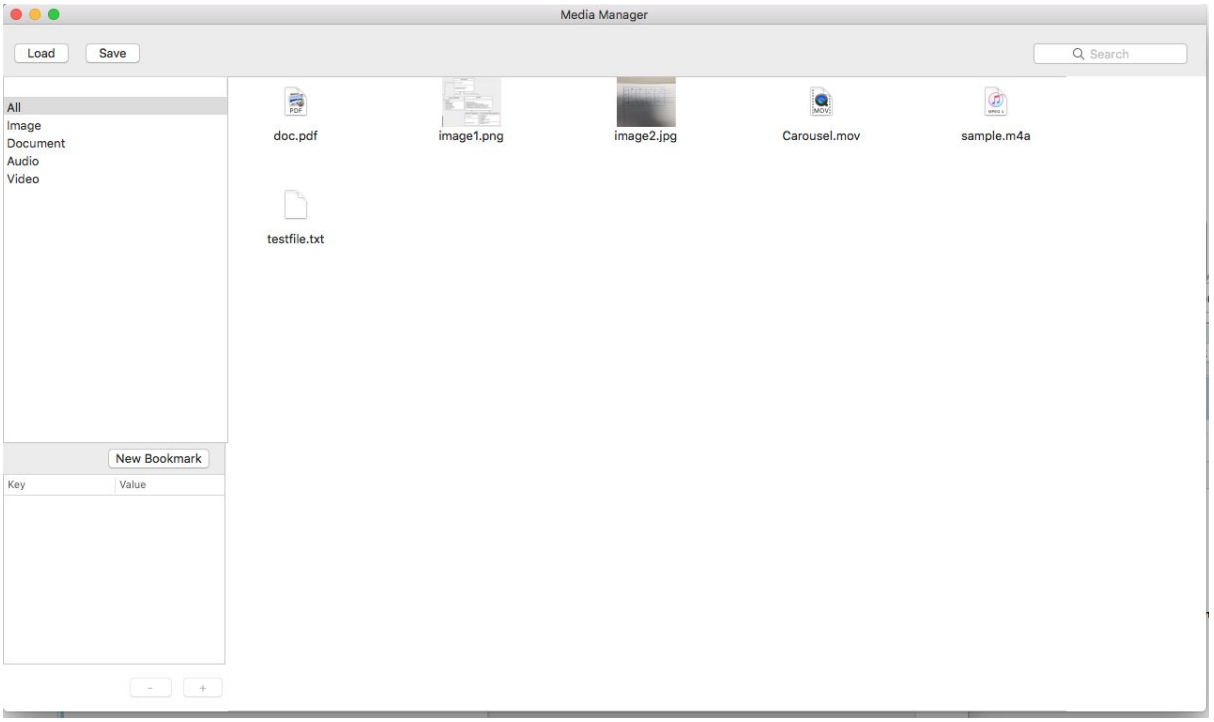
NOTE: THE PROJECT MUST BE STORED IN /home/346 DIRECTORY FOR THE TEST JSON FILES TO WORK

Names: Jerry Kumar (3921871) and Ryan Mcgoff (4086944).
Github usernames: jerydoom and T0MBRaider
Github group name: hamzabanana

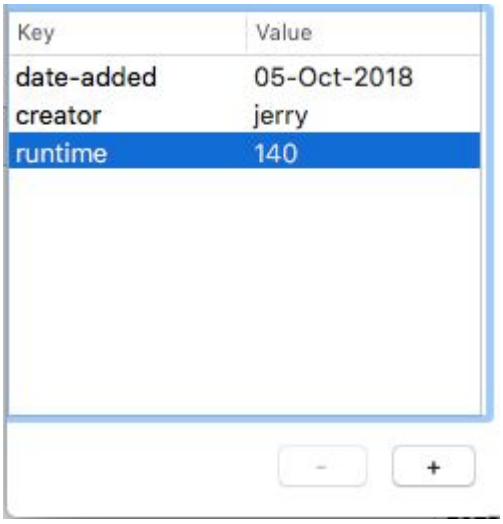
GUI Design Decisions

For our app we decided to model our interface as a collection. Similar to other Mac and IOS applications, the files in our library are displayed in a tiled format with previews and titles to allow users to quickly and easily identify and select their media. We chose this approach (rather than a standard table view), as we felt that with previews and file titles, it was a lot easier for users to access and comprehensively view their collection without needing many additional selects/clicks. We followed a similar design to that found in Apple's finder app (e.g. our search bar is on the top right, navigation is fixed on the left etc) so a user would be somewhat familiar with how the navigation worked.



