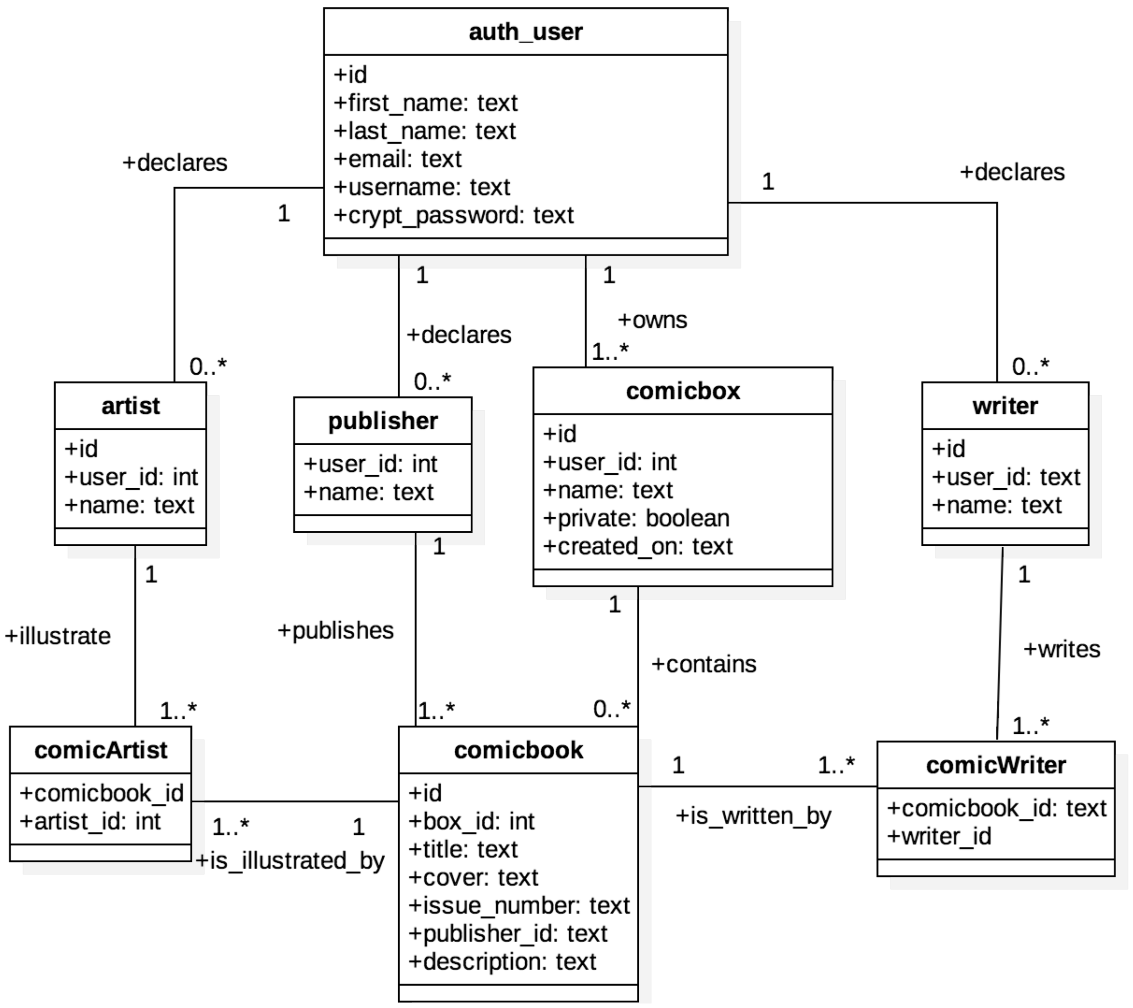
University of York
Department of Computer science

COM000016H

Interactive Application Programming Techniques
Individual Assessment

Exam candidate number: Y0076159
Word count excluding references and figures: 1566

Data modelling



The requirements document for this project states that the following transactions need to be supported:

- Comicboxes need to be: created, removed, renamed, able to set a privacy mode, and able to display their content.
    - Displaying the box's content is achieved using 0..n relation with comicbooks.
    - As users are forced to have an 'Unfiled' box by the design specification and the comicbox and comicbook data are separated into two classes, the deletion of a box can be accomplished without losing the box's comic books which will be 'adopted' by the Unfiled box owned by the same user. Due to this, the 1..n relation between users and comicbox whereby users must have at least 1 comicbox must be enforced.

- Comics (and their publishers, artists and writers) need to be: created and assigned to a box (by default 'unfiled'), updatable, re-assignable (to a box), deletable and duplicable.

- The create and update transactions are achieved through CRUD operations, although redundancy checks are performed to avoid duplicate artists/writers/publishers and comicArtist / comicWriter entries.

- Comicbooks can be re-assigned to boxes by changing their box_id, and as no objects are individually dependent on them (artists/writers/publishers with no associated comics are deleted automatically) they can be removed without affecting any other components in an undesired way. This also allows them to be easily duplicated by copying all fields except the comicbook_id and box_id.

- The separation of artists/writers/publishers into multiple classes from the comic books (using comicArtist & comicWriter classes) allow the user to do individual updates to each comicbook's writers and comics whilst reducing the database size by re-using artist entries across multiple comics where possible (eg: artists can now illustrate multiple comics, and multiple comics owned by the same user can be illustrated by the same artist). As the specifications require 1..n writers and 1..n artists per comicbook, this requires an additional table for each requirement (comicArtist & comicWriter, whereas the singular multiplicity of publishers to comicbooks does not require a subsidiary table to share publishers across comicbooks. This was not expanded to sharing artists, writers or publishers between users so as to maintain the state across user sessions and simplify the code base. This solution was implemented to avoid storing lists of strings, which forced redundancy in the stored data.

