Table of Contents

[User stories for project 3 – Marvel Reading List 2](#_Toc2440024)

[Epic: 2](#_Toc2440025)

[End User Stories: 2](#_Toc2440026)

[Development Stories: 2](#_Toc2440027)

[Features left to implement: 4](#_Toc2440028)

[Mockups: 5](#_Toc2440029)

[Bugs encountered throughout 13](#_Toc2440030)

[Testing 14](#_Toc2440031)

[Heroku scheduler screenshots: 15](#_Toc2440032)

[Unit tests 16](#_Toc2440033)

[Screenshot: 16](#_Toc2440034)

[Text output unit tests (nose2): 17](#_Toc2440035)

[Database information 18](#_Toc2440036)

[Collection ‘users’: 18](#_Toc2440037)

[Collection ‘comics’: 18](#_Toc2440038)

[Collection ‘user\_comic\_list’: 19](#_Toc2440039)

[Collection ‘admin\_base’: 19](#_Toc2440040)

[Rationale for this flat approach: 20](#_Toc2440041)

# User stories for project 3 – Marvel Reading List

#### User stories, development stories based on them, and a summary of features left to implement to enhance the experience overall. MVP has been reached (all development stories are checked off).

## Epic:

As an end user, I want to be able to maintain an archive of read and unread comics

### End User Stories:

1. As an end user, I want my list to be private by having it behind a login
   1. I want to be able to log into my account
   2. I want to be able to log out of my account
2. As an end user, I want to see newest comics and add them to my to-do list
   1. I would like the list to be refreshed whenever new comics come out
   2. I want an overall list of comics to go through at a later date
3. As an end user, if a comic is in my list (unread or read), I want it to be hidden from the list of comics available for adding to the list
   1. As an end user, I want to be able to remove comics from my archive of read comics
   2. As an end user, I want to be able add comics again once I removed them
4. As an end user, I want to be able to change my password
5. As an end user, I want to see detailed images of comic covers
6. As an end user, I want to be able to maintain my list on a mobile device
7. As an end user, I don’t want to have to input text as much as possible

## Development Stories:

1. User Story 1: **√**
   1. Create a welcome page, where users can login, or sign up
   2. Provide some basic explanation and description
   3. Create user registration page
      1. Allow user to go back to main page or sign in page
      2. Create logic to ensure usernames are unique
      3. Create logic for allowed characters in user names
      4. Create logic for passwords
      5. Create logic for allowed characters in passwords
      6. Create logic to let user view entered password(s) before submission
   4. Create user login page
      1. Allow user to go back to main page or registration page
      2. Ensure messages show mistakes made by the user
      3. Make sure it isn’t obvious which field was wrong, so username and password guessing gets harder.
   5. Create user homepage
      1. Require logged in user for the page
         1. Return 401 page on tries to open page without login
2. User Story 2: **√**
   1. Personalise the navbar for the user **√**
      1. Show username in the navbar
   2. Include buttons in the navbar to current list or adding comics **√**
      1. Two main buttons, one for the list now, one for adding comics
   3. Create a logout button for the user **√**
   4. Make sure the user logs in to their current list of unread comics **√**
      1. Show a message when their list is empty with link to add comics
      2. Make links to current page disabled to support visibility of location
      3. Give links to unread and read comics
   5. Make sure navbar link to add comics shows latest comics by default **√**
      1. Make links similar to read/unread comics for the new/all releases
      2. Show the timespan for the list of new releases
      3. On new releases, show the released date
      4. Add a ‘top of page’ link to avoid users having to scroll all the way back up
3. User story 3: **√**
   1. On loading of page, find the user’s current comic list **√**
      1. When adding, match current list with comics to add and exclude the view from the comics in the list 🡪 allows comics to show for addition again after delete from list
   2. If a comic is in the ‘add comics’ section, show it with a button to add to the reading list **√**
   3. If a comic is in the ‘my comics’ section, and unread, show it with a button to mark is as read **√**
   4. If a comic is in the ‘my comics’ section, and read, show it with a button to delete it from the list **√**
4. User Story 4: **√**
   1. Add a settings button to the navbar for logged in users **√**
   2. Create page for user to reset password **√**
      1. Require current password, and new password (twice)
      2. Create logic for checking the new password
   3. Bonus: Create option for user to change displayed name **√**
      1. Update the navbar to show the displayname once name change is done
   4. Show visual response on error, or success of changes **√**
   5. Allow user to view entered passwords prior to submission **√**
5. User Story 5: **√**
   1. Make thumbnail images of clickable links for detail image **√**
   2. Use Lightbox to pop up a detailed image fitting the screen used **√**
6. User Story 6: **√**
   1. Add in scalability **√**
      1. Barring portrait mode, scale to mobile landscape mode
      2. Add message to index page stating landscape on mobile advised
7. User Story 7: **√**
   1. Make sure clicks and taps are needed whenever possible, aside from users logging in or registering, or changing display name and/or password **√**