We decided that the best way to display the details for each collection element was in a panel in the bottom left corner of our app on the same window. This panel allows users to quickly see details of whichever media they select, but more importantly it is displayed on the same screen. This makes it easier for the user as they do not have to deal with multiple windows when trying to access data. The controls to manipulate the metadata details are also located in this panel and are programmed to minimise user overhead. In assignment one, we sent warning messages if users attempted to complete actions we couldn't allow (e.g. trying to remove a required metadata type or add metadata when a file isn't selected). With this assignment, we decided to program these validations into the GUI so the user never needs to worry about it. For example, the add and remove buttons are not enabled

until a media file is selected, and you can't click add if an add window is already open etc. All of these decisions were made to increase usability.



Data decisions

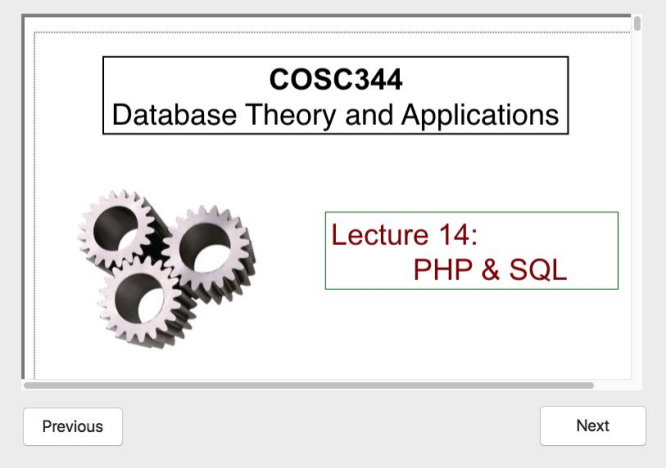
We handle our libraries as split between the view controller and the collection view data source. The data source has its own collection (DS collection) and when it loads in data, this collection is copied to the view controller (VC collection). The VC collection is the stable copy of the collection that does not change. When users filter or search, we use this VC collection as a reference point to conduct the change and then we change the DS collection appropriately. E.g. if the users want to search for image files, we used the VC collection to get the appropriate files and then overwrite the DS collection to be just these files. This approach allows the changes in the collection to be quickly and smoothly updated, and keeping a "hard copy" of the collection helps us make sure that we don't accidently change things we didn't mean to. This also allows us to reload the new requested data easily by simply reloading the collection view that the data source is attached to.

We also decided to disallow users trying to load multiple copies of the same library/json file. This is because there is no reason to have 2 copies of the same file in a library as both copies would be identical in every way. In other words, after loading a json file with the data for a set list of files, loading in the same json is pointless an so we removed the ability from users to do so.

Implemented Features

- **"Users can navigate through a given mediafile, including moving forward and backward for video file, moving to next image in the same folder, moving to the previous image, and jumping to a specific media file while showing a list or a grid."**
  Our movie and audio files have play bars that allow users to more forward and backwards in the file, and our Collection View grid layout allows our users to jump to specific media files.

- "**Users can navigate between different media files within a set, providing both "local" navigation, such as "next media" and "previous media".**
  Our Collection View accomplishes much of this by default. Due to the grid layout with previews and names, users can easily navigate between files to choose which they wish to select and view. On top of this, in our file specific window, there are next and previous buttons implemented that allow users to navigate between files in the current selection (E.g. If they select to filter by image and open up and image, the next and previous buttons will navigate through only images)



- **"An indication is provided of the current media file being viewed, with the specific notes, and the current time for example for the video type, or page for the text file that is being viewed."**
  In our content window, users can view the media file, the notes and the metadata associated all in one window. On top of this, there are specific notes as well e.g. users can see the time elapsed on videos.

