

Revision Tools for the Smartphone Era: The Creation of a Veterinary Revision Aid App

Murray Crichton, s1232200

Honours Project (INFR10044)

Abstract

This report details and discusses the creation of a revision assisting smartphone app, directed at veterinary medicine students. The project was proposed by Amy Livens, a vet student at the University of Edinburgh, who provided the motivation and material necessary to realize her vision for the app. The project goal was to convert a set of anatomical diagrams Livens had created, contained in a bulky physical form with little hope for replication, to an easily distributable smartphone app. The unique format Livens had used in this diagram set, designed to ease revision of complex anatomy and physiology, was to be preserved and presented in a digital form.

The goal of the project was accomplished through the completion of a desktop editor application capable of converting hand-drawn diagrams to a useable format, along with a smartphone app capable of displaying the processed material. The final smartphone app, the part of the project to be presented to the public, was then evaluated by both Livens and an anonymous group of testers. A generally positive response was received from both Livens and the testers, such that the outcome of the project could be considered one of reasonable success.

1 Introduction & Synopsis

1.1 Introduction to the Project

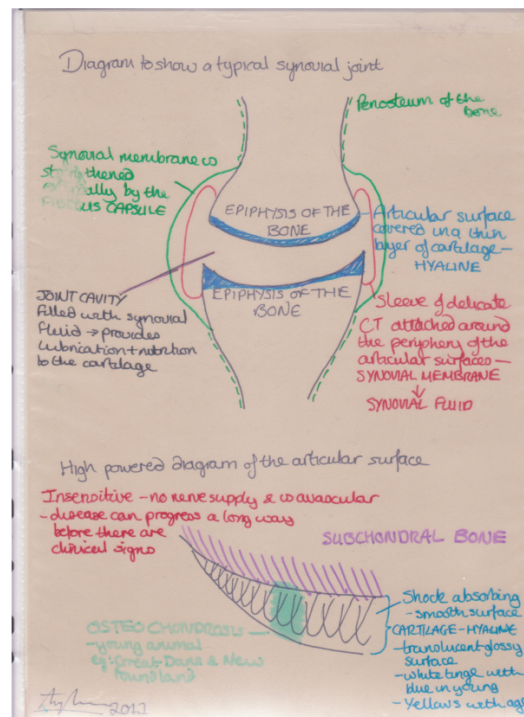
During a veterinary student's time at university, they are required to absorb a vast amount of information. A high degree of competence is required to be a successful vet, and so any student in the field must succeed in learning, among other things, the complex anatomy of various animals. The process of building up and maintaining this anatomical knowledge is a mandatory, and not insignificant, part of the process involved in training to become a fully fledged vet.

The goal of this project, proposed by veterinary student Amy Livens, was to convert her own personalized revision material, a large set of handmade anatomical diagrams [A1], into a format suitable for distribution to other students. Livens provided a part of this diagram set, which she had previously created, to be processed and converted to such a format. This subset totalled over one-hundred and thirty hand-drawn diagrams, and being a mere part of her full revision material, only detailed the general anatomy of the dog.

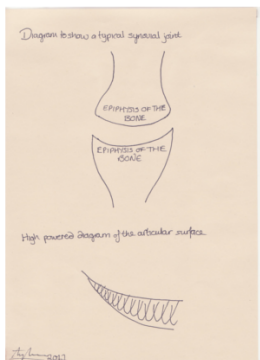
The original motivation for the project began with a strong interest in Livens' diagram sets from her fellow students. She became inundated with requests to borrow the folders containing her revision material: the sets had a unique feature, in Livens' usage of stacked acetate sheets to allow layered diagrams to be created (see Figure 1-1 below).

This was intended to make revising the complex anatomical structures easier; breaking information into more manageable chunks in order to help build up knowledge gradually and aid memorization. Judging by the interest of Livens' peers, this format she had created was an excellent means of presenting complex diagrams, and so the material she had produced was of great desire.

