Campaigns Module

Table of Contents

Section 1: Overview – Campaign Module (Updated with Backend Integration)

Purpose

The Campaign Module is the core of ArcanaTable's TTRPG functionality. It enables users (both Dungeon Masters and players) to **create, join, view, and manage campaigns**, leveraging a persistent backend via a RESTful API and Supabase for media hosting.

Backend Integration

All campaign data (e.g., metadata, players, rules, session dates) is stored and managed via:

- **Express API endpoints** (campaignRoutes.js)
- **Campaign controller functions** (campaignController.js)
- **MongoDB schema** via Mongoose (Campaign.js)
- **Supabase** for image uploads (supabase.js)

Key Functionalities

- **Create Campaign**: Via CreateCampaign.jsx, campaigns are submitted to the backend using POST /api/campaigns.
- **Join Campaign**: JoinCampaign.jsx uses POST /api/campaigns/join to join by invite code.
- **View Campaigns**: CampaignDashboard.jsx loads all user-related campaigns with GET /api/campaigns.
- **Manage Campaigns**: ManageCampaigns.jsx fetches and allows modification of DM-owned campaigns via GET and PATCH requests.
- **Edit Campaign Details**: Rules, players, and metadata are updated through full modal interfaces with backend persistence.
- **Image Uploads**: Handled via Supabase (with POST /api/campaigns/upload endpoint).

Main Entry Point

CampaignDashboard.jsx acts as the landing interface for a user's campaigns.

Core responsibilities:

- Fetches user-related campaigns using their authentication token.
- Renders each campaign via CampaignCard.jsx.
- Enables modal inspection of campaigns.
- Provides navigation to:
    - /create-campaign
    - /join-campaign
    - /manage-campaign

Authentication Context

The AuthContext.jsx provides the JWT token and user data to authorize requests across the module.

Section 2: Campaign Lifecycle (Updated with Backend Integration)

This section outlines the lifecycle of a campaign: creation, joining, and listing. Each part of the lifecycle is powered by backend API calls, secured with bearer tokens via the AuthContext.

2.1 Create Campaign – CreateCampaign.jsx

Component Role:
Renders a full-page form allowing a DM to define a new campaign and submit it to the backend.

Key Backend Interactions:

- **Image Upload (optional):**
    - Endpoint: POST /api/campaigns/upload
    - Uploads an image file to Supabase and returns a URL.
- **Campaign Creation:**
    - Endpoint: POST /api/campaigns
    - Payload includes: name, gameSystem, description, imageUrl, and rules array.
    - Requires Authorization header with bearer token.

Frontend Logic:

- Uses useState to manage form values.
- Uses FormData for image handling.
- On success, redirects to /campaign-dashboard.

2.2 Join Campaign – JoinCampaign.jsx

Component Role:
A lightweight interface that allows players to join campaigns by entering an invite code.

Key Backend Interactions:

- **Join Request:**
    - Endpoint: POST /api/campaigns/join
    - Payload: { inviteCode }
    - Requires user token in Authorization header.

Frontend Logic:

- Validates input.
- Sends join request and displays confirmation or error.
- Navigates back to /campaign-dashboard upon success.

2.3 Campaign Listing & Entry – CampaignDashboard.jsx + CampaignCard.jsx

Component Role:
Shows a list of all campaigns a user is part of. Each is displayed with its visual, metadata, and actions.

Key Backend Interactions:

- **Campaign Fetching:**
    - Endpoint: GET /api/campaigns
    - Retrieves all campaigns where the user is a player or creator.

Features:

- **Delete Campaign:**
    - Handled in CampaignCard.jsx via DELETE /api/campaigns/:id.
    - Only visible to the campaign creator.
- **Info Modal:**
    - Shows campaign details, players, and rules in an overlay.
- **Navigation Buttons:**
    - Create, join, and manage campaign routes are linked from this page.

Section 3: Campaign Interaction Panels (Updated with Backend Integration)

This section describes how users interact with individual campaigns through dashboard elements, modals, and rule viewing components.

3.1 Campaign Display Card – CampaignCard.jsx

Component Role:
Renders each campaign in CampaignDashboard.jsx as a compact visual card.

Features:

- **Visuals:**
  - Displays campaign image (or placeholder), name, system, and invite code.
- **Player Avatars:**
  - Maps player avatars with fallback to default image.
- **Actions:**
  - Launch: Placeholder button for future session entry.
  - Info: Opens the detailed info modal.
  - Delete (creator only): Sends a DELETE request to /api/campaigns/:id with token.

Backend Notes:

- Deletion logic includes a confirmation prompt.
- On success, parent component (CampaignDashboard.jsx) updates state via onDelete.

