# CasterFire Web Application

By Jason Holtzen

## Part A: Comprehensive Summary

### The Problem

The second edition of the Pathfinder Roleplaying Game (PF2E) came out in 2020. Shortly after, the world began to experience an epidemic. As games around crowded tables became a public danger, players turned to digital video conferencing. PF2E suffered from a major lack of digital tooling to help players manage their characters, making it more difficult to play the game digitally. This setback was exponentially worse for those who chose to play spell casting characters. The first two books released for the system contain eight hundred spells, which are spread across both books and span hundreds of pages. Given that an average character will only have access to around 50 spells, there needed to be a better solution.

### The Solution

Enter CasterFire, a digital spell book for players who love to play spell casting characters. The application will allow players to login from any computer and be able to save spells for any number of characters that they are playing in a separate spell book for each character. Some players also like to create their own custom spells. CasterFire has that covered. too. When a custom spell is created, it is saved to the user, able to be added to any of their characters. Intuitive filtering and searching round out the application, enabling players to find just the right spell for every occasion.

### Methodology

The Agile methodology for software development will be used to develop the application, allowing the developer the ability to make changes more quickly when problems arise or new features become obvious during development.

### Deliverables

1. Acquire spell data (which is freely available through the Open Gaming License).
2. Make spell data available in application.
3. Make spell data searchable.
4. Make spell data filterable based on important fields.
5. Create authentication for users to be able to sign in to the application.
6. Create character create/edit interface and APIs
7. Create ability to add spells to characters.
8. Create custom spell interface.
9. Soft Launch site on public internet.
10. End to End user testing.
11. Go live.

### Validation and Verification

Automated testing will be implemented, and developer will use Jest for automated testing in places where the algorithms are more complex, like the filtering and searching. End-to-end testing will be used to validate and verify features and deliverables.

### Implementation Plan

1. Evaluation of framework, to make sure it will meet the needs of the application.
2. Write code locally.
3. Write Test Scripts.
4. Publish to Netlify.
5. End to End testing.
6. Repeat steps 2-5 until testers sign off on completion.
7. Go live.

### Validation and Verification

Automated testing will be implemented, and developer will use Jest for automated testing in places where the algorithms are more complex, like the filtering and searching.

### Programming Environment and Costs

* Languages: HTML, SCSS, and JavaScript
* JavaScript Framework: Svelte and Sapper (Note, this was later changed to SvelteKit, as the Sapper project was scraped and replaced with SvelteKit)
* Server Framework: Nodejs
* Automated Testing Framework: Jest
* Validation Testing: End-to-end user tests
* Hosting: Netlify
* Database: MongoDB Atlas Cloud
* Source Management: GitHub
* IDE: Visual Studio Code
* Costs: The project will use free tiers of MongoDB Atlas and Netlify, which only require payment past a generous threshold, which CasterFire should not surpassed during testing/deployment.
* Human Resources: One developer and four testers for end-to-end testing.

### Projected Timeline with Resources

Given that the project will be using the agile methodology, sprints will be in 1-week increments. Development is expected to be from September 1, 2021 to December 1, 2021, though this could change, as development continues and testing warrants feature changes. After this, testers will complete two weeks of end-to-end testing, leaving two weeks for deployment by January 1st, 2021. This timeline allows for 6 weeks of additional development, in case features or bugs are brought about during testing or development, making the latest possible delivery date February 14th, 2022.

* Development (3 months): 1 developer
* Feature Testing (1 day per week of development): 1-4 testers, testing as features are released
* End-to-end Testing (2 weeks): 4 testers testing full functionality of the application.
* Deployment (2 weeks): 1 developer

## Part B

Application Type: Web

Code: Will be included in zip file. README.md file in project directory contains directions for developer setup to be able to run the project locally. Note: Developer setup instructions reference a .env file, which will already be included in the zip. That step can be skipped.

Search Functionality: Multiple row results and criteria are included via the Filters bar and the Search box.

Database: MongoDB database can be accessed by going to <https://cloud.mongodb.com> and using the credentials below. Note that access is read only for this account, and it will be removed once this evaluation is complete.

* Username: [jason@holtzen.rocks](mailto:jason@holtzen.rocks)
* Password: BQ)\*L&fU4TXs7Ez

Once logged in:

* From left menu, click on Databases link
* Under the Database Deployments area, click on pf-2-spellbook
* Next, click on Collections from the navigation under the pf-2-spellbook header
* Any of the database collections can then be opened from the choices on the left. The DB is configured to show the first 10 unfiltered results for each collection.

Report Functionality: This was added as a sort of ad-hoc part of the application to meet this requirement. Due to this, it isn’t super robust. It is available via the menu, once logged in, by clicking on the User Report link.

Exception Controls: Can be found throughout the application. The easiest to access and verify would be an invalid username or password when logging in.

Validation Functionality: This can be found in form areas throughout the application, namely: authentication forms, character create/edit forms, and custom spell form.

Security Features: Authentication was hand-coded using industry standards for creation, encryption, and storage of authentication tokens and authentication information.

Design Scalability: The following features were chosen to make the application scalable, with a description of how each one meets this requirement.

