Campaign Module

Table of Contents

Section 1: Overview – Campaign Module

The **Campaign Module** in ArcanaTable serves as the central feature for tabletop roleplayers to organize and access game sessions. It includes support for campaign creation, joining via invite code, rule management, and session scheduling. The system is modular and UI-driven, facilitating both players and Dungeon Masters (DMs) in managing active campaigns.

The Campaign module is centered around:

- Campaign creation and join flows
- Campaign dashboard and listing interface
- Campaign detail viewing and editing
- Player and rule management per campaign
- Invite code handling for quick access

Campaign state is primarily maintained through local React component state and mock datasets (to be replaced with API/backend logic in future iterations). The interface emphasizes responsiveness and modular panel-based layouts.

Main Entry Point: CampaignDashboard.jsx

The CampaignDashboard.jsx component is the **central dashboard** for interacting with all campaigns a user is involved in.

Key responsibilities:

- Renders a list of campaigns using the CampaignCard component
- Provides CTA buttons for:
    - Creating a new campaign (/create-campaign)
    - Managing DM'd campaigns (/manage-campaign)
    - Joining existing campaigns via invite (/join-campaign)
- Displays campaign-specific overlays for detailed viewing
- Filters campaign-specific house rules and displays them contextually

This component handles **state logic** for selected campaigns (selectedCampaign), selected rule views (selectedRuleId), and transitions between dashboard and modal overlays.

Section 2: Campaign Lifecycle

The campaign lifecycle includes the processes of **creating**, **joining**, and **listing** campaigns. These flows are implemented via standalone components that feed data into the centralized dashboard view.

2.1 Create Campaign: CreateCampaign.jsx

This component is responsible for rendering a full-page form to initialize a new campaign. It allows the user to define the structure and metadata of a new game session.

Key features:

- **Form Fields**:
    - Campaign Name
    - Game System (e.g., D&D 5e, Pathfinder)
    - Description
    - Image URL / Campaign Visual
- **Custom Rules Assignment**:
    - Users can select from predefined rules (currently mocked via MOCK_RULES)
    - Rules are added or removed from form.rules using state-based manipulation
- **Preview Logic**:
    - Displays a preview image (or placeholder) as the user edits
- **Submission Handling**:
    - On form submit, data is logged (stubbed for future backend API)
    - Prepared for integration with persistent storage

The form's internal state is managed via useState, and the rule assignment system mirrors the one in the edit panel for consistency.

2.2 Join Campaign: JoinCampaign.jsx

This lightweight form allows users to join an existing campaign using a provided invite code.

Key aspects:

- **Invite Code Input**:
  - Single field entry for inviteCode, validated as required
- **Submission Handling**:
  - Logic is stubbed (placeholder) for future backend API interaction
- **UI Notes**:
  - Styled as a focused single-panel form for user simplicity
  - Integrated into the app shell with shared Navbar component

This component is minimal and designed for fast interaction, typically used by players receiving invite links.

2.3 Campaign Listing and Entry: CampaignDashboard.jsx + CampaignCard.jsx

Campaigns are listed on the main dashboard interface. The system pulls from a mocked campaign array (MOCK_CAMPAIGNS) and renders each using the CampaignCard component.

Behavioral notes:

- **Card Content**:
  - Image preview
  - Campaign name and game system
  - Invite code display
  - Player avatar strip (with fallback to default avatar)
- **Actions**:
  - Launch (stubbed)
  - Leave (stubbed)
  - Info (opens modal with campaign details)

Each campaign card is interactive and feeds back into the dashboard state for detailed inspection.

Section 3: Campaign Interaction Panels

This section outlines how campaign information is presented and interacted with once campaigns have been created or joined. Core display components are card-based and modal overlays, with support for contextual rule display.

3.1 Campaign Display Card: CampaignCard.jsx

The CampaignCard component is used within the CampaignDashboard.jsx to visually represent each campaign a user is part of.

Key UI Elements:

- **Image**: Displays either the campaign's custom image (imageUrl) or a placeholder.
- **Info Strip**: Shows campaign name, game system, and invite code.
- **Player Avatars**: Rendered as a strip of player icons with tooltips (fallbacks to a default avatar if not provided).
- **Actions**:
    - **Launch**: Placeholder for session entry point
    - **Leave**: Placeholder for campaign removal logic
    - **Info**: Triggers modal overlay to view campaign metadata

Design-wise, each card is compact and stylized for dashboard browsing, with minimal text and clear visual hierarchy.

3.2 Campaign Info Modal: CampaignDashboard.jsx

When a campaign card is clicked (via the "Info" button), an overlay modal is rendered from within CampaignDashboard.jsx.

Modal Features:

- **Detailed Header**: Campaign name and system
- **Player Listing**: Renders usernames of all current players
- **House Rule Selector**:
    - Dropdown list of assigned rules (filtered by campaign.name)

    - Displays title and description when a rule is selected
- **Next Session**: Currently hardcoded but reserved for dynamic scheduling

The modal captures click events to close itself and resets rule selections, keeping modal state scoped to its parent.

