- handleListChange and addListItem for repeatable fields (traits/actions)
- Submits via onSubmit(formData, currentCampaign)

---

4. MonsterCard.jsx – Summary Display

Purpose:
Displays monster preview information in grid view.

Shown Info:

- Image and name
- Type (size/type/alignment)

- AC, HP
- Challenge Rating

Actions:

- Edit and Delete buttons (not hooked up)
- onClick triggers detail modal

---

5. MonsterDetail.jsx – Full Info Modal

Purpose:
Provides a complete stat block display, styled over a parchment background.

Sections:

- **Header:** name, type, size, alignment
- **Stat Block:** AC, HP (and hit dice), initiative, proficiency, CR, speed
- **Ability Scores:** rendered in a grid
- **Saves/Skills:** single-value display
- **Senses, Languages, Description**
- **Traits & Actions:** rendered as entry blocks with title and description

Render Helpers:

- renderAbilityScores() – maps 6 scores visually
- renderEntry() – used for traits/actions

---

6. Interactivity Summary

Component	State	Action	Outcome
Monsters.jsx	showForm	Toggle New Monster	Opens MonsterForm
Monsters.jsx	selectedMonster	Select card	Opens MonsterDetail modal
MonsterForm	Submit	onSubmit()	Closes form (stubbed for now)
MonsterCard	Click	onClick()	Triggers setSelectedMonster()

Component	State	Action	Outcome
MonsterDetail	Close	onClose()	Hides modal

---

7. Mock Data Usage

- Pulled from monsterTemplate.content
- List is hardcoded via .map() loop, no dynamic storage
- Form submission has a stub only – no persistence

---

8. Future Integration Notes

- Add support for:
    - Additional fields (legendary/lair actions, resistances, conditions, etc.)
    - Real-time HP tracking or encounter tagging
- Persist monster data per campaign
- Hook up Edit/Delete buttons on card
- Consider mobile-friendly stat block presentation for quick reference

NPCs Module

1. Overview

The **NPCs** module helps Dungeon Masters build a library of characters that populate the game world — including allies, quest givers, rivals, and townsfolk. NPCs are statted like monsters, but with more social and narrative metadata such as occupation, background, and personality hooks.

---

2. NPCs.jsx – Main View

Purpose:
Displays the campaign's NPC list, with controls for searching and creating characters.

Key Features:

- Pulls currentCampaign via useOutletContext()
- Renders cards for NPCs with filtering by name
- Shows and hides NPCForm
- Opens NPCDetail modal on selection

State Management:

const [showForm, setShowForm] = useState(false);

const [searchTerm, setSearchTerm] = useState("");

const [selectedNPC, setSelectedNPC] = useState(null);

Data Source:

- Uses static npcTemplate.content, repeated 10× as placeholders

---

3. NPCForm.jsx – NPC Creation

Purpose:
Collects full stat block and story-driven details about a character.

Core Fields:

- **Role Data:** name, class, race, gender, age, occupation, background

- **Combat Stats:** armor class, hit points, hit dice, proficiency bonus, CR
- **Movement & Senses:** walk, fly, swim, climb, burrow; passivePerception
- **Ability Scores:** STR, DEX, CON, INT, WIS, CHA
- **Skills & Saves:** keyed dictionary input
- **Description & Image URL**
- **Traits / Actions:** repeatable name/desc blocks

Form Mechanics:

- Uses controlled inputs, nested object handling (abilityScores.str, etc.)
- Adds traits/actions dynamically via addListItem(key)
- onSubmit(formData, currentCampaign) is called on form submit

---

4. NPCCard.jsx – Summary Display

Purpose:
Compact, tappable representation of an NPC for the card grid.

Display Fields:

- Portrait
- Name
- Class & Race
- HP / AC
- Occupation

Actions:

- Edit and Delete buttons shown but unimplemented
- onClick selects the card and triggers setSelectedNPC

---

5. NPCDetail.jsx – Full Info Modal

Purpose:
Displays the full stat block and narrative data for an NPC.

Sections:

- **Header:** name, class, race, alignment, occupation
- **Stats:** AC, HP, CR, proficiency, speed
- **Ability Scores:** full grid
- **Saves & Skills:** rendered as comma-separated key-value pairs
- **Languages**
- **Description**
- **Traits & Actions:** rendered as expandable blocks with name and body

Conditional Display:

- Traits/actions only shown if populated
- Close button triggers onClose()

---

6. Interactivity Summary

Component	State	Action	Outcome
NPCs.jsx	showForm	Toggle New NPC	Opens NPCForm
NPCs.jsx	selectedNPC	Click card	Opens NPCDetail modal
NPCForm	Submit	onSubmit()	Closes form (stub only)
NPCCard	Click	onClick()	Sets selected NPC
NPCDetail	Close	onClose()	Dismisses detail modal

---

7. Mock Data Usage

- Driven from static npcTemplate.content
- Placeholder array .map()d in NPCs.jsx
- No live persistence or edit/delete functionality

---

8. Future Integration Notes

- Add personality and relationship metadata (e.g., likes, motivations, factions)
- Implement edit/delete functionality
- Add tagging (e.g., town, faction, enemy) for better filtering
- Consider rich text fields for backgrounds/descriptions
- Connect to encounter builder or dialogue manager in future

Potions Module

1. Overview

The **Potions** module is tailored for consumable magic items that typically provide temporary effects, healing, or utility boosts. It simplifies the stat block to emphasize effects, usage limits, and side effects, making it easier to manage during gameplay.