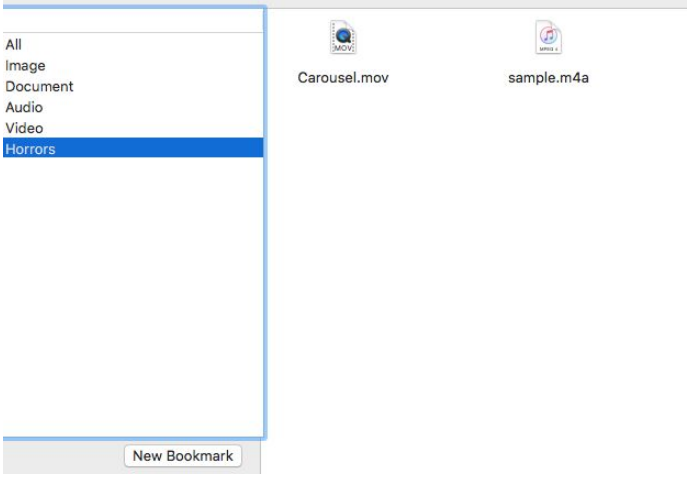
- **"A "useful" menu structure is implemented, that complements your other user interface controls"**
  We have implemented a minimalist menu structure that gives the users the abilities required such as load, save and add bookmark.

- **"The application's controls should resize in a sensible manner when its containing window is resized"**
  Our application maintains a fixed size navigation bar and buttons, resizing the window only alters the size of the collectionview. This is similar to how Apple's Finder application handles resizing - keeping the UI panels a fixed size while changing how many file items are displayed in a row/column, which helps users not get lost in the app. We also have a minimum size so all our views are displayed appropriately even at the application's smallest size.

- **"Users can record brief textual notes that are related to a particular media file."**
  When a user selects a file in collection view, a previously hidden notes panel pops up on the right hand of the screen. This notes panel contains a text field editor, the user

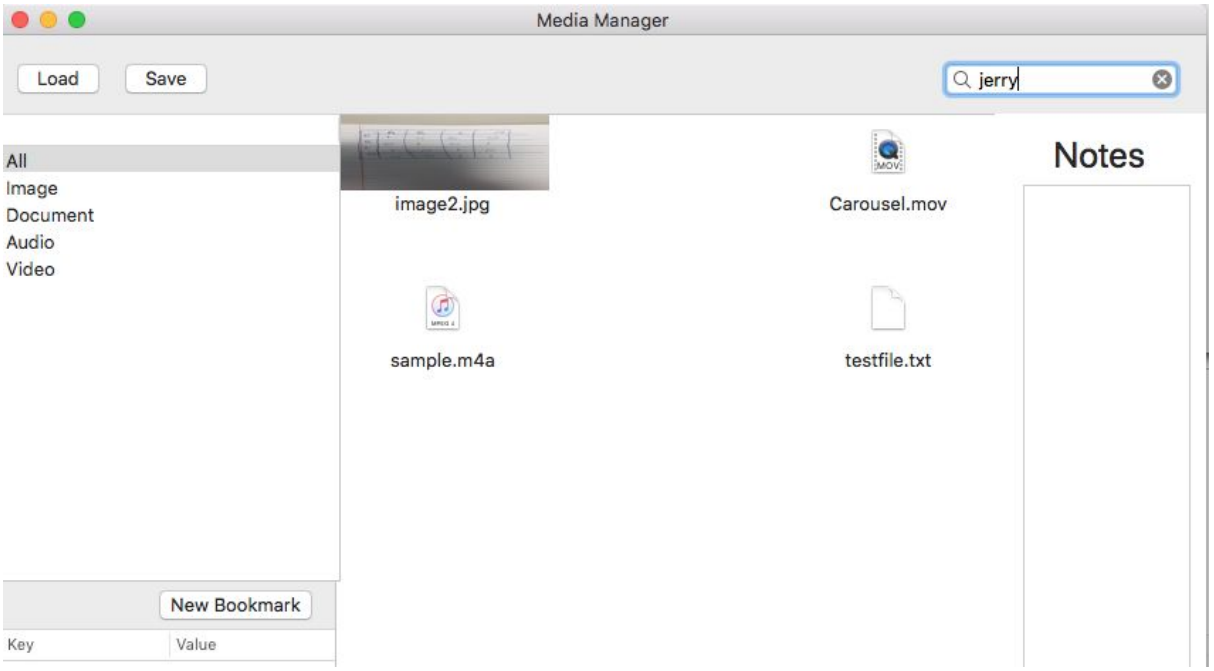
can type a number of words to this, which is then stored in a variable in the file's object for exporting and persistent storage.