3.3 Rule Display and Selection: CampaignDashboard.jsx + Rules Integration

Rule integration is tightly coupled to the campaign modal experience:

- Campaigns maintain a rules field (array of rule IDs)
- Rules are filtered based on campaign name (rule.campaigns.includes(name))
- Upon selection:
    - A styled detail panel (ruleCard) shows rule name and description
    - Intended for player review, not editing

This logic uses mock data for now, simulating what would eventually come from persistent user-defined rule sets or a backend rule repository.

Section 4: Campaign Management

This portion of the Campaign Module allows Dungeon Masters to administer campaigns they created, including editing metadata, managing players, and assigning rules. These tools are accessible from the “Manage Your DM’d Campaigns” button in the main dashboard.

4.1 Campaign Management Entry Point: ManageCampaigns.jsx

This component serves as the administrative interface for DM-controlled campaigns.

Responsibilities:

- **Campaign Listing**:
    - Renders a grid of ManageCampaignCard components
    - Data is mocked using MOCK_CAMPAIGNS (to be replaced by live backend)
- **Editing Workflow**:
    - Clicking "Edit" opens a modal (EditCampaignOverlay) for campaign configuration
- **Overlay State Management**:
    - Uses editingCampaign, formData, and isClosing to manage modal behavior
    - Closing the overlay includes a brief animation delay using setTimeout

This component acts as the router-level shell for all DM campaign administration.

4.2 Campaign Management Card: ManageCampaignCard.jsx

A compact card used within the management view for each campaign owned by the user.

Displayed Information:

- **Campaign Name and Game System**
- **Invite Code**: Shown prominently
- **Next Session**: Shown as text (non-interactive)
- **Player List**: Full player usernames listed in a nested

- **Actions**:
    - **Edit**: Opens the editing overlay
    - **Leave**: Currently stubbed (intended for DM relinquishing ownership or campaign deletion)

Cards are structured similarly to CampaignCard.jsx, but adapted for administrative use.

4.3 Edit Campaign Overlay: EditCampaignOverlay.jsx

This full-screen modal allows DMs to update campaign settings in place.

Editable Fields:

- **TTRPG System**: Freeform text input
- **Next Session**: Freeform input (date/time suggested)
- **Player List**:
    - Displays usernames
    - Each player has a "Remove" button that triggers onRemovePlayer()
- **Rules Section**:
    - Lists assigned rules with the option to remove
    - Dropdown for assigning new rules (filtered dynamically)
    - State updates are done via onAddRule and onRemoveRule

Additional Elements:

- **Send Invite**: Button placeholder for future invite functionality
- **Close**: Dismisses the modal

The overlay relies on props passed from ManageCampaigns.jsx to control visibility, data flow, and updates. All updates are handled locally and not persisted beyond the session yet.

Section 5: Data and Configuration

This section outlines the shape of campaign data used throughout the module, the mocked datasets that currently simulate backend behavior, and how rule assignments are processed and represented in the UI.

5.1 Campaign Data Shape

Campaigns follow a standard data structure used across both creation and dashboard display components. The fields are consistent between Campaigns.json, MOCK_CAMPAIGNS, and runtime form state.

```
{
  "_id": "string",
  "name": "string",
  "gameSystem": "string",
  "inviteCode": "string",
  "creatorId": "string",
  "nextSession": "ISO8601 datetime string",
  "imageUrl": "string (optional)",
  "description": "string (optional)",
  "players": [
    {
      "_id": "string",
      "username": "string",
      "avatarUrl": "string (optional)"
    }
  ],
  "rules": ["rule-id-1", "rule-id-2"]
}
```

Notes:

- players is an array of user objects.
- rules is an array of rule IDs (used to filter and render rule descriptions).
- inviteCode is used as a string match in JoinCampaign.jsx.

5.2 Sample Campaigns: Campaigns.json

This static file provides example campaigns for local rendering and UI development.

Contents:

- Two full campaign entries:
    - “Dark of the Moon” (D&D 5e)
    - “The Broken Crown” (Pathfinder 2e)
- Each campaign includes multiple players, an invite code, and a scheduled nextSession.

This data mirrors what would be retrieved from an actual backend once integrated and serves as the mock baseline for development and testing.

5.3 Rule Assignment and Logic

Rule objects are defined in multiple places depending on context:

- CreateCampaign.jsx and ManageCampaigns.jsx both define a MOCK_RULES array
- Each rule follows this schema:

```
{
  "_id": "string",
  "title": "string",
  "description": "string",
  "tags": ["string"]
}
```

Assignment Workflow:

- When creating or editing a campaign:
    - Assigned rules are stored by ID
    - Display logic filters from the full list of available rules using .includes(rule._id)
- When viewing:
    - Rule objects are matched against the campaign name (rule.campaigns.includes(name)) to enable contextual display in the info modal

These assignments are currently local but designed to map cleanly to future API structures.