The issue, then, was that demand far outweighed supply: Livens had created these sets of diagrams to aid her own revision, and as such would be heavily utilizing them during exam periods and beyond. Furthermore, even under ideal circumstances, sharing a single set of diagrams (despite being split, as they were, into multiple binders based on the animal concerned) between multiple students would always create issues. Other students could have replicated the diagram sets by hand, but this would have been extremely time-consuming, incredibly dull, and relatively expensive in terms of material cost. Livens noted that her own process of creating the original set had been one of complete tedium, and required the duration of a Winter break for the bulk of the work. A solution, therefore, would be to find some method to quickly create a great many copies of the set, with minimal expense.



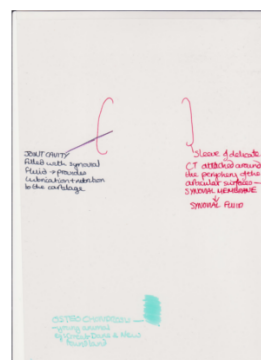
Combined layers



Base layer



1st layer



2nd layer



3rd layer

Figure 1-1: Livens' original layered diagrams, which were converted to an app

Livens met with her anatomy lecturer, one Dr. Kempson, to find a better solution. Creating additional physical copies, likely by scanning and printing to duplicate the diagrams, would still have incurred a hefty material cost (the acetate sheets required being an expensive medium), and would introduce the additional problem of distribution. Another issue, the physical size and weight of the final set, due to the bulk added by layering these sheets, would not be solved either. To publish a book of the diagrams (with some appropriate solution to maintain the layering effect) would incur similar problems, likely with much higher costs. Printed media seemingly out of the question, an electronic version of the set was considered instead.

A smartphone app, capable of displaying diagrams in a way which would mimic the original folder with its seemingly-unique layering, emerged as a viable solution. Compared to a standard or web application, this would allow a user to access the information conveniently while away from a desktop environment or in circumstances where internet connectivity could be limited. The costs of the physical alternative would be avoided, along with the aforementioned size and bulk. For development of an app the Android operating system was to be targeted, this being both Livens' personal choice as well as the operating system running on the vast majority of smartphones.

1.2 Overview of Published Literature

To set the project in the context of "published literature," existing anatomical references, related software, and the theory behind the application (and others like it) should be examined. Existing products, be they books or software, with potential as revision material will first be discussed, followed by the general design principles of software learning aids. How these principles were applied to this project will be summarised here, and some aspects will be inspected in greater detail in Section 2.4.

As previously mentioned, the folders of diagrams Livens had produced were of a unique nature in the sense that seemingly no other published physical material shared their layered characteristics. One can speculate about the possible reasons for this: the cost, size, and physical bulk of, for instance, a published book (as discussed in Section 1.1) of layered diagrams may outweigh the advantages such a product would present; a book of this style may not be in high enough demand to be commercially viable; or perhaps the prevalent accepted solution, standard labelled diagrams, is seen to be sufficient. Whatever the reason, comparable published material did not appear to exist, thus sparking the interest in the original diagram sets Livens had created.

In fact, these sets had more in common with existing software solutions for presenting anatomy. Outside of simple diagrams available on the web (essentially presenting the information as it would appear in a standard anatomy book), many three-dimensional anatomy viewers could be found, which appeared to be the closest in spirit one could get to Livens' layering method. For dog anatomy specifically, a few web searches brought up examples of both free in-browser software [1] as well as paid-for software packages [3]. While not entirely comparable to Livens' approach, these 3D models allowed layers of tissue, muscle, and organs to be gradually stripped away, exposing whatever lay beneath: a concept with some reasonable similarity. Neither of these solutions, however, matched the requirements presented for this project: the browser-based

viewer (besides being implemented in Adobe Flash Player [2]) required the user to be connected to the internet to function, and the paid-for package was both desktop-only and prohibitively expensive. These tools were likely better suited for use as references, rather than revision aids, although could be used for general revision purposes.

