\*\*Development Guidelines for JSON-Based Classcraft-Style Web App\*\*

---

### 🔒 Project Scope Boundaries (for JSON-only version)

\*\*DO NOT:\*\*

- Use cloud databases (e.g., Supabase, Firebase)

- Implement OAuth or third-party login

- Require an internet connection to function

- Store data in SQL/PostgreSQL (we're simulating with JSON)

- Add features not described in MVP: no complex analytics, payment systems, or third-party integrations

\*\*DO:\*\*

- Store data in local `.json` files (using Node.js' `fs` module or similar)

- Run entirely offline in a browser or Electron

- Keep the data logic and rules identical to the Classcraft system described in the research

---

### ✅ Core Development Directions

#### 1. 🧠 Game Logic Layer (Back-End Rules & Calculations)

\*\*Folder:\*\* `/backend/`

\*\*Environment:\*\* Node.js

All logic for XP, HP, AP, and class powers must live here in JS modules.

Use JSON files as the "database" (e.g., `users.json`, `characters.json`, `teams.json`, `xp\_log.json`).

\*\*Game Logic Modules to Build:\*\*

- `xpSystem.js` → Handles XP gain/loss

- `hpSystem.js` → Handles HP loss, fall-in-battle checks

- `apSystem.js` → Handles AP costs and regeneration

- `powerSystem.js` → Validates power usage rules (e.g., AP cost, prerequisites, team impact)

- `teamSystem.js` → Applies shared team consequences/rewards

- `eventSystem.js` → Handles daily random events and their triggers

All modifications to the state should result in a `.writeFileSync()` or equivalent JSON write.

---

#### 2. 🛠 File-Based “Database” Structure (Stored as JSON)

Create a `/data/` folder with JSON files like:

```bash

/data/users.json

/data/characters.json

/data/teams.json

/data/powers.json

/data/quests.json

/data/events.json

```

\*\*Important:\*\*

- Design the schema to match the \*\*relational structure\*\*, but keep it nested and easy to query

- For logs, append to arrays rather than overwriting (e.g., all XP logs are cumulative)

---

#### 3. 👥 Front-End Interface

\*\*Folder:\*\* `/frontend/`

\*\*Framework:\*\* React (preferred), but any SPA framework is fine

\*\*Views to Include:\*\*

- `DashboardView.jsx` → XP, HP, AP bars, powers, recent events

- `TeamView.jsx` → Show teammate status, allow class powers like “Protect”

- `QuestView.jsx` → Narrative quest display with steps and branching

- `TeacherView.jsx` → Only shown to `user.role === 'teacher'` with controls for:

- Awarding XP

- Customizing behavior triggers

- Creating/editing quests

\*\*JSON Communication:\*\*

Frontend will use `fetch()` or `ipcRenderer` (Electron) to call local APIs exposed by the Node backend (via Express or a light API interface).

---

#### 4. ✿ Minimum API Endpoints (Simulated or Live with Express)

All endpoints affect only local JSON.

\*\*Basic Routes:\*\*

```

POST /api/characters/:id/xp → Add XP

POST /api/characters/:id/hp → Deduct HP

POST /api/characters/:id/power → Use a class power

GET /api/characters/:id/status → Return XP, HP, AP, level, powers

GET /api/team/:id → Get team data

GET /api/quest/:id → Get quest progress

POST /api/quest/:id/advance → Move forward in a quest

```

If not using Express, simulate this flow with local function calls from frontend (Electron IPC or bundled modules).

---

#### 5. 🔄 Real-Time / Local Refreshes

Since it’s local:

- You can skip WebSockets

- Use polling or direct function calls to sync data views on the frontend (React `useEffect + setInterval`)

---

#### 6. 🧚‍♂️ Testing Mode & Debugging

Add a developer mode toggle to:

- Auto-load mock student and teacher users

- Simulate daily random events

- Preload a test team with all classes

---

#### 7. 📄 Sample JSON Structures (Guideline Only)

\*\*`/data/characters.json`\*\*

```json

[

{

"id": "c1",

"userId": "u1",

"name": "Aiden",

"class": "Warrior",

"level": 4,

"xp": 1400,

"hp": 60,

"ap": 15,

"powers": ["protect1", "firstAid"],

"teamId": "t1"

}

]

```

\*\*`/data/powers.json`\*\*

```json

[

{

"id": "protect1",

"name": "Protect 1",

"class": "Warrior",

"tier": 1,

"cost": 10,

"description": "Take 10 damage for a teammate, only take 80%."

}

]

```

---

### ✅ Success Criteria

- [ ] Full XP/HP/AP system logic implemented

- [ ] Characters can use powers with proper cost checks

- [ ] Quests work with branching and reward XP

- [ ] Local JSON used for all persistent state

- [ ] Teacher dashboard allows real-time gamification control

- [ ] Frontend displays XP/HP/AP/powers and updates on use

- [ ] No internet connection required to run