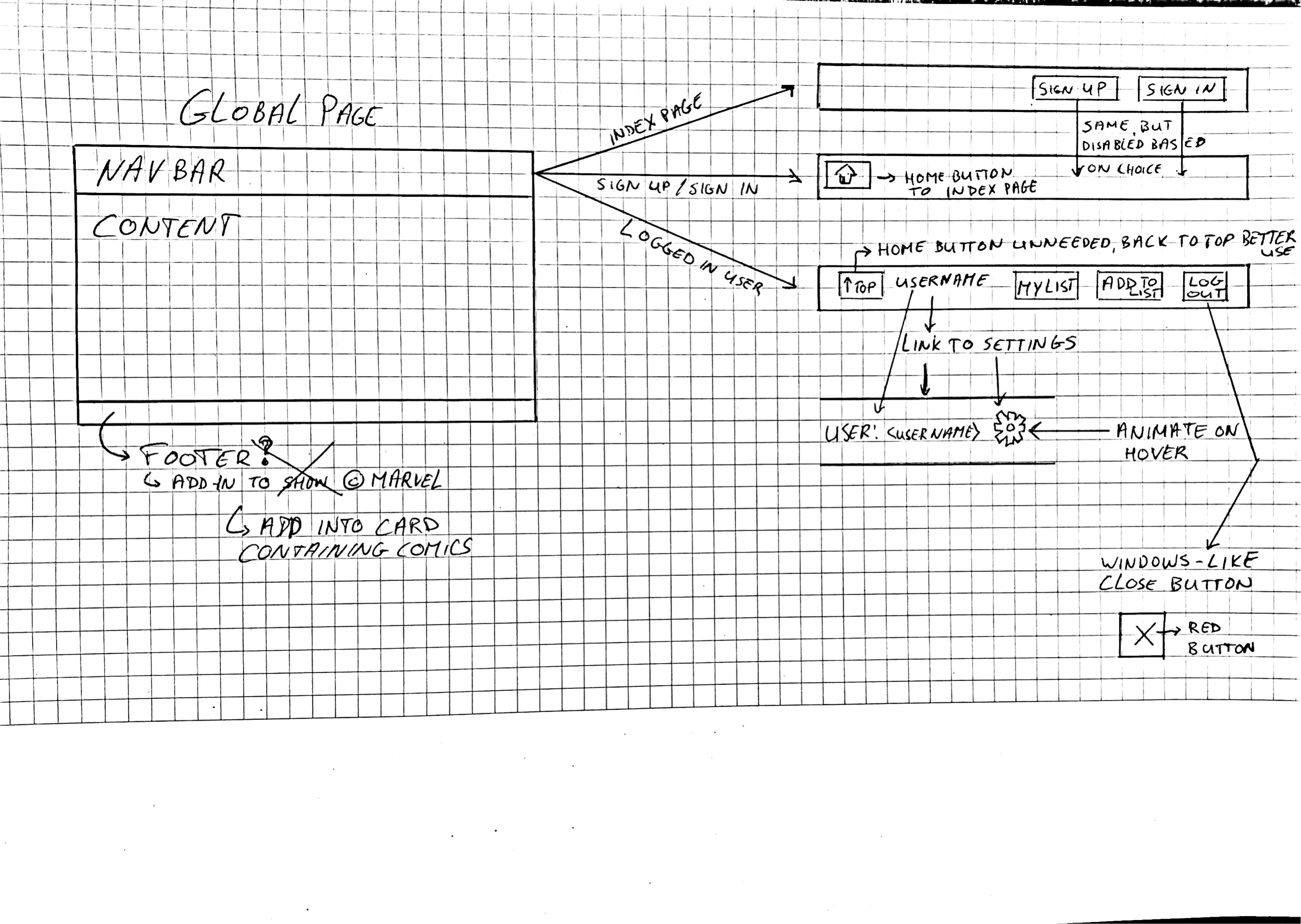
## Features left to implement:

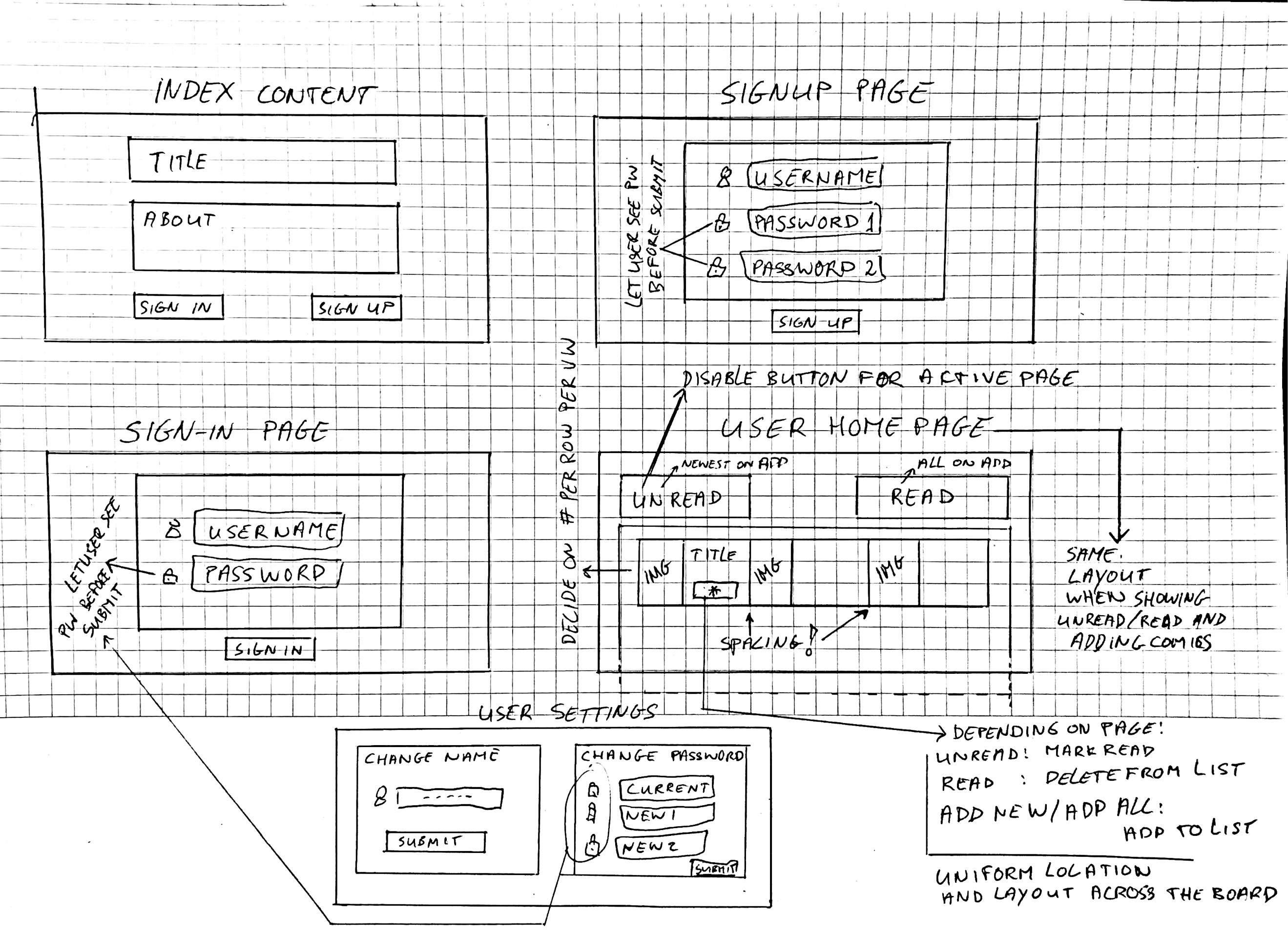
1. Allow the user to zoom in on a comic to retrieve more information on:
   1. Series the comic belongs to:
      1. Characters
      2. Other comics in the series
      3. Events
   2. Information on how to obtain the comic
2. Create a filter function so users can filter the results based on what they look for
   1. Preferably autocomplete
   2. Preferably pre-filled dropdown with series titles
3. Expand on user settings
   1. Allow users to download a csv file with their list items
   2. Add email into registration to allow users to have a password reset link sent to them
4. Expand on admin page
   1. Create statistical view for user creation
   2. Add last\_logon to users and make it visible

## Mockups:

Below images were global thoughts on how it should look.

Underneath are screenshots of the end result. The only missing mockup is the admin page created, which was developed on the fly.