Another issue present in the previously mentioned browser-based solutions was a common lack of mobile support. Being able to explore the material easily from a mobile device was naturally an important aspect of the project, and while in theory one could access available diagrams or 3D viewers in a mobile browser (the viewer seen in [1] requires Adobe Flash Player, for instance, and so was not supported on Android devices [2]), of the resources examined none were optimized for mobile viewing. These therefore presented a clunky and frustrating user experience, creating an irritating hurdle to accessing the material. A more reasonable view, then, of products available on mobile devices (specifically Android devices, in this case) came from searching for downloadable apps on the Google Play store.

True to the expression “there’s an app for that,” there were, in fact, already a handful of apps aimed at animal anatomy, especially dog and horse anatomy. However, these were similarly designed not as revision tools, but more for general reference. There existed software versions of the “traditional” anatomy book with labelled diagrams [4], as well as 3D models allowing one to drill down through the various layers of muscle, tissue, and organs viewing the names and functions of various body parts [5, 6].

The “traditional” approach, again, did not match the proposal for this project, being largely similar to every other standard book of anatomy available, just with the added advantages (data as opposed to a physical object, patches to fix errors, and so on) of a smartphone app. The 3D model viewers, producing an effect in some ways similar to Livens’ layering technique, were much closer to the project requirements, but with some key differences.

Firstly, they were organised as a full anatomy, convenient for reference but not as suitable for revision purposes, where one wishes to focus on a small subsection of the animal in order to learn or revise it fully before moving on. They were rich in information, but limited in scope: each app (only those illustrating dog anatomy are given as examples above, but others existed covering, for instance, horse anatomy) contained only information on one animal, and were therefore incapable of providing a comprehensive revision tool for veterinary students in one convenient package. In addition, these apps had an (admittedly small) monetary cost associated; ideally learning or revision aids should be free, to be as accessible as possible. In summary, while there were many resources or tools available which could be used to aid revision, be they published books or software solutions, there were none specifically arranged as revision tools aimed at undergraduate veterinary students.

The theory, however, behind creating an effective piece of software to aid learning has been widely detailed, discussed, and documented. Many of these ideas and practices are not confined to software applications, either, but are general good practices when presenting material to be

studied. It is important to note, however, that not all of these principles apply to aiding revision, and often can hamper revision efforts [7]. This is generally due to excessive information being presented, which could aid learning, but simply bloats the content and wastes the time of a user already reasonably familiar with the material.

Additionally, good user-interface ideologies play a role, as presenting a solid user experience eases frustration and provides the best environment possible for learning or revision; this will be discussed further in Section 2.4. Some of the principles given here, however, could not be realised in this project, whether due to time constraints or otherwise, and this will be noted where appropriate.

The principles of effective e-learning, presented in *Six Principles of Effective e-Learning: What Works and Why* [8], are as follows:

“The multimedia principle - adding graphics to words can improve learning”:

Learning is improved when text and relevant graphics (or some other combination of two media types) are combined. The Dual-Coding theory [9] is potentially of relevance here, the idea that remembering information is made easier by encoding it as two separate representations (for example, images and words).

“The contiguity principle - placing text near graphics improves learning”:

A simple but impactful principle, by ensuring any information related to a graphic is kept as close as possible to that graphic learning sees an “average improvement of 68%” [8].

“The modality principle - explaining graphics with audio improves learning”:

This principle was not particularly relevant to the revision focus of the app, being more effective when the material being explained was “unfamiliar to the learner” [8].

“The redundancy principle - explaining graphics with audio and redundant text can hurt learning”:

This was highly relevant to the revision focus of the app, as having to waste time sifting through information when revising is even more damaging than when learning [7].

“The coherence principle - using gratuitous visuals, text, and sounds can hurt learning”:

This principle warns against both unnecessarily decorating material (for instance, adorning information with an irrelevant image purely for the sake of appearance), as well as using excessively lengthy methods to present information (for example, expanding text around information to make it more readable rather than briefly summarizing).

“The personalization principle - Use conversational tone and pedagogical agents to increase learning”:

This principle was of no great relevance to the app, being directed towards learning instead of revision.