- **"Users can bookmark particular topics of media or bookmark a particular media file then later use these bookmarks to jump back to the appropriate media file or list of media file."**
  In our application, a user can create bookmarks by clicking the add bookmark button, this display a dialog box that lets the user enter the new bookmark name. Once a new bookmark name has been chosen by the user, the bookmark is then added to the same mutable array that our media filters (video/audio etc) are stored in. The user can then add a bookmark to any file from collection view by selecting a bookmark from the top navigation bars drop down menu. All files that have a particular bookmark can be filtered in the same table that lets users filter by media type.



- **"Search within a set of media file related to a particular topic or particular type or media file."**
  In our application, users can quickly search the media files by topic and type. Because a user will search by media type more often than topic, we choose to prioritise search-type by implementing type filters that easily allow users to filter the collection by type of media. If a user wishes to search by topic, they can do so by

using the provided search bar in the top right. By separating *“search by media”* and *“search by topic”*, we allow the user to search for metadata while also filtering by the file’s type (eg: displaying all the audio files that have a resolution of 720 x 1080).



- **Your “About” panel must be customised to include some relevant information about your project, e.g., giving credit to the creators of any resources that you acquire from elsewhere, such as icons.**
  We created a simple credit.rtf file that automatically overwrites the default about panel. Our credit file contains information about our project as well as github links to the 3rd party code we used.

- **“The same import/export functionality for assignment one should be a feature”**
  Import - the user has two choices when it comes to importing. They can either import via a JSON file or import a file by dragging an appropriate media file into our drag and drop interface. The drag and drop view checks against an array of expected file types, if the file being dragged matches, the collection view temporarily switches its background color. This visual representation helps signify to the user that the application has detected their file is of an appropriate type - similar to how Google Drive or DropBox drag and drop features work.
  Save - After modifying the data in the collection, users can save these changes into a new (or as a replacement of the old) JSON file. This data is persistently stored and can be reloaded if the user wishes to.

- **Extension support**
  We support all the specified extension types .png, .m4a, .mov and .txt, in addition to PDFS and JPEGs (see additional functionality bellow for more details). For files trying to be added that are not supported, an icon is displayed instead. The drag and drop feature only allows the allowed extensions to be added, and only allows one file to be dragged at a time.

Additional Functionality

- **Implement persistent storage of the notes made on media files.**
  We achieved persistent storage of our notes by storing them as a variable inside the File's object. When the user hits the save button, these notes (along with the rest of the file's metadata) are saved to the exported json file. If the user decided to reload the JSON collection file later on, the previously saved notes are checked for in our File Creator Class, if they exist for that particular file they are saved to the newly created File Object's notes variable, if not this notes variable is set as an empty string.

- **Fast search processing of the media files by topic and by type.**
  When any metadata is added to the collection or imported via JSON, it is added to a Collection dictionary variable (along with the associated filename). This lets our Collection quickly search through its dictionary to find the filename with the associated metadata/topic. Any time the user modifies the input of the search bar, the search bar calls Collection's search function (implemented in assignment 1) and finds the files that have matching metadata. These files are added to a second collection object and are displayed by collection view. We have two seperate collection objects, a collection containing every file and a collection of the returned search results. This is allows us to keep the original state of the library and quickly switch from displaying the search results to displaying the full collection.

Our own additional functionality

- **Dynamic, editable metadata, bindings.**
  We implemented metadata displaying via bindings. As files are selected, our dynamic objective c array is updated via a binding to an array controller and users can view each file's metadata. Users can also edit metadata via the UI. This extensions allows users to add metadata to any selected file as well as remove metadata. On top of this, users can also modify values of metadata via the table. All of these changes are reflected in the model and are persistent. On top of this we added validation checking to make sure that only metadata that the user is allowed to modify is able to be modified and only data that CAN be removed, is available to be removed. These extensions all took significant time and we believe the work is worth 10 points.

- **Validation on windows**
  For both of our windows (Add metadata and Add bookmark), we implemented in build validation and error messages. For Add Metadata, if users attempt to add metadata that already exists, or overwrite a required variable such as creator or runtime. An error message appears and we disallow them from doing so. Similarly in the Add Bookmark window, users aren't allowed to add bookmarks that already exist, or conflict with one of the preset filters. We believe this validation implementation is worth 10 points.