3.2 Campaign Info Modal – CampaignDashboard.jsx

Component Role:
Renders an overlay when a campaign card's "Info" button is clicked.

Displays:

- Campaign title and system.

- List of players.
- Next session info.
- House rules (titles only — rule metadata is not yet dynamically fetched from a rule DB).

Backend Integration:

- Modal content is based on data previously loaded from GET /api/campaigns.

Planned Enhancements:

- Dynamic rule detail fetching.

3.3 Rule Display and Selection – CampaignDashboard.jsx + (future rules integration)

Current Implementation:

- Rules assigned to campaigns are stored as rule IDs.
- Display logic filters a mock list or locally passed rules.
- The modal can show rule titles/descriptions based on campaign rule assignments.

Design Notes:

- No backend fetch for full rule objects yet — this is still mock-driven (MOCK_RULES).
- A future backend feature would allow:
    - Fetching all rules per user/toolkit.
    - Dynamic detail rendering by rule ID.

Section 4: Campaign Management (Updated with Backend Integration)

This section covers administrative functionality for Dungeon Masters, including editing campaign metadata, managing players, and assigning house rules. These tools are accessible via the **"Manage Your DM'd Campaigns"** button from the dashboard.

4.1 Management Entry Point – ManageCampaigns.jsx

Component Role:
Provides the DM interface to view and edit their campaigns.

Backend Integration:

- **Fetch DM Campaigns:**
  - Endpoint: GET /api/campaigns/dm
  - Requires bearer token.
  - Returns all campaigns where the user is the creatorId.
- **Update Campaign:**
  - Endpoint: PATCH /api/campaigns/:id
  - Payload includes updated fields, especially nextSession (merged from date + time fields).
  - Requires bearer token.

Frontend Logic:

- Manages campaign state, current form, overlay visibility.
- On "Edit", opens the EditCampaignOverlay.jsx modal.
- On "Save", prepares a nextSession ISO string from the form fields and sends a PATCH request.

4.2 Management Card – ManageCampaignCard.jsx

Component Role:
Compact campaign card shown in the DM's grid view.

Displays:

- Campaign name and system.
- Invite code and formatted next session.
- Full player list (by username).
- “Edit” button — launches the overlay editor.

Differences from CampaignCard:

- No avatar rendering.
- Focused on admin-level info instead of player-focused visuals.

4.3 Edit Campaign Overlay – EditCampaignOverlay.jsx

Component Role:
Full-screen modal for editing a campaign’s metadata, rules, and players.

Editable Fields:

- **Game System** (text)
- **Next Session Date & Time**
- **Rules:**
    - Assigned rules are listed and removable.
    - Unassigned rules are available via a dropdown.
- **Players:**
    - Shows all players except the creator.
    - Each player has a “Remove” button (updates local state only).

Data Flow & Save Logic:

- Controlled via props from ManageCampaigns.jsx.
- Calls onSave() when the "Save Changes" button is pressed.
    - Triggers a PATCH request to update the backend.

Section 5: Data and Configuration (Updated with Backend Integration)

This section describes the structure of campaign data as it now flows through real backend services, replacing the old mock datasets.

5.1 Campaign Data Shape

Campaign objects stored in the database (and returned by the API) follow this schema:

```
{
  "_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"]
}
```

Backend Source:

- Defined in Campaign.js (Mongoose model).
- Created and modified via campaignController.js.

Supabase Integration:

- Images uploaded by the user are sent via POST /api/campaigns/upload, then stored on Supabase.
- Returned imageUrl is embedded in the campaign payload and saved to MongoDB.

5.2 Sample Campaigns (Deprecated)

Previously:

- Sample data was defined in Campaigns.json or inline MOCK_CAMPAIGNS.

Now:

- These have been fully replaced by real backend data retrieved from MongoDB.
- Static mock data is only used temporarily when backend calls fail or for dev prototyping.

5.3 Rule Assignment and Logic

Current Implementation:

- Rules are represented as objects with this shape:
- {
- "_id": "string",
- "title": "string",
- "description": "string",
- "tags": ["string"]
- }

Assignment:

- Assigned in both CreateCampaign.jsx and EditCampaignOverlay.jsx.
- Stored as an array of rule IDs in the rules field of each campaign.

Limitations / In-Progress:

- Rule selection is local — hardcoded mock rules (MOCK_RULES).
- Full rule objects are not yet fetched from a persistent backend or user rule library.
- Future improvements may include:
    - Rule library API (GET /api/rules)
    - Rules per user or organization
    - Tag-based filtering and searching