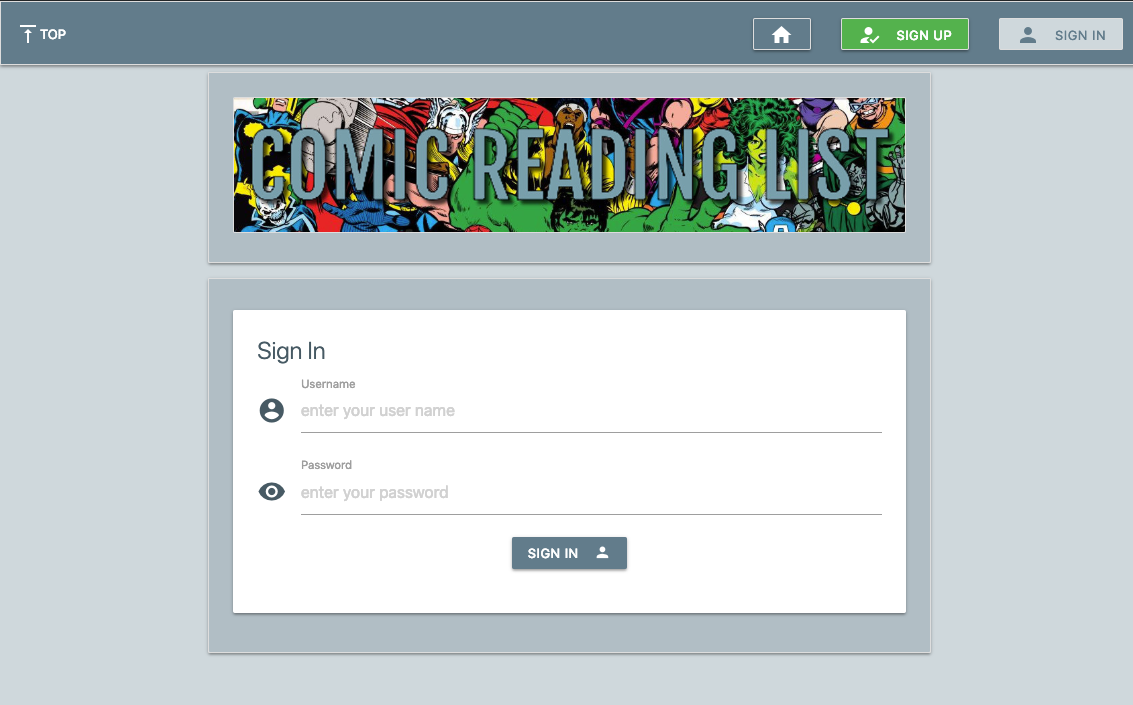




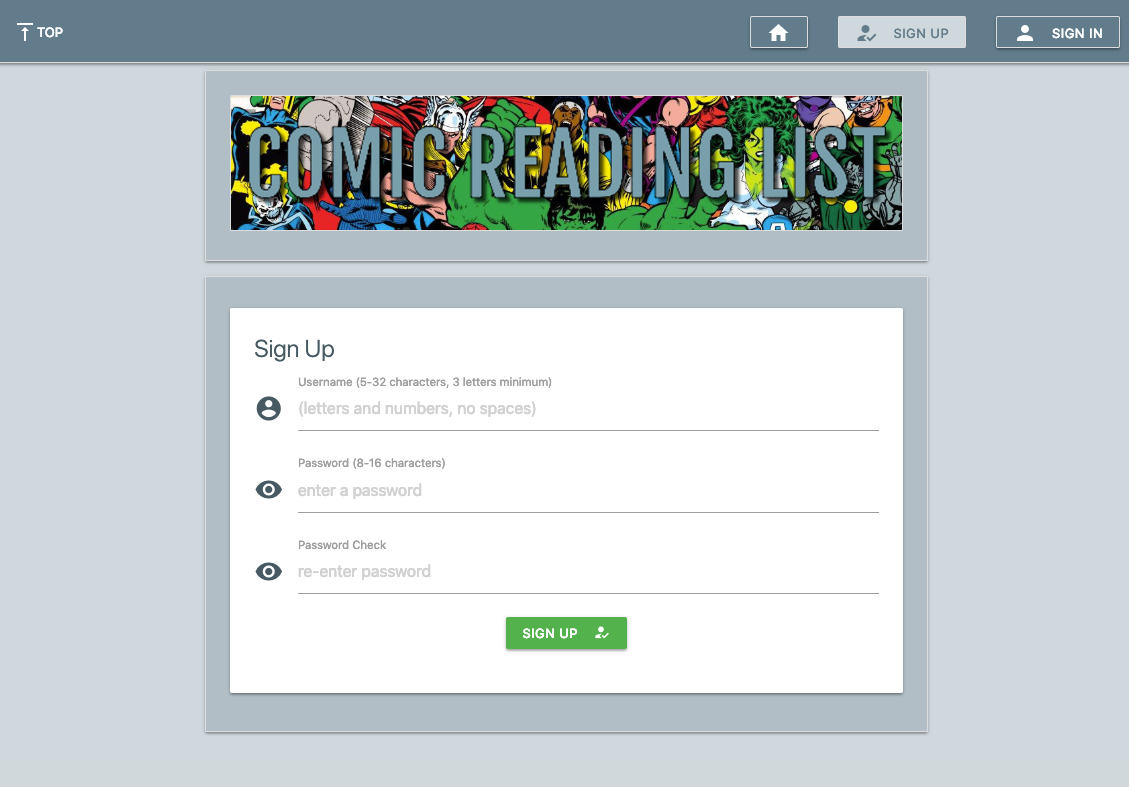
Index Page



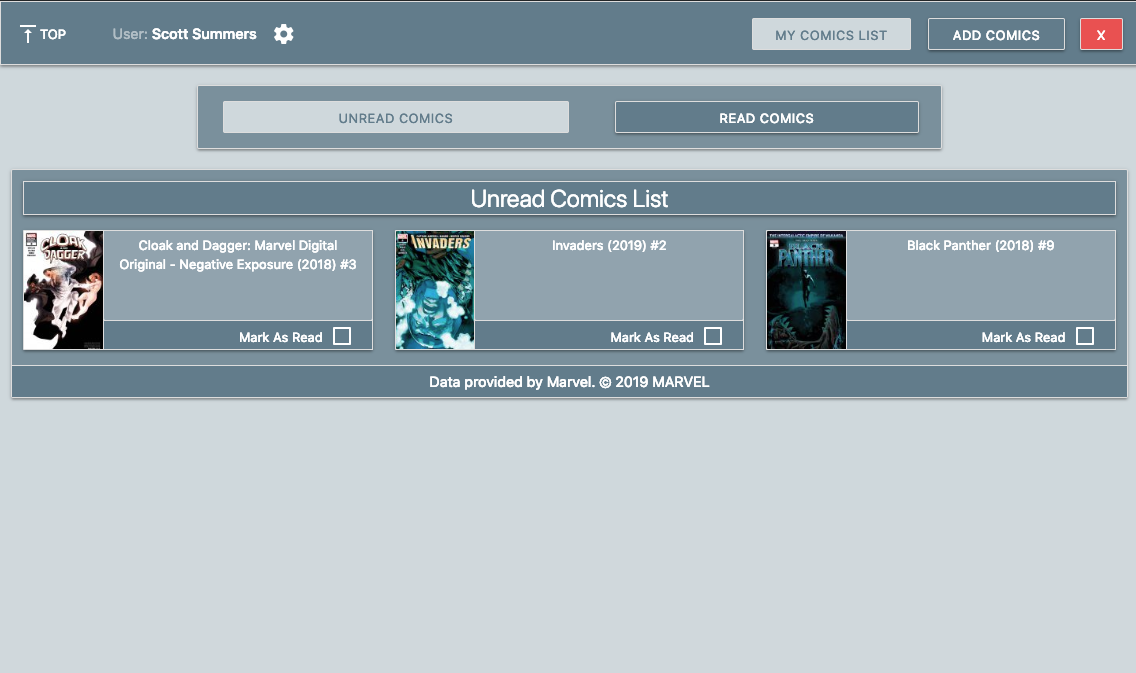
Sign In Page