The motivation to keep the content within the app to strict minimum to aid revision was further reinforced by the theory of the “experience effect” [10], which advocates the presentation of only the required information to allow a user to focus entirely on the process of revision. As previously mentioned, there was little other than the base required information contained within the app, satisfying this requirement.

1.3 Overview of Completed Work

The original summary specification for the project was as follows:

“Build a smartphone/tablet app to help veterinary students with revision. [...] Being able to build up a picture slowly and in the student's own time works better than looking at a final picture with everything on it. Think of a diagram with acetate sheets over the top to build up the final picture.”

To achieve this, there were several sub-goals that were to be completed. These were:

- Process the source material (hand-drawn labelled diagrams) into a digital form (discussed in Section 2.1)
- Convert the digital material into a format ready to be displayed in a mobile app (discussed in Section 2.3, Section 2.5)
- Create a mobile app to display the material, capable of loading any set of diagrams provided in the correct format (discussed in Section 2.2, Section 2.4)
- Evaluate the finished app against the original requirements, and run a study to evaluate the app as a whole (discussed in Section 3)

2 Discussion of Work Undertaken

2.1 Requirements Analysis

The original specification, given in Section 1.3 above, outlined the basic requirements for the project. In addition to this, a meeting with Livens provided some further information: the app was to be, in essence, a software version of the revision material diagrams set she had created (as discussed in Section 1.1). A basic diagram package, the general anatomy of the dog, was to be made available within the app. A means to later expand the on this, a method of creating more content and having it available within the app, was the ideal outcome. Any features beyond this (an example discussed was a self-test function to help a user discover areas they needed to revise further) would be welcome enhancements but were not core requirements; if the basic work was completed sufficiently early in the development period these could be implemented.

The requirement breakdown, then, was similar to the list of goals presented in Section 1.3:

- The source material, provided by Livens, was to be converted into a digital form. This was accomplished with the use of a conventional flatbed scanner, to copy and save the diagrams as high-resolution images.
- The digitized material was to then be converted into a format readable by a mobile app. This was accomplished by creating a desktop editor application to process the

high-resolution image scans, annotate them, add other information, and export the final combination in a package format readable by a smartphone app.

- A mobile app capable of displaying any given diagram package was to be created. This was accomplished by creating a small skeleton smartphone app which could then download revision material, and save it for offline use, through a simple repository system.

Several options were considered for the task of inputting and converting the large source diagrams set. Recreating the diagrams electronically, either by hand-tracing scans or drawing replicas in an image editing application was one such possibility. Other information (labels, annotations, notes, and titles) could then be added back as plain text, formatted and displayed as appropriate. This idea was quickly abandoned: the amount of time and work required to fully recreate the diagram set electronically would have taken a large component of that available for overall development. While superior in-app diagram quality could have been achieved this way, it would have been at the expense of all other aspects of the project, and would have been a task more suited for a Computer Arts student.

Another option was to scan the original hand-drawn and annotated diagrams, process and enhance them to remove as many imperfections as possible, and then use the resulting images as material to be displayed within the mobile app. The primary problem with this route was in ensuring the readability of the labels and annotations. As the source material was annotated in Livens' own handwriting, this would have been used in the finalized content, scaled down to fit on a mobile display. At best, this method would have produced passable quality labels and annotations, and at worst, a cramped mess of unreadable handwritten text. I can personally attest to the difficulties found in reading much of the text, from my experience transcribing the notes (partially due to my lack of familiarity with the subject area, but often due to imperfections in the material or the apparent deterioration of the text written on acetate sheets). With small mobile screen sizes added to the equation, these issues would only be magnified.

Instead, a solution combining parts of each of the options presented above was chosen. The images were to be scanned in; enhanced and cleaned to remove any imperfections, markings, or detritus made visible in the scanning process; any visible text components removed, and added back as true text data in a relevant form (generally labels displayed as an overlay). This would require a specifically tailored piece of software (discussed in Section 2.3 and Section 2.5) created to process the source material efficiently, and ensure none of the information present in the original diagrams was lost.