The primary key for all tables is an id, except for the artist, writer, comicArtist, comicWriter and publisher tables where a composite key is implemented in the source-code (through a on_insert callback in db.py) to avoid duplications.

The transactions described above are mapped to controllers in their respective controllers, depending on the location at which they are displayed in the application and the format of the data given to the view.

Database class	Transaction	Controller	Method
Comic box	Create box	boxes.py	boxcreate()
	Update box (including set public/privater)	boxes.py	boxview()
	Delete box	boxes.py	boxview()
	Display box	boxes.py	boxview()
	Display 5 largest boxes	default.py	index()
	Display 5 more recently created boxes	default.py	index()
	Display all user boxes	boxes.py	myboxes()
Comicbook	Create comicbook (including duplicating comic using optional get parameters)	comics.py	comiccreate()
	Update comicbook (including changing parent box, add & remove artists / writers / publisher)	comics.py	comicedit()
	Delete comicbook	comics.py	comicdelete()
	Display individual comicbook	comics.py	comicview()
	Display entire user comicbook collection	comics.py	mycomics()
	Search (by keyword &| title &| writer &| artist &| publisher	default.py	search()

User interface design rationale part a: Searching for and duplicating a comic
In the context of this user story, a user wants to search for and duplicate a comic.

After logging in, the user is presented with his own comic collection. Whilst the comic boxes are the most salient widgets (owing to their tiling and bright-colored covers) it is the main content of the website for which users travel to the website in the first place, and therefore it cannot lose precedence to other widgets. The only other feature on this page, is the main menu, which is consistent across all pages of the website. This consistency improves the site's feed-forward, allowing the user to easily find features and execute transactions.

As the user has no known access to the comic, he needs to search for it. This option is made readily available to him/her by the menu's options, the last of which points to a search menu. After clicking on the search menu option, a search form is returned.

This form's function can be inferred from the fields presented and the context of the application; it's submit button is clearly labelled 'Search' to avoid any confusion in it's usage; and it's function is standard: returned results are filtered through every field before being presented to the user, and returned in a table. The form is therefore a feed-forward cue towards the search function. In the case where the search function returns a large number of results the user can re-submit a more specific query based on the values saved from the last query and pre-loaded into the form. The saved search parameters and result provide the user with feedback on his search transaction.

After finding the row for the comic he was searching for, the user's attention is drawn to the darker, bolder view icon on the far right of the row; it's form, in addition to the column's heading and the row's highlighting on hover, create feed-forward for the user to infer that this button will allow him/her to view the selected row in more detail. If the user decides to click the view icon, he will be presented with additional detail of the comic book but more importantly the same set of editing buttons. The universal usage of the copy icon allows the user to deduce its function through another affordance, thereby bringing him/her to the comic edit page. This page is pre-loaded with information from the selected comic, confirming that the user clicked on the correct link. He can then submit this form using the 'Save comic' button (an affordance in terms of the button's usage), and will then be redirected to his 'mycomics' page. A non-modal dialog confirms the transaction's success, providing the user with feedback, but without preventing the user from simply ignoring it as no further action on his part is required.
In effect, the application has provided the user with a series of feed-forward cues and affordances through appropriate naming, shape of clickable buttons, the usage of common button icons and correct pointer icons. Feedback is provided using either information visible in the page or using non-modal dialogs. These contribute to the user's ability to use the site fluidly and efficiently.

User interface design rationale part b: improving a comic box's content
In this user story, a user wishes to improve the quality of content of his worst comic box by removing the 'weakest link', i.e: the least impressive comic book in the box.

After logging in, the user is presented with a page displaying all of his comic boxes. These are tiled into 3 columns, giving the user an itemized & colorful view of each box's content, separated by a contrasting white margin. This clusters the comics by box, clearly marking their borders and content, allowing the user to visualize them with ease. The visual hierarchy and is established through the comic's clustered layout and the difference in background color, splitting the page into a menu section at the top, and content in the main body. Using this layout, the user can quickly compare the content of his boxes, and view more details of the box in need of pruning using the view icon positioned in the top right of each comic box.

The user is now on the box's display page in which a horizontal line of comics is paged, with buttons for the following or previous set of comics placed under the leftmost and rightmost comics and therefore following the user's line of sight. Whilst it is already quite obvious what paging buttons do for a majority of users, the additional feed-forward cue created by their positioning further reinforces the user's understanding of the paging system.

After having perused the comicbox's comics using the previous/next buttons, the user can decide which to delete. This can easily be achieved by hovering over the comic to reveal additional buttons, which was done in order to capitalize on the user's most intuitive action: placing the mouse cursor over the comic he wishes to delete. The timing at which these additional affordances are revealed simplify the layout of the page, whilst associating the individual affordances with the comicbook over which the user hovered. This association, in addition with the form (generic icons were used) and position of these icons can be used to infer their usage and target: editing/deleting/viewing in relation to the comicbook.

Once the user clicks the delete icon, a modal confirmation message will appear to ensure the user did not accidentally request the transaction. Upon confirmation, the transaction is executed, and the visual representation of the now deleted comic shrinks in size until it disappears, and any floating comics to its right will take its place by shifting leftwards. This provides a visual (in addition to a non-modal confirmation dialog) feedback to the user that the comic has indeed been deleted. The change in visual layout therefore strongly confirms the user's action, whilst also giving him/her affordances as to the transactions that can be performed on each displayed item.