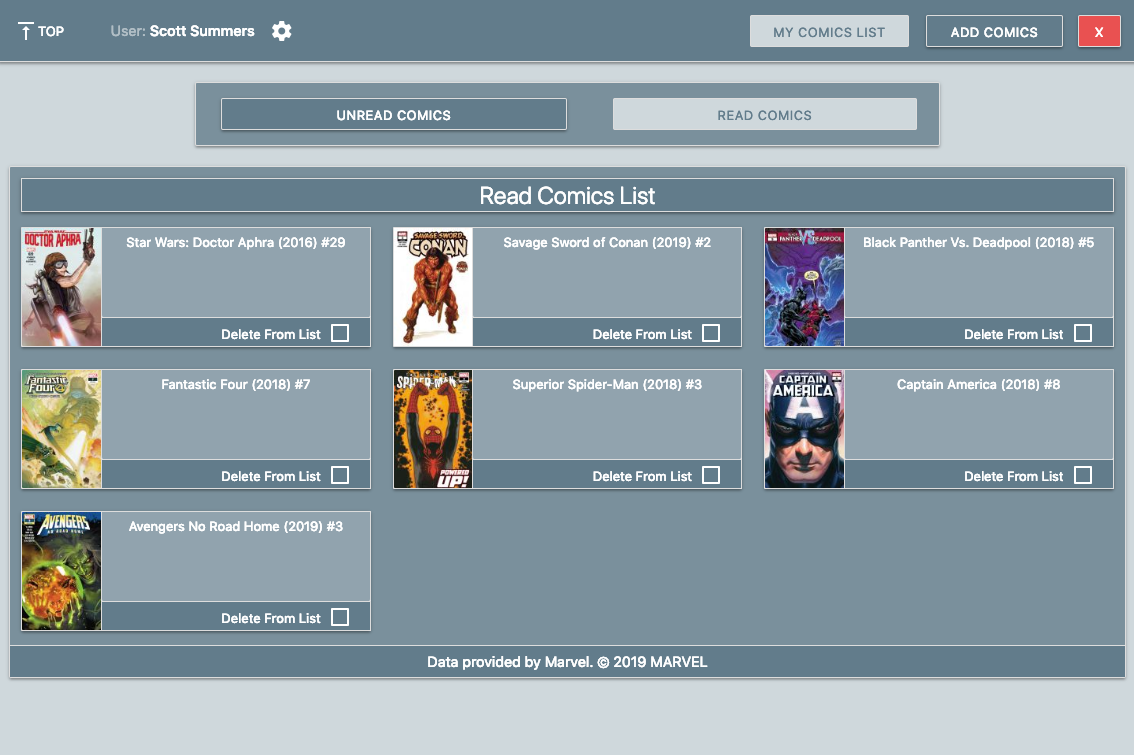


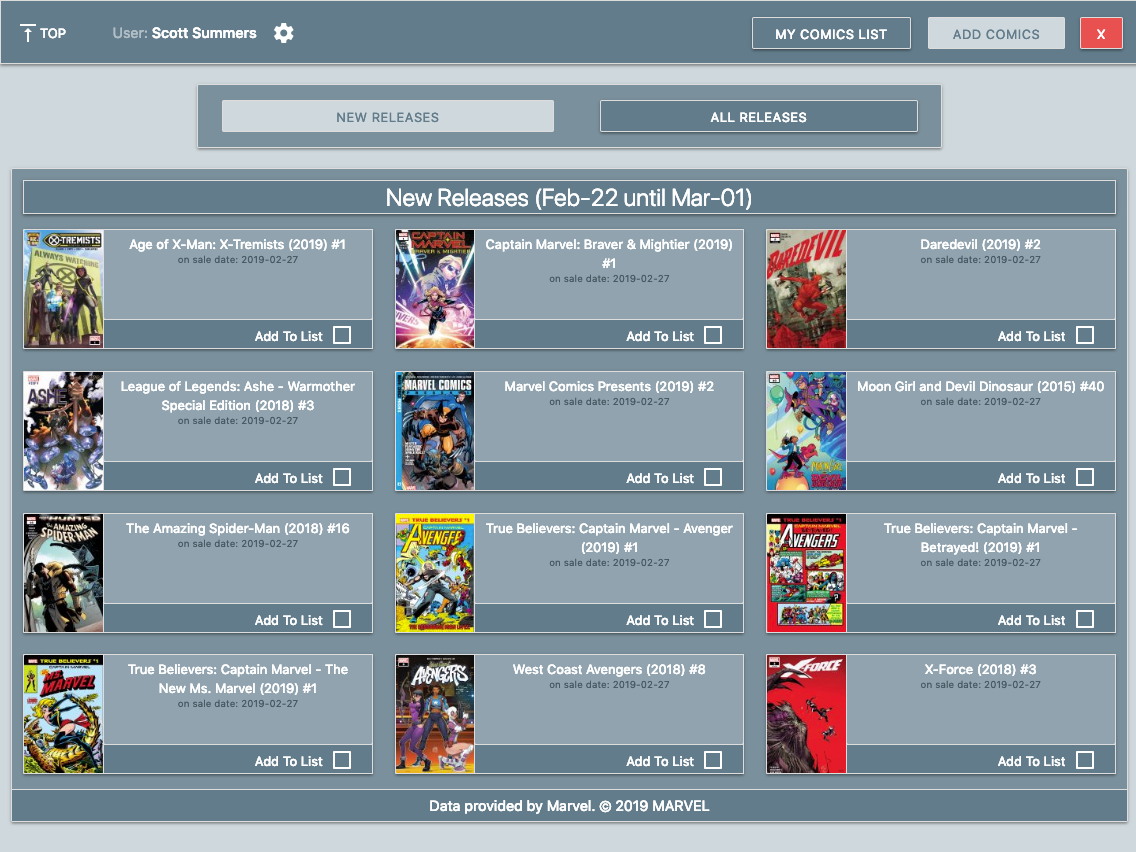
Sign Up Page



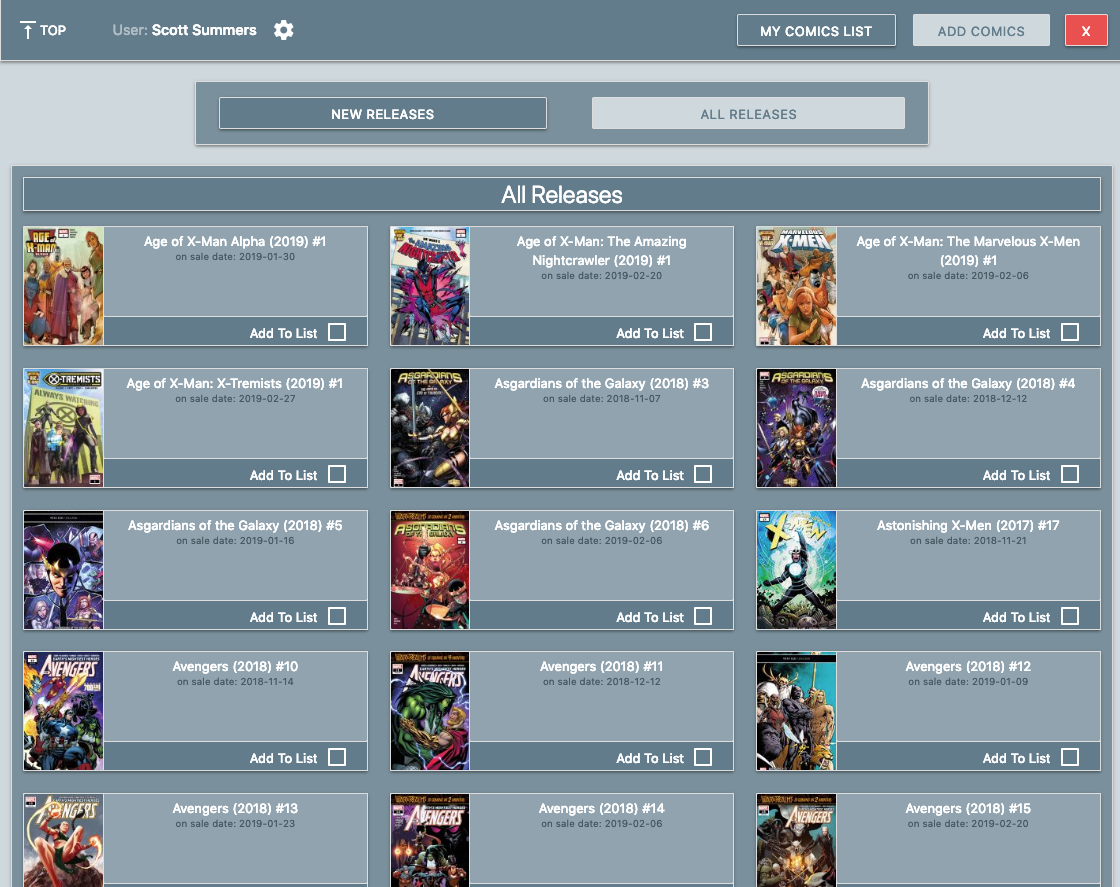
