# Web App Planning

## Pages & Objects Overview

- Homepage/Landing Page
- Character Pages
- The Gunroom
- Reading/Story Pages

### Homepage/Landing Page

Shows Landing Page Image and text, maybe as if it is the front cover of the book and then leads to the start of the first book. Grab content from the DB straight away. Will most likely require a short loading screen depending on connection. Assuming the first page will be consistent across all first time visitors we could hard code the first chapter into a JSON file stored in the server with the app code to load this immediately and lazy-load the rest as usual.

Upon return visit, if a user is logged in or using the same device as last time (with the cookie on it) the page will open up directly where they left off.

See **Reading/Story Pages** Below

### Character Pages

Full Index of characters with information including images, bios, books they appear in (Clickable - taking you to the book, or your progress in it)

**Character Objects:**

Properties:

Books they belong to

Bio (Age, Gender, Back story?)

Resource Keys (UID store of related resources)

DB Relationships:

Has many books

Has many resources

### The Gunroom

A full archive of resources, imagery and other interesting things related to each of the books

(Each organised by book.)
Will most likely need Lightbox functionality for viewing images.
Will need upload functionality, can use a drag'n'drop uploader or native browser uploader and store in database, references saved to the db object. Use a promise so that all info is entered, the upload is dropped in (in any order) then on submit the app waits for the file to be written to the database and .then() => the resource object is stored in the resources object along with the file reference.

In terms of expandability, more types (see below) could be added on the fly to add new categories of resources at a later date.

Types
Images
Maps
Source Materials

**Type Objects**
Due to the nature of these objects they will most likely just be references to files stored in the DB stored alongside their unique ID and tied together in the UI by their DB Relationships.

DB Relationships
Has one book
Has many characters

**Reading/Story Pages**
Pages that are purely book content. As stripped back and immersive as possible.

**Features:**

- Sidebar
  - Progress Indication (I.e. page number, chapter number)
  - Bookmark button (Will force save progress to current position, but also make a reference to the current position so a user could return at a later date regardless. Possibly allowing multile bookmarks, as appose to one per book, that allows users to jump to sections they have saved. This could also be split into two separate features, bookmarking for current position and 'post-it' or 'marker' or something similar for saving certain sections for later reference.)
  - Share button that opens modal to send (potentially) pre-written social post (i.e. "Im reading X. Check it out!")
  - Could stay visible on mobile but be sticky to the bottom of the page to account for the

lost ‘screen-real-estate’ (Possibly even only appear on scroll up, and re-hide itself on scroll down etc.)

- Lazy-Loading (See Below)
- Sticky Header (See Below)

## Site Wide Functionality Overview

- Sticky-Header
- Admin Backend
- Lazy-Loading of content after scrolling within X amount of pixels of chapter end
- Sign In Functionality (for extra features such as saving progress across multiple devices)
- Search

### Sticky Header

Fairly straightforward, content slides under header instead of header scrolling with the content. On mobile this would most-likely become an off-canvas menu, however we can add a tiered menu with dropdowns showing the submenus of chapters, characters etc.

### Admin Backend

I will create a bespoke admin backend to accomodate for the specificity of certain database items, i.e. Adding some form of pseudo tag that will split the content into pages (see **Books** below) and the addition of characters and their custom meta. This will require a WYSIWYG text editor that speaks to the database and will effectively mirror the Wordpress back-end i.e. periodically saving drafts and allowing fields for adding the necessary meta. It’s important to remember to have strong form validation on these meta fields where necessary, i.e. data formatting so as to keep everything in the database uni-formed.

Thinking along the lines of an ‘Add Book’ button which will create a new empty object in the database. The UI will then have, effectively, a list of books, when clicked on shows all the chapters, characters & resources related to that particular book (These could be added to any book in theory, for example, if a resource belonged to the new book, but also a book that already existed in the database.)

### Lazy-Loading

Again, fairly straight-forward, the content of the next page will load in from the database with X pixels of the bottom of the current page giving a seamless reading experience. Depending on the payload size we could grab two/three at a time and store locally to cover the user

should their connection dip.

### Search

Due to all the meta information saved for each item, users will be able to quickly filter through the database to find what they need. There are a few options, all are achievable and maybe a combination of them would be best. For example:

- Site wide search that filters through entire list of data and results are categorised by resource, book, character etc.
- Page specific search, search is relative to the page your on, resource page searched resources etc.
- Text Search, When reading, maybe a search button would be nice to allow users to search the text similar to 'ctrl(cmd) + F' when in the browser or a word doc etc.

## Dev Functionality  Overview

- Webpack Code-Splitting to speed up builds and lazy-load chunks of code as they are needed (Separate bundle for front-end and admin)
- PWA Features
  - Add-to-homescreen
  - Book Caching for offline reading
  - **Will Require An 'App-Shell'**

### PWA Features

To implement the PWA features, the best bet is to first of all complete development and initial testing to make sure all the data is flowing well and the app is 100% working efficiently. This also helps give a better idea of exactly what needs caching and when as well as what types of push notifications might be useful (if at all, however, I'm thinking along the lines of 'We just added a new book' etc.)  Then I need to add two things:

- Web App Manifest
  - A JSON file outlining the details & options for the app and how it should behave when installed to a users mobile/tablet homescreen
- Service Worker
  - Responsible for the bulk of the features outlined above. Effectively a list of functions containing promises and switch statements that react to certain user input, or specific conditions by caching content and sending notifications. This allows the app to work offline, with this I think a good initial implementation would be to give users the option to cache particular books, or we can automatically cache a book ahead of the user this would reduce the amount of lazy-loading required and would mean offline was just a standard feature, without the user specifically saving books. However realistically, this

may mean constantly adding a new page to the cache and deleting the last page off as they swipe through the content so tests will need to be done on the best way to achieve this.

## Purely Database Items Overview

- Books
- Users

### Books

Chapters of books will be stored in the database and referenced in each books DB object to tie them together. This way they can be loaded as objects from database 'lazily' as the user scrolls.

To keep track of page count and sync with the physical edition of the book, each page should be its own child in the chapter object. This will make it much easier to get page counts for total books and chapters and helps a lot with progress indication.

DB Object example:

```
book: {
    chapter: {
        page: "",
        page: "",
        page: ""
    },
    chapter: {},
    chapter: {},
    etc… (Other properties too, see below)
}
```

Presumably the best way to split the paragraphs into pages will be a proprietary tag/atribute and teach the database upload function to split the strings at these specific points, that way the writing experience should be fairly seamless as appose to pushing the data up page by page.

Properties

Characters

Author

Date Released/Published

Page count
Chapter Count
Chapters (DB Objects containing references to the content for each book, divided into chapters)
Front Cover/Artwork (Link to image reference)

DB Relationships
Has many Characters
Has one genre

Users
If a user signs up for extra features such as cross device progress saving (possibly book favouriting or even cross device caching - will need to double check this is possible) this will be an object in the database that stores their personal data i.e. name, image, last sign in date and objects containing their progress for each book

DB Relationships
Has many progressPoints

- These would most likely be the UID's of the respective book stored as the key and the value being the point in the book they are up to.

Has many favourites

- Could have sub-objects dividing the favourites into book, characters, resources and storing UIDs of each.