2.2 Technology Choices - Smartphone Application

There were many options available for creating an app to run on Android devices. The most prominent of these was creating a "native" Java application, using the Android Studio [11]. In addition to this, however, multiple third-party development toolkits were available, catering to various different needs or allowing other programming languages to be used [12]. In most cases, these third-party tools were capable of creating apps with native or near-native performance and

features, easing development significantly by allowing a developer to choose the language or programming style most familiar to them.

Initially, the planned method of implementation was to create a native Java application, this being the advertised route (perhaps unsurprising, given that the Android Studio was offered by Google, the developers of the Android operating system) for potential app developers. This seemed a sensible strategy, utilizing the official tools and using the standard language for the target system. While I would not personally choose Java over other languages (due in part to a lack of experience with the language, but largely due to a general dislike of the syntax, style, and documentation), in this instance it seemed a reasonable choice, and provided a starting point for development. After a period of experimentation using the Android Studio, however, an alternative method, within the native development suite, emerged: an element (the WebView [13] widget) which could be added to the interface of the app was capable of loading locally-stored web pages, transparently displaying them (utilizing the rendering engine of the basic smartphone browser to power the effect) to the user.

This option, to develop the app using web technologies, was an appealing solution: I had far greater experience with the various web languages used compared to Java. A basic mockup of the smartphone app was therefore implemented as a web application using HTML, CSS, and JavaScript [14], and with minimal effort was implanted within a native Android app wrapper. The ease with which this mockup was created was a significant improvement over the convoluted process of implementing a similar piece of software in pure Java, and the notion of developing the entire app as a true native application was quickly abandoned. This decision, to switch to programming using web technologies instead of Java, would save a great deal of development time, and due to the relatively simple requirements of the mobile app (alongside some other technology choices yet to be discussed) would not have a negative impact on the final app produced.

While it would have been entirely possible to create a hybrid web and Java app, using native solutions for problems such as network or filesystem access while keeping the user interface confined as much as possible to the wrapped web application, the Android Studio was not designed for to promote such implementations.

Other solutions, however, were available. The first discovered was Adobe PhoneGap [15], a development tool allowing Android apps to be created entirely using web technologies. This seemed an ideal solution, Adobe having provided an easy-to-use interface for compiling web code into apps, and documentation on how to create appropriate code. After a period of development using PhoneGap, however, several problems became apparent.

Firstly, issues with certain required PhoneGap plugins, meant to allow interaction with a smartphone's file system, were encountered. The documentation on installing and using these plugins seemed to be incomplete or outdated in places, preventing the immediate resolution of this problem. This led to a temporary solution, where all the material to be displayed within the app

was hard-coded and compiled into the app itself (violating the requirement that one should be able to easily add viewable material to the app, or that existing material should be modifiable). This caused an additional problem: the app became extremely bloated, the storage requirement and time necessary to install it was pushed far past the point of reason. This problem became critical when the bloated nature of the app caused it to run afoul of the PhoneGap upload size limit, preventing this workaround from functioning when the all desired content was added. This issue remained unsolved as development in various other areas continued, until it was finally the main roadblock preventing further work. It seemed then, that while PhoneGap had the potential to be a convenient solution, it simply couldn't function well enough to be a viable development platform. Thankfully, another alternative emerged, a sister platform to Adobe's PhoneGap : Apache Cordova [16].

An open-source project on which PhoneGap is based, Cordova had fewer gimmicky ease-of-use tools, and more basic user-empowering ones, as well as up-to-date documentation. Porting the PhoneGap code over was trivial, given that the platform was essentially the same. Cordova was chosen as the final implementation platform for the mobile application, being a well-documented and feature-rich solution, capable of meeting the requirements of the project. The performance was more than sufficient for the simple project requirements, while the platform was quick and easy to develop in, using web application development tools and languages. Testing the application was made somewhat easier, given the web code could be evaluated, in part, on the desktop workstation being used for development. While there were many other tools that could have been chosen to implement the project in, Cordova fulfilled all of the requirements, while using a set of languages that I was personally comfortable and familiar with, greatly easing this process.