User Home Page (defaults to Unread Comics list)



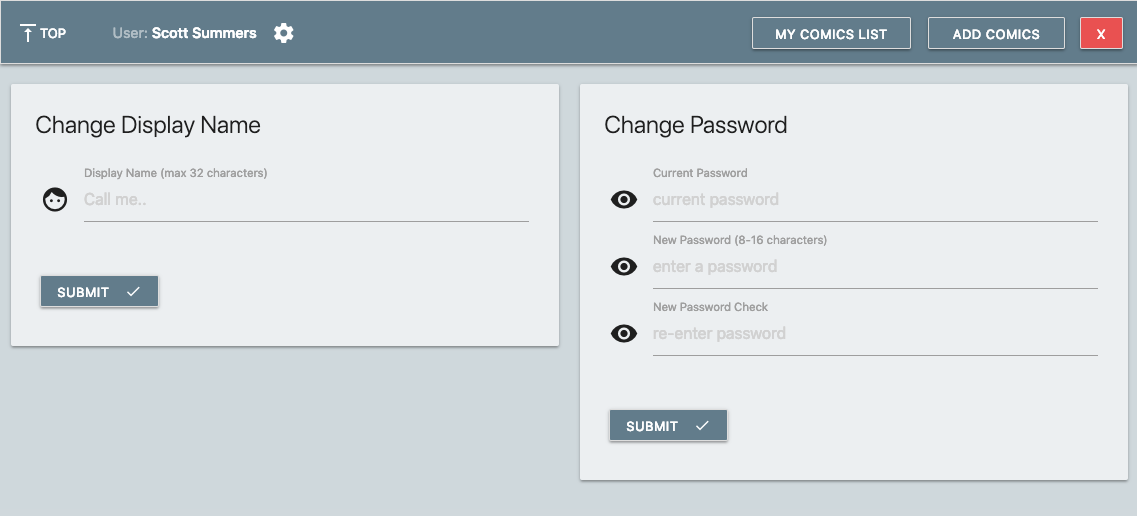
User Read Comics List

Add Comics Page (defaults to New Comics)