---

2. Potions.jsx – Main View

Purpose:
Displays the campaign's potion inventory and manages UI state for creation and detail views.

Features:

- Pulls currentCampaign from context
- Displays a searchable grid of potion cards
- Toggles the PotionForm for new entries
- Opens PotionDetail on selection

State:

const [showForm, setShowForm] = useState(false);

const [searchTerm, setSearchTerm] = useState("");

const [selectedPotion, setSelectedPotion] = useState(null);

Data Source:
Uses static potionTemplate.content, repeated 6× to simulate entries

---

3. PotionForm.jsx – Potion Creation

Purpose:
Captures concise and practical information required to define a potion.

Core Fields:

- Name, Rarity, Type (default: "Consumable")
- Effect, Duration, Side Effects

- Description (freeform)
- Cost, Weight, Uses (default: 1)
- Image URL

Data Shape:

{

name, rarity, type, effect, duration,

sideEffects, description,

cost, weight, uses, image,

campaigns: [currentCampaign]

}

Behavior:

- Controlled inputs updated via handleChange
- Submits data to onSubmit(formData, currentCampaign)
- Does not currently persist or validate data

---

4. PotionCard.jsx – Summary Display

Purpose:
Shows a quick view of a potion's key gameplay stats.

Display Fields:

- Image, Name
- Rarity
- Effect
- Cost

Card Actions:

- Edit and Delete buttons present but unhooked
- Clickable card opens detail modal via onClick

---

5. PotionDetail.jsx – Full Info Modal

Purpose:
Displays full potion data in a stylized parchment overlay.

Details Shown:

- **Header:** Name, Type, Rarity
- **Stat Block:** Effect, Duration, Side Effects, Cost, Weight, Uses
- **Description:** Full freeform text

Conditional Rendering:

- Side effects section only appears if potion.sideEffects is populated

Modal Behavior:

- onClose() dismisses the overlay when clicking outside or pressing ×

---

6. Interactivity Summary

Component	State	Action	Outcome
Potions.jsx	showForm	Toggle New Potion	Shows PotionForm
Potions.jsx	selectedPotion	Select a card	Opens PotionDetail modal
PotionForm	Submit	onSubmit()	Closes form (stub only)
PotionCard	Click	onClick()	Triggers setSelectedPotion
PotionDetail	Close	onClose()	Hides modal

---

7. Mock Data Usage

- Sourced from potionTemplate.content
- 6 placeholder entries rendered in UI
- No back-end or data store persistence yet

---

8. Future Integration Notes

- Add potion types or tags (e.g., healing, buff, utility) for filtering
- Enable stacked usage tracking (for multi-dose potions)
- Hook up Edit and Delete buttons
- Add support for ingredients or crafting metadata
- Consider duration timers or active-effect integration in combat tracking

Cheat Sheet Module

1. Overview

The Cheat Sheet module provides a lightweight reference system for quick-look rules, reminders, or procedures. It is designed for at-the-table efficiency, letting DMs curate and organize bite-sized gameplay entries.

---

2. CheatSheet.jsx – Main View

Purpose:
Renders a searchable library of Cheat Entries tied to the active campaign context.

State Management:

const [showForm, setShowForm] = useState(false);

const [searchTerm, setSearchTerm] = useState("");

const [selectedEntry, setSelectedEntry] = useState(null);

Behavior:

- Toggle form for new entry creation
- Filter entries via title
- Open CheatEntryDetail on click

Mocking:
Uses mock data (cheatTemplate.content) repeated 5× for display.

---

3. CheatEntryForm.jsx – Entry Creation

Purpose:
Handles controlled input fields to create a new Cheat Entry.

Form Fields:

- title
- description
- tags (comma-separated)
- campaigns (auto-assigned)

Submission:
Calls onSubmit(formData, currentCampaign) with cleaned tag array.

---

4. CheatEntryCard.jsx – Summary Display

Purpose:
Displays a brief summary of an entry with tags.

Components:

- Title
- Up to 3 tags
- Buttons for Edit/Delete (currently non-functional)

Actions:
Clicking the card triggers onClick() to show full entry modal.

---

5. CheatEntryDetail.jsx – Full Info Modal

Purpose:
Overlays a modal with full entry information.

Displays:

- Title
- Description
- Full list of tags

Modal Behavior:
Click outside or press ✕ to close.