2.3 Technology Choices - Package Creator and Image Editor Application

Compared to selecting technologies to use when creating the smartphone app, the choice of platform in which to build the desktop application to process, annotate, and label the high-resolution scanned diagrams was not so initially obvious. Given that the vast majority of the work here was in implementing methods to process the images, selecting a solid image-processing library was the primary concern. My personal choice of programming language for a relatively small project such as this was Python [17], ideal for fast prototyping and development, and additionally being the language I have the most experience using outside of web application technologies. While using web technologies would have been a potential solution, the performance demands here warranted the use of a faster, more heavyweight implementation, which could be provided, relatively speaking, by Python.

An image processing package within the Python ecosystem was therefore required, which had to be capable of manipulating high-resolution images in as close to real-time as possible. Using the popular OpenCV [18] here seemed a natural choice, it being a renowned Python library for image processing, with a sensible enough interface that I had prior experience using, the advantage of familiarity again hastening the development process. As the name would suggest, the library is open-source, and is in fact "written in optimized C/C++" with a set of Python bindings provided

when installing. This underlying use of C and C++ ensures the fastest possible execution, as the purpose of the library is to provide tools for real-time image processing, where speed is clearly key. While the image manipulation component of the editor application may not necessarily be what one would associate with real-time processing, where handling video feeds is the common case, it did still require sufficiently fast image processing to ensure the smoothest user experience possible.

Other popular Python packages aimed at image manipulation were available, such as the now-dated Python Image Library (PIL) [19] and its up-to-date fork Pillow [20], however OpenCV was chosen on the recommendation that it was a more fully-featured of these options [21]. Outside of these alternatives, most other packages appeared to be for more general use (and so not featuring OpenCV's fast C/C++-powered execution), or focussed on wider mathematical applications, with image processing capabilities existing only as a convenient side effect of large array or matrix processing functionalities. OpenCV was therefore the package of choice for the implementation of the desktop editor application, and proved well suited for the task.

In addition to the image processing library, a few other packages were chosen to help build the editor application. A data format for exporting titles, labels, and other information alongside the images of diagrams was needed, and JSON [22] was chosen here as a lightweight option that would be easy to parse within the JavaScript-based mobile application. Already utilized by OpenCV, Numpy [23] was used to help manipulate image data. To create the user interface for the editor, the simple TkInter package [24] (available as a basic default option for UI creation in Python) was used. While making a visually appealing interface in TkInter is extremely challenging, this was not a requirement of the editor application (intended to be used only by a curator of diagram sets available in the mobile application, and not potential users of the app itself), and was not a focus during development. Programming a TkInter interface, however, was simple, due to the limited nature of the package. As a basic user interface was sufficient for this part of the project, the package was used instead of any potential alternatives.

2.4 Implementation Details - Smartphone Application

The primary goal of the smartphone application was to download and display packages of annotated diagrams. To achieve this, a basic skeleton viewer was created in HTML and CSS, capable of rendering layered images. Code to style labels which were to be added as an overlay to diagrams (where appropriate) was created using CSS, to allow future populating of annotations. A user interface, styled using HTML and CSS, consisting of buttons and menus to navigate easily between layers or pages within a package was implemented in JavaScript. This provided the additional function of displaying which layer or page the user was currently viewing to provide signposting and ease navigation. Touch gestures were added using the Hammer [25] JavaScript package, providing shortcuts to quickly navigate between layers, for added convenience.

The appearance of the app was kept clean and minimalistic, so as not to distract from the material presented. In general, the interface was designed to offer the simplest user experience possible,

attempting to avoid the potential for confusion, error, or other difficulties when using the app. Due to the basic nature of the project requirements, this was not a particularly challenging goal to achieve, as the functionality the app needed to expose to the user was minimal.