* Database – MongoDB Atlas was chosen because the database will be automatically scaled out, based on demand, and can easily be upgraded to a paid tier as the number of users increases.
* JavaScript Framework – SvelteKit was chosen to negate the need to have separate server and client-side frameworks. It was chosen over Next.js and Vue due to its scalability.
* Hosting platform – Netlify was used as the hosting platform for similar reasons as MongoDB was used for the Database. It is both automatically scaled and easy to convert to a paid model, based on usage.
* Filtering – Early on in testing, it was discovered that the spell list was taking a great deal of time to load and the cause was found to be filtering. At that point, the filters were rearchitected to have each filter only be ran if the filter’s default value had been changed. Additionally, all applied filters are evaluated for each item at the same time. The truthy statement for an item will stop evaluating as soon as it hits a false filter, to negate unnecessary processing for that item. This was done using filters and predicates in, what became, a very complex filtering mechanism that works efficiently.

User Friendly GUI: The user interface was created using intuitive, industry-standard design decisions. Additionally, much thought was given to layouts and screen sizes, with specific layout changes being present on various device sizes. While the scope of the application was designed for and tested on desktop and laptop computers, the interface should also look and function on both tablet and mobile screens. Note: The application was not tested on mobile or tablet, due to the countless screen sizes involved, and testing suites for screen sizes being cost prohibitive for anyone but a decent-sized company.

## Part C

Design Documents:

Note: the design documents are rather large and hard to read when inserted into word. For this reason, the files were not added to this document directly.

* Class Diagram: See ClassDiagram.png, in documentation folder.
* Design Diagram: See DesignDiagram.png, in documentation folder.

Unit Testing:

The unit tests, themselves, can be found in the project file in the folder /casterfire-api-test/src/\_\_tests\_\_ and include unit tests only for the filter components, as this is, by far, the most complex part of the application.

If you have set up the project to develop locally (see README.md in project root), you should be able to run the unit tests by entering the shell command “npm run test" from the project root. A screenshot from the time of submission can be found in the documentation folder, titled UnitTests-Screenshot.png.

Source Code and Executable:

Source code is available in /casterfire-api-test/src, with necessary config files in /casterfire-api-test, and static files contained in /casterfire-api-test/static.

Being a JavaScript web application, there is no executable, however the project can run locally in both dev and production modes, after the local development environment is setup (see README.md) these shell commands can be used to run the project.

* Dev: npm run dev
* Production: npm run build followed by, npm run preview

Web Application Link:

<https://casterfire.holtzen.rocks>

User Guides:

User guides for end-user and developer setup are included in the README.md file, located at the project root. Here are the important parts:

**Getting Started (End User)**

This web application was designed for use on a computer and, while some considerations were made for mobile, it may not function as intended on a phone-size screen, due to the nature of testing on multiple devices. I am but one man, with limited funds and device testing is expensive. The site should still function fine on mobile or tablet, but, depending on the screen size, some things could look a little wonky.

The initial release of the spell list only includes spells from the first two Pathfinder 2nd Edition books released by the publisher. The intent is to add more as the app gains popularity.

* Menu
  + Located in upper left corner.
  + Used to register, login, and logout.
  + Once logged in, character options are visible.
  + Contains a "Users Report" link to see all users and when they joined.
  + Character Options
    - Initially, the character list will be empty.
    - Click on the "New Character" button to create a new character.
    - Switch between characters, edit, or de-select characters using the menu.
* Spell list
  + Located along the left side of the application.
  + Clicking a spell opens it in the details pane, to the right.
  + By default, contains only the full list of spells in the Pathfinder books.
  + If logged in an option to create a new spell appears at the top of the list.
* Spell Details
  + The spell details screen is populated by selecting a spell from the spell list.
  + If logged in:
    - Option to copy a Pathfinder spell and edit it as a custom spell.
    - Option to edit, delete, or clone an existing custom spell.
    - If character is selected:
      * Spell will have option to add spell to selected character's spellbook.
* Filters
  + Perform live filtering on the spell list.
  + By default, shows all Pathfinder and Custom spells available.
  + Sources Filter
    - Rulebook and Custom buttons are used to hide or show Pathfinder spells or custom spells, respectively.
    - If logged in and a character is selected, Sources will contain a button for "Only [Character Name]'s".
      * To view only spells that have been added to the selected character (that character's spell book), select "Only [CharacterName]'s".
      * Note: you can still filter a character's spells by Rulebook and Custom. In order to see a character's full spell book, with both Rulebook and Custom spells, both of these options would need to remain checked.
  + Actions Filter
    - Used to filter spells that contain one or more of the checked action costs. By default, does not perform filtering.
  + Components Filter
    - Contains both Cast Times (if longer than 1 round) and Cast Requirements.
  + Spell Level Filter
    - Allows filtering by the level of the spell (less than or equal, greater than or equal, and equal.
  + Traditions Filter
    - Allows picking a specific magic tradition (also known as a spell list).
    - Note: if an option is selected, a spell will only show up if it is on all of the selected tradition lists.
  + Traits Filter
    - This filter is rather large, but will filter by specific spell traits.
    - Organized by trait category.
    - Note: Spells will only show up if they contain all of the selected traits. This may mean that the filter yields zero results (for example, if you selected Darkness and Light).
  + Search Filter
    - Filters spell list, searching both spell name and spell description for the typed text.

**Local environment (dev) Setup**

1. Download and install the [Node 16.9.1](https://nodejs.org/dist/v16.9.1/) installer for your operating system.
2. Download or clone this Git Repo.
3. Place .env file in casterfire-api-test/src/ folder of the project. This will be provided by project maintainer.
4. Open a command or bash prompt at the project root folder (casterfire-api-test/), and run npm install.
5. Once dependencies are installed, run the command npm run dev -- --open. This should open the dev version of the site in your default browser.