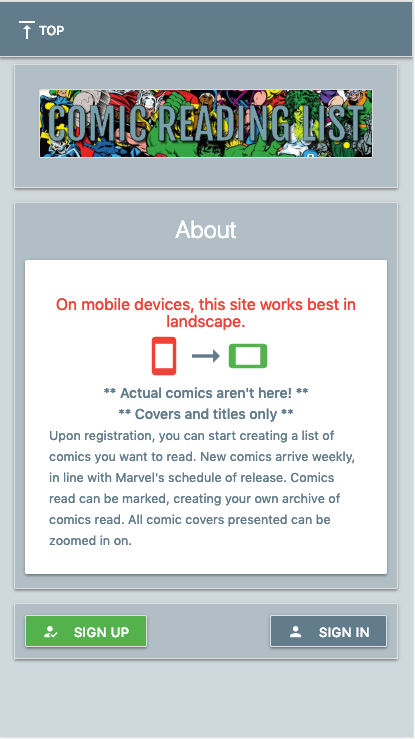
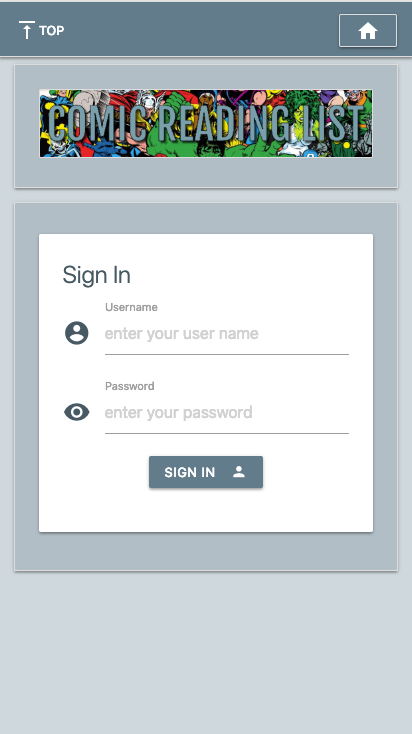
Add Comics (all) Page

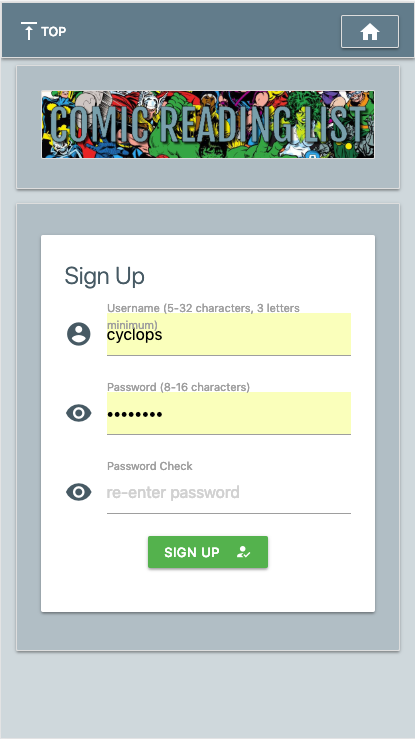


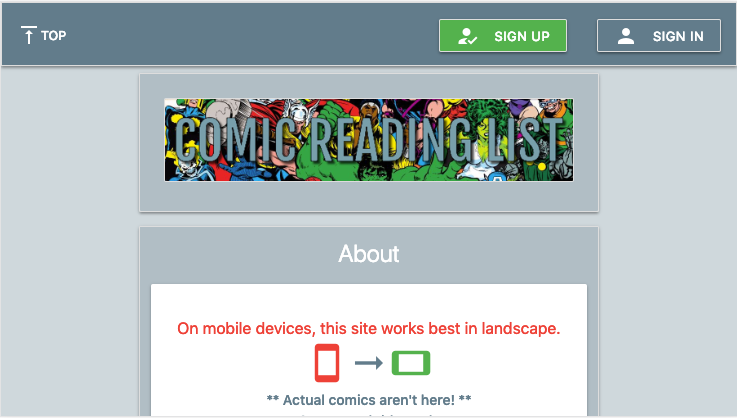
User Settings Page

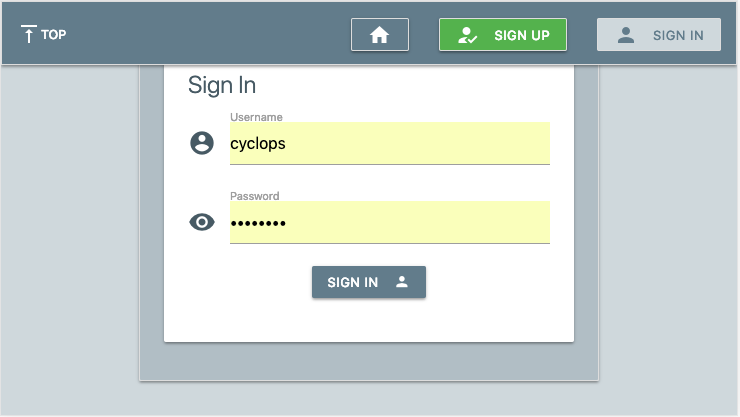


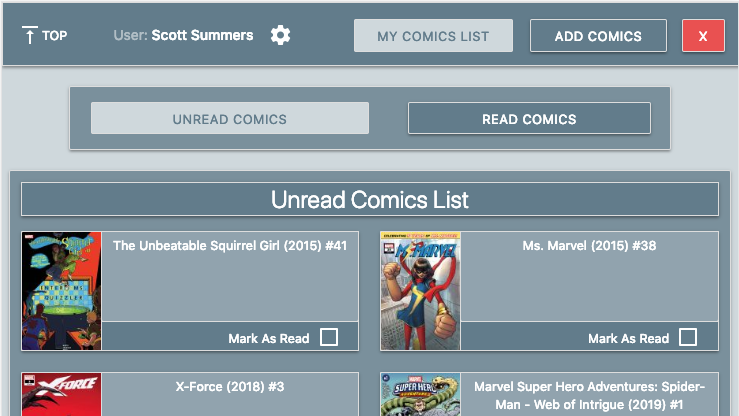
Mobile images (simulated with Chrome developer tools)

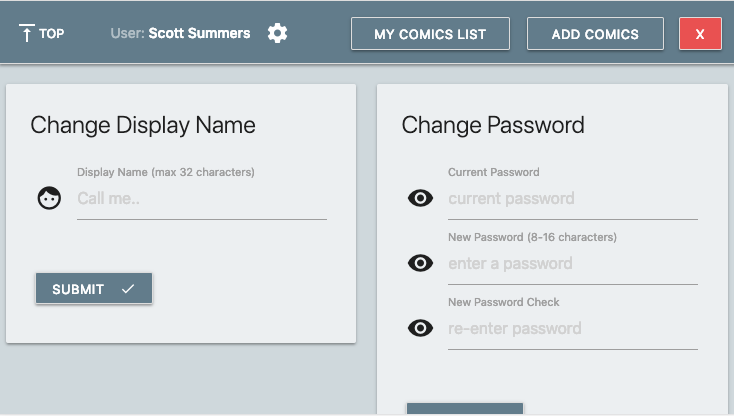
 









Admin Page:



The Admin page was developed on the fly and was made to alleviate database actions on the mLabs site.

Here, the admin user can see all users, and choose to delete them if needed.

## Bugs encountered throughout

Throughout, small bugs were encountered while creating the site.