An issue that arose during the development of the mobile app interface was one of input latency. There existed a delay upon tapping the screen, before the expected action occurred, which made the app feel unresponsive and clunky. This was present throughout all versions of the app (the initial native/wrapped WebView version, the PhoneGap version, and the Cordova version) and upon investigation was found to stem from a design decision by the creators of the basic Android browser. On older smartphone models, without modern software installed, a 300 millisecond delay exists between tapping the screen and the action appearing to register within the native browser [26], upon which the WebView, PhoneGap, and Cordova are all based. The reason for the existence of this delay is to capture “pinch-zoom” touch gestures correctly, which are relevant in general web browsing. Supporting these gesture was not, however, required for the planned implementation of the revision app. In general, this delay no longer exists in newer software, but to offer maximum compatibility to older smartphone models, a solution was required that would provide a smooth user experience as many devices as possible. As recommended by Google Developers [26], the JavaScript package FastClick [27] was used to circumvent this issue, providing immediate response to user input, and therefore a much snappier interface.

Having created a working user interface for the smartphone app, a method of loading content to be viewed was required. As previously mentioned, the initial solution (due to issues with file storage plugins) was to bake the image and data files required into the base app, but this was completely unscalable and failed to meet the requirements of the project. An alternative solution was designed as a replacement, using a basic repository system where the data packages available to load into the app could be stored and maintained. These packages could then be downloaded for offline viewing at the user’s discretion. This allowed theoretically unlimited packages to be available (storage permitting), and complete user control over what content they did or did not want to have available to revise within the app.

The final implementation of this repository system had, however, an underlying flaw, in that diagram images downloaded from the repository within the app were saved and encoded as Base64 [28] strings in plain text files. This was due to an problem which remains unsolved that prevented images from being saved correctly as true image files. The downside of the workaround, with a string-based encoding, is that the resulting files take up slightly more storage space (the plain text files are around 1–1.3 times the size of their corresponding images) than the true image originals. This issue could almost certainly be fixed, given enough time, but was not a critical problem given the relatively small overall size of the diagram packages and was therefore left unresolved.

Due to the nature of the repository system, the app did not contain any revision material upon download. To guide the user through the process of installing a package of material, a small tutorial provided instruction on using the system. Another tutorial informed the user about the

basic navigation controls of the app. Both tutorials display only once, but can be revisited by the use of a reset option. Even without the tutorials, the app was designed to be simple and intuitive enough that installing and navigating through a package should not present any difficulty for the user.

2.5 Implementation Details - Package Creator and Image Editor Application

To fulfill the requirement of converting a large collection of high-resolution scanned diagrams into a format ready to be displayed within the mobile app, a Python-based image editing and annotating program was created. This application was similar, in some ways, to general purpose image-editing programs, but contained only a specific set of feature and tools that would be required to process the scans.

While this program greatly streamlined the editing of the images, this aspect alone did not provide a huge advantage over using a traditional image-editing application to achieve the same purpose. The key feature implemented here was the ability to group images into titled “pages” (analogous to a set of sleeved acetate sheets with a paper backing in the folder), by layering individual diagram “slides” (analogous to a single acetate or paper sheet) with their relevant annotations and associated information to create a layered set of diagrams complete with all the appropriate information and labels. The editor then allowed multiple pages to be collated into a completed package (see Figure 2-1 below), the virtual folder of diagrams, and this package exported in a pre-defined format which could be read by the mobile app for display.

To process the images and prepare them for eventual export, various specialized tools were created using the OpenCV library. Additionally, a set of preprocessing steps was implemented to automatically (or as close to automatically as possible) remove the background from the image, leaving only transparent space and the coloured lines of the diagram.

This preprocessing stage followed a few relatively simple steps. Firstly, a Gaussian blur [29] was applied to any image loaded. The resultant image was then converted to grayscale, and thresholded using a “binary inverse” threshold to produce a binary mask: any pixel with a value less than the threshold converted to white, otherwise converted to black. This mask was then recombined with the original image using a binary AND, and an alpha channel added to produce a final image, generally with all of the diagram visible, and the background removed.