- **Using notification center**
  We used notification centre as a form of local broadcasting, to enable us to communicate between our view controller and custom views. An example use case was for our double click function. Because collection view does not have its own double click function like table view, and because we were using view controller to control multiple components of our app, we couldn't override the onclick function - we instead create a custom view which we overlaid with our collection view to register double clicks. Upon a double click, the custom view sends a notification to our view controller, the view controller checks if a file in collection view is selected and opens a new content windows containing that file if it is. This creates the effect of double clicking on a file in collection view on having it open up in a new window.

- **Extra file extensions**
  Even though it is not required, we allow users to also add pdf files to be viewed in the content window. We also allow JPEG's.

- **File date stamps**
  As per our extension in assignment one, we have updated our metadata to dynamically store date created and date modified. Date stamps are added to any file when it is loaded, these are stored in the metadata called "date-added", giving the user an indication of when they added the file to the collection. Anytime the user modifies metadata or creates new metadata, the application sets the "date-modified" metadata to the current date. This extension required a lot of fine tuning to allow metadata arrays to be accurately updated. On top of this, for files that are dragged and dropped we have to manually find or create these variables. We believe this work merits 10 points.

- **DropView**
  We created a custom drop view that overlays our collection view, this provides users with an alternative way to load files into the media library. If the user drags an item with the appropriate file extension over collection view, the drop view switches to a grey background color indicating it is safe to drop the item. Once the user has "dropped" the file, dropView extracts the file's path from pasteboard and posts this path to notification centre. Our main view controller then takes the path name and determines the type of file it is. Then, using a series of objects and libraries, it extracts the important data from the file, such as the resolution of the video file, size of the image and audio runtime. Because these details aren't provided to us in a JSON file like our other import option, we needed to use the given the file path to manually find the other metadata for overselves. This was a huge task and we believe it is worth ten points.

Testing

In order to test our application we would gather a sample group of at least 10 people and get them to use our app. Through this process we would give them rough guidelines on features we want to test and make sure they are working as expected, as well as give them free time to test any ideas they had. The goal of this as we would outline would be to "try to break the

app” AKA do something that wasn’t allowed, or try something that we had not thought about. We would try to make this sample group as diverse as possible, and we would also bring a questionnaire with set questions asking the users to rate their experience with app. This would included questions on interface usability, design, ease, and many others as well as general comments they had for us.

Roles

We designed our media manager User Interface together and discussed the design principles we wanted to use, as well as the base ideas for how our classes would be implemented and interact with each other. From here we seperated to complete specific tasks. We found it easier to work on one computer as merging storyboards was incredibly tricky, so we shared Jerry’s login for most of the assignment (hence why most of the commits are from Jerry).

Jerry took the lead on the array binding and MVC set-up for the main view and the view controller along with the validation checks we wanted to implement. Ryan took the lead on designing our additional windows, customs views like drop view and setting up the notification functions. Together we wrote this report, commented the code, debugged each command and regulated each others code to make sure that we were each following the principles we set out to achieve at the beginning. We came up with the extension ideas together and implemented them as a team.

Other Notes

There were a few changes that we would have liked to implement if we had more time. Firstly, in our content window (when users double click a item to bring up an in depth preview and the details on that file), the notes and metadata of a file aren’t editable. We designed it this way because we saw the content window as a in depth look into the file, but not as a means to edit. In hindsight, we think it would have made more sense to allow users the freedom to also edit the file while in the content window.

Secondly, we thought that perhaps adding the ability to remove bookmarks, as well as remove specific files from bookmarks would have been good functionality to add to the application. We were constrained by time and didn’t have enough to complete this addition, but it would have been very simple to do by simply adding a delete bookmark button that removed that bookmark from all files.

If we had more time, we would also have liked to implement a more comprehensive menu structure that made full use of all the functionality in our application. We feel that the menu structure we did is appropriate for the application, however adding extras such as the add metadata options etc would have been ideal.

Also, our constraints give some console errors when we build. The application resizes appropriately and works as intended, but these errors are still being given. If we had more time to explore why these errors were occurring and how to fix them we would.

We provided a JSON file with a selection of images, documents, a video and an audio file for convenience. **NOTE: THE PROJECT MUST BE STORED IN /home/346 DIRECTORY FOR THE TEST JSON FILES TO WORK**. But you can drag and drop any files into collection view and test it that way if you wish.