Several were humorous enough, to the point of facepalming oneself.

|  |  |
| --- | --- |
| Add animation to user settings cog on the wrong level | Have all links within the site spin on hover. Unwanted but funny nonetheless.  Quickly fixed by adding a custom class to CSS and applying the animation to that. |
| Create script to get comics without checking for existing comics in the database | Exploded the database quickly when testing the script.  Solved by having the script connect to the database, retrieve all comics that were in the list, then match those against the API call results from Marvel.  Matchmaking is done based on the comic ids served by Marvel as they are also stored in the database. |
| Add a check whenever a logged in user goes to the login page or signup page when logged in to reroute them to the user home page | The check worked well while logged in as a user. However, logging out would not let the index redirect to the sign in/up pages.  Instead of performing the check ‘if session[‘user’]’, it needed to be ‘if ‘user’ in session’.  Small mistake with very unfortunate consequence. |
| Switch the Heroku deployment to use gunicorn and encounter the same problems as above | Not solved yet, couldn’t for the life of me get it going even without checking running sessions.  Remains in the todo list for now. |
| Despite setting link to disabled, Lightbox still tried to open a pop-up for images after marking them. This would then be an everlasting wait state | Solved by removing all Lightbox attributes from the link when marking comics, effectively killing the pop-up option. When the page is reloaded, the comic has migrated to whatever state it’s in and everything works as should again. |

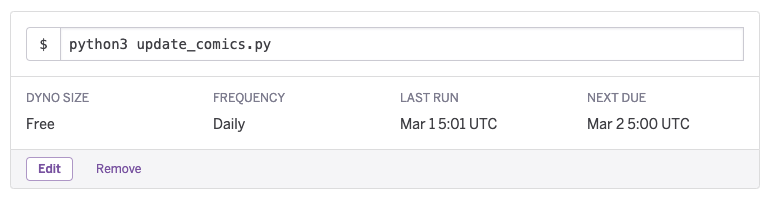
Other than the above, bugs encountered usually revolved around incorrect indentation, or around modifications to HTML leading to unclosed tags screwing the layout up entirely.

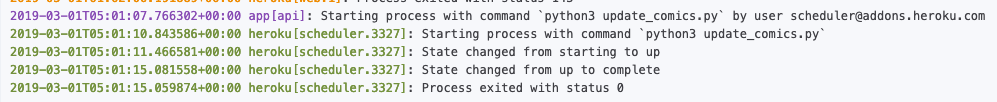
None worth mentioning aside from the above.

## Testing

|  |  |
| --- | --- |
| Initial Page | * On first installation, a base.html was made, followed by a simple index page. |
| Add mongo to the site | * User registration and user login basics tested. * User stored in mongo, test login functionality. * Check password hashing for ‘saltiness’ by creating multiple users with the same password and comparing the hashes. * Create a new page and pull data from it in a crude list like manner to test connection. |
| Create and test update\_comics script | * Test addition to the mongo database repeatedly by grabbing a subset of results in the script and adding those to the database. * Perform multiple runs to test avoidance of double adding comics to the total list * Have the script print out results of mongo and Marvel pulls * Modify script to use the same config file as the main app and test functionality again |
| Expand user creation logic to test for uniqueness and minimal requirements | * Create a user with less than 5 characters and get an error * Create a user with less than three letters in the name and get an error * Create a username with more than 32 characters and receive an error * Create a username with disallowed characters and receive an error * Create a duplicate username and receive an error * Try creating empty user and give easter-egg error * Create a password less than 8 characters and receive an error * Create a password longer than 16 characters and receive an error * Try empty password and give easter-egg error * Try mismatching passwords and receive error * Check message returned on any of the above * When correct, check addition into database * When correct, check correct re-routing to user home page |
| User home has no current list of comics | * Verify showing of Drax message ‘nothing here’ |
| User read comics list has no current entries | * Verify showing of Drax message ‘nothing here’ |
| Manipulate the address bar to make a fake endpoint page | * Confirm non-existence by 404 page showing Iron Army drone |
| Try to enter a page without logging in | * Confirm 401 error with Black Widow image * Confirm for user home, user settings, admin page, add comics, read comics, add all comics |
| Try to manipulate the bar to perform POST actions | * Verify 405 error showing Deadpool for disallowed methods |
| Mark a comic for addition to the unread list | * Verify the comic is added to the list |
| Mark a comic as read | * Verify the comic is added to the read comics list |
| Delete a comic from the read comics list | * Verify its re-addition to available comics |
| Add site to Heroku and change development database | * Verify that on Heroku the database is different from development through available comics and users available * Create a user in development, verify the user does not exist on Heroku and vice versa |
| Delete a user | * Verify deletion of the user by trying to log on with the account credentials and receive errors |
| Re-create the deleted user | * Verify that the user has no longer a list to their account |
| Pick a user from the admin page and note their unread and read items | * Log in as the picked user and verify that the numbers for read and unread comics are correct with what was shown on admin page |
| Have the user add and remove items from the list | * Verify in admin page that numbers have changed to match |
| Add Heroku scheduler to automate the updating of comics on Heroku | * Set schedule for hourly and check if script is run exiting with an error code 0 * Set schedule for daily |
| Every week, check new additions to the new comics list | * Slow check, verifiable by the onsaledate visible in new comics * Comics in new are never older than a maximum of 8 days |

## Heroku scheduler screenshots:



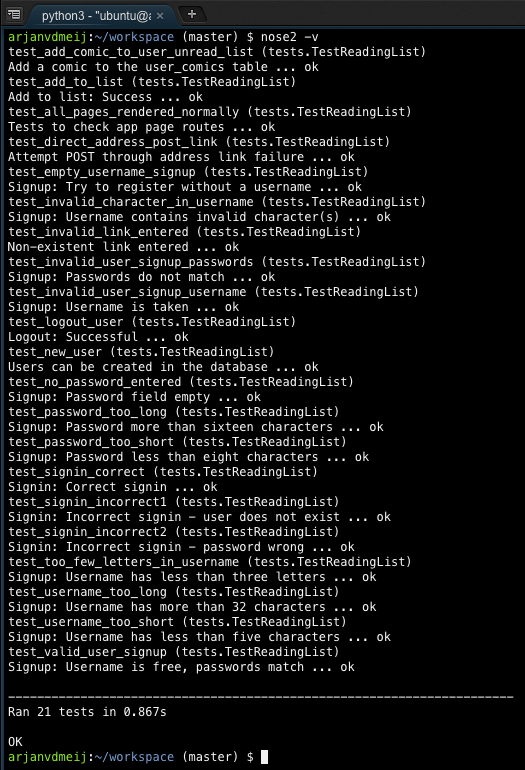


## Unit tests

Unit tests were created to perform some automated tests alongside the visual testing by hand.