---

6. Interactivity Summary

Component	State	Action	Outcome
CheatSheet.jsx	showForm	Toggle New Entry	Shows/hides form
CheatSheet.jsx	selectedEntry	Card click	Shows full detail modal

Component	State	Action	Outcome
CheatEntryForm	Submit	onSubmit()	Submits form, closes
CheatEntryCard	Click	onClick()	Triggers setSelectedEntry()
CheatEntryDetail	Click outside	onClose()	Dismisses modal

---

7. Mock Data Usage

Renders static entries from cheatTemplate.content, repeated 5 times.

---

8. Future Integration Notes

- Add persistent backend storage or in-app state
- Implement Edit/Delete logic on CheatEntryCard
- Tag filtering and advanced categorization
- Markdown or rich text formatting for descriptions

Rules Module

1. Overview

The Rules module enables Dungeon Masters to store and display homebrew or optional rules tied to their campaigns. Entries may include images, tags, and rich descriptions to ensure clarity and recall.

---

2. Rules.jsx – Main View

Purpose:
Presents a searchable rules library for the selected campaign.

State Management:

const [showForm, setShowForm] = useState(false);

const [searchTerm, setSearchTerm] = useState("");

const [selectedRule, setSelectedRule] = useState(null);

Features:

- Displays a set of Rule Cards
- Opens RuleForm to add entries
- Opens RuleDetail modal for full view

Mocking:
Mock rule data (ruleTemplate.content) is repeated to simulate multiple entries.

---

3. RuleForm.jsx – Rule Creation

Purpose:
Form component for controlled creation of a rule entry.

Fields:

- title
- description
- tags (comma-separated)
- image (optional)

- campaigns (auto-filled)

Submission:
onSubmit(formData, currentCampaign) with cleaned tags.

---

4. RuleCard.jsx – Summary Display

Purpose:
Compact preview card of rule data.

Contents:

- Optional image
- Title
- Up to 3 tags
- Edit/Delete buttons (non-functional)

Actions:
Clicking triggers setSelectedRule() to open modal.

---

5. RuleDetail.jsx – Full Info Modal

Purpose:
Displays full rule entry with rich formatting.

Displayed Fields:

- Title
- Optional image
- Description
- Full tag list

Modal Interaction:
Clicking outside or ✕ button triggers onClose().

---

6. Interactivity Summary

Component	State	Action	Outcome
Rules.jsx	showForm	Toggle New Rule	Shows/hides form
Rules.jsx	selectedRule	Card click	Opens detail modal
RuleForm	Submit	onSubmit()	Submits rule, closes form
RuleCard	Click	onClick()	Selects rule for modal
RuleDetail	Click outside	onClose()	Dismisses modal

---

7. Mock Data Usage

Uses ruleTemplate.content as the data stub, repeated 5×.

---

8. Future Integration Notes

- Hook into backend for CRUD operations
- Add markdown rendering for rule descriptions
- Enable Edit/Delete functionality in RuleCard
- Categorization or filtering based on tags or campaign context

Map Assets Module

1. Overview

The Map Assets module provides a visual library of resizable and descriptive environmental or object tokens that can be used during map-building or encounter design. These assets include dimensions, tags, and optional images.

---

2. MapAssets.jsx – Main View

Purpose:
Displays a campaign-scoped library of assets with search, add, and detail view functionality.

State Management:

const [showForm, setShowForm] = useState(false);

const [searchTerm, setSearchTerm] = useState("");

const [selectedAsset, setSelectedAsset] = useState(null);

Features:

- Search bar for asset name
-
    - New Asset button to open MapAssetForm
- Detail modal via MapAssetDetail

Mocking:
Uses mapAssetTemplate.content, rendered 6× for layout testing.

---

3. MapAssetForm.jsx – Asset Creation

Purpose:
Controlled form for entering map asset data.

Fields:

- name
- image (URL)

- width & height (in grid squares)
- description
- tags (comma-separated)
- campaigns (auto-filled)

Submission:
onSubmit(formData, currentCampaign) after sanitizing tags.

---

4. MapAssetCard.jsx – Summary Display

Purpose:
Compact visual display of asset data.

Display Elements:

- Image
- Name
- Dimensions
- Up to 3 tags
- Edit/Delete buttons (non-functional)

Interactions:
Click triggers setSelectedAsset(asset) to show modal.

---

5. MapAssetDetail.jsx – Full Info Modal

Purpose:
Displays full asset details in an overlay with image and structured information.

Displayed Fields:

- Name
- Dimensions
- Description
- Full tag list

Modal Mechanics:

Overlay can be dismissed with ✕ or outside click.

---

6. Interactivity Summary

Component	State	Action	Outcome
MapAssets.jsx	showForm	Toggle New Asset	Opens/closes form
MapAssets.jsx	selectedAsset	Card click	Opens full detail modal
MapAssetForm	Submit	onSubmit()	Closes form after submit
MapAssetCard	Click	onClick()	Opens detail view
MapAssetDetail	Outside click	onClose()	Hides detail modal

---

7. Mock Data Usage

Assets use mapAssetTemplate.content, repeated 6 times for simulated display.

---

8. Future Integration Notes

- Add persistence via backend or local storage
- Support asset grouping or categorization
- Implement Edit/Delete functionality
- Enrich asset data with usage notes or creator info