Nose2 was used as a more visual response when executing tests. Below are both a screenshot and the textual output from that screenshot highlighted for execution:

### Screenshot:



### Text output unit tests (nose2):

|  |
| --- |
| **arjanvdmeij**:~/workspace (master) $ nose2 -v  test\_add\_comic\_to\_user\_unread\_list (tests.TestReadingList)  Add a comic to the user\_comics table ... ok  test\_add\_to\_list (tests.TestReadingList)  Add to list: Success ... ok  test\_all\_pages\_rendered\_normally (tests.TestReadingList)  Tests to check app page routes ... ok  test\_direct\_address\_post\_link (tests.TestReadingList)  Attempt POST through address link failure ... ok  test\_empty\_username\_signup (tests.TestReadingList)  Signup: Try to register without a username ... ok  test\_invalid\_character\_in\_username (tests.TestReadingList)  Signup: Username contains invalid character(s) ... ok  test\_invalid\_link\_entered (tests.TestReadingList)  Non-existent link entered ... ok  test\_invalid\_user\_signup\_passwords (tests.TestReadingList)  Signup: Passwords do not match ... ok  test\_invalid\_user\_signup\_username (tests.TestReadingList)  Signup: Username is taken ... ok  test\_logout\_user (tests.TestReadingList)  Logout: Successful ... ok  test\_new\_user (tests.TestReadingList)  Users can be created in the database ... ok  test\_no\_password\_entered (tests.TestReadingList)  Signup: Password field empty ... ok  test\_password\_too\_long (tests.TestReadingList)  Signup: Password more than sixteen characters ... ok  test\_password\_too\_short (tests.TestReadingList)  Signup: Password less than eight characters ... ok  test\_signin\_correct (tests.TestReadingList)  Signin: Correct signin ... ok  test\_signin\_incorrect1 (tests.TestReadingList)  Signin: Incorrect signin - user does not exist ... ok  test\_signin\_incorrect2 (tests.TestReadingList)  Signin: Incorrect signin - password wrong ... ok  test\_too\_few\_letters\_in\_username (tests.TestReadingList)  Signup: Username has less than three letters ... ok  test\_username\_too\_long (tests.TestReadingList)  Signup: Username has more than 32 characters ... ok  test\_username\_too\_short (tests.TestReadingList)  Signup: Username has less than five characters ... ok  test\_valid\_user\_signup (tests.TestReadingList)  Signup: Username is free, passwords match ... ok  ----------------------------------------------------------------------  Ran 21 tests in 0.867s  OK  **arjanvdmeij**:~/workspace (master) $ |

## Database information

Note: while different databases, development and production have the exact same composition of data in the collections.

As described in the README.md, there are 4 collections to the database:

* users
* comics
* user\_comic\_list
* admin\_base

Below examples of documents in each collection are given, along with reasoning:

### Collection ‘users’:

Example:

|  |
| --- |
| {  "\_id": {  "$oid": "5c6539e0c8c5d9000431a4b1"  },  "user\_name": "arjan",  "pwd": "pbkdf2:sha256:50000$lPK0qAQO$786163a7d205eddafcd994fa6e0f31ba024128e675a6e3397833bd332ec3418f",  "creation\_date": "20190214",  "display\_name": "Arjan van der Meij"  } |

A user will initially be created with a display\_name equal to the user\_name. The user can later decide to change their displayname.

As can be seen, passwords are stored hashed. This is an added bit of security to avoid plaintext passwords being available.

The creation\_date is nothing more than an administrative addition at this point, to be used in time within the admin page when it gets expanded on.

### Collection ‘comics’:

Example:

|  |
| --- |
| {  "\_id": {  "$oid": "5c5da0e8f417770777f5d15f"  },  "comic\_image\_fs": "https://i.annihil.us/u/prod/marvel/i/mg/4/e0/5c379577c958f/detail.jpg",  "comic\_series\_id": "26899",  "comic\_title": "Invaders (2019) #1",  "comic\_id": 74763,  "comic\_image": "https://i.annihil.us/u/prod/marvel/i/mg/4/e0/5c379577c958f/portrait\_large.jpg",  "on\_sale\_date": "2019-01-16"  } |

Comics are added with needed information, along with a future-use series ID, which can be used to get more information on a comics series, as well as filter for a specific series.

Two links to images are added, one for detail and one for thumbnail, to be used by Lightbox to show zoomed in covers of comics.

### Collection ‘user\_comic\_list’:

Example:

|  |
| --- |
| {  "\_id": {  "$oid": "5bf68440f4177702eb49740d"  },  "comic\_series\_id": "26485",  "comic\_id": 73485,  "user\_list\_identifier": {  "$oid": "5bf6840bf4177702eb497408"  },  "comic\_image\_fs": "https://i.annihil.us/u/prod/marvel/i/mg/a/10/5beca8756966a/detail.jpg",  "comic\_status": "unread",  "on\_sale\_date": "2018-11-21",  "comic\_title": "Spider-Man: Enter the Spider-Verse (2018) #1",  "user\_name": "wolverine",  "comic\_image": "https://i.annihil.us/u/prod/marvel/i/mg/a/10/5beca8756966a/portrait\_large.jpg"  } |

Whenever a user adds a comic to their list, a document is added to the collection, containing the information needed to show the comic in the list.

Rather than cross-referencing based on comic\_id, I decided to clone the comic entry into the document for easier retrieval of data.

Next to the comic data itself, three fields are added:

1. user\_list\_identifier  
   used as a secondary unique user identifier. It contains the \_id of the user document
2. user\_name  
   used to obtain all documents that make up the selected list for the user, in combination with:
3. comic\_status  
   this field is updated as the state of the comic changes in the user’s lists:  
   a. when first added, it’s set to ‘unread’  
   b. when marked as read, it’s changed to ‘read’  
   Queries used decide whether the comic should appear in the Unread or Read comics list for the user

### Collection ‘admin\_base’:

Example:

|  |
| --- |
| {  "\_id": {  "$oid": "5c6539e0c8c5d9000431a4b2"  },  "user\_name": "arjan",  "display\_name": "Arjan van der Meij",  "comics\_unread": 13,  "comics\_read": 11  } |

Whenever the admin login is used, this collection’s entries are updated.

User entries into this collection are made whenever a user registers. At that point, comics\_unread and comics\_read fields do not exist.

Whenever the admin page is called, queries gather information on comics in the users’ lists, located in the user\_comic\_list collection.

At that point, all entries in the admin\_base collection are updated with the correct information for the comics states.

### Rationale for this flat approach:

When deciding on how to structure the database, I faced several questions, where the predominant one was how much code I was willing to create to gather all the information by cross-referencing across collections.

I wanted as little as possible.

This led to using the four collections described above, where queries could be based on information about the current user. This keeps the overall code needed to gather the information needed to a minimum.