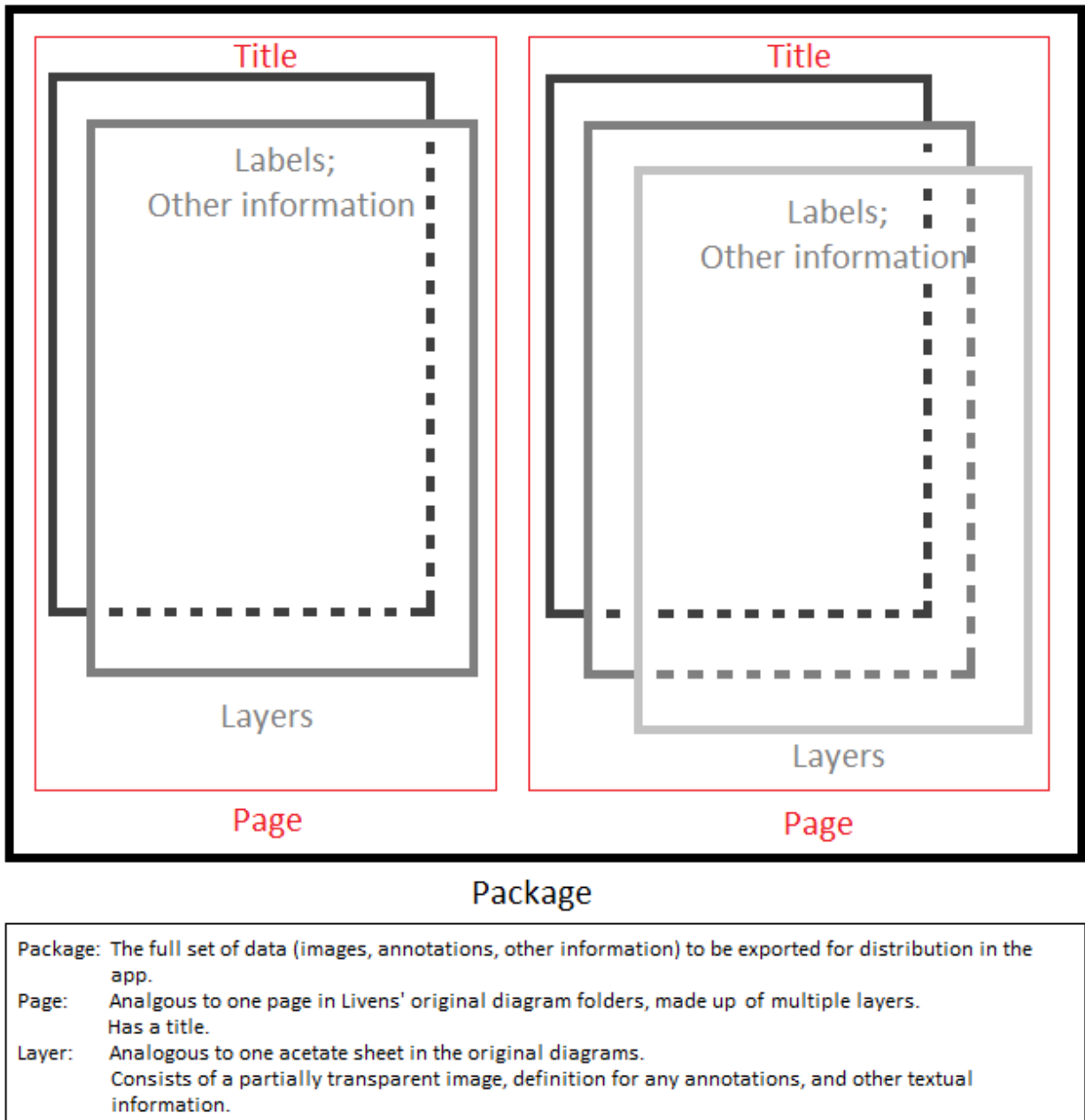


Figure 2-1: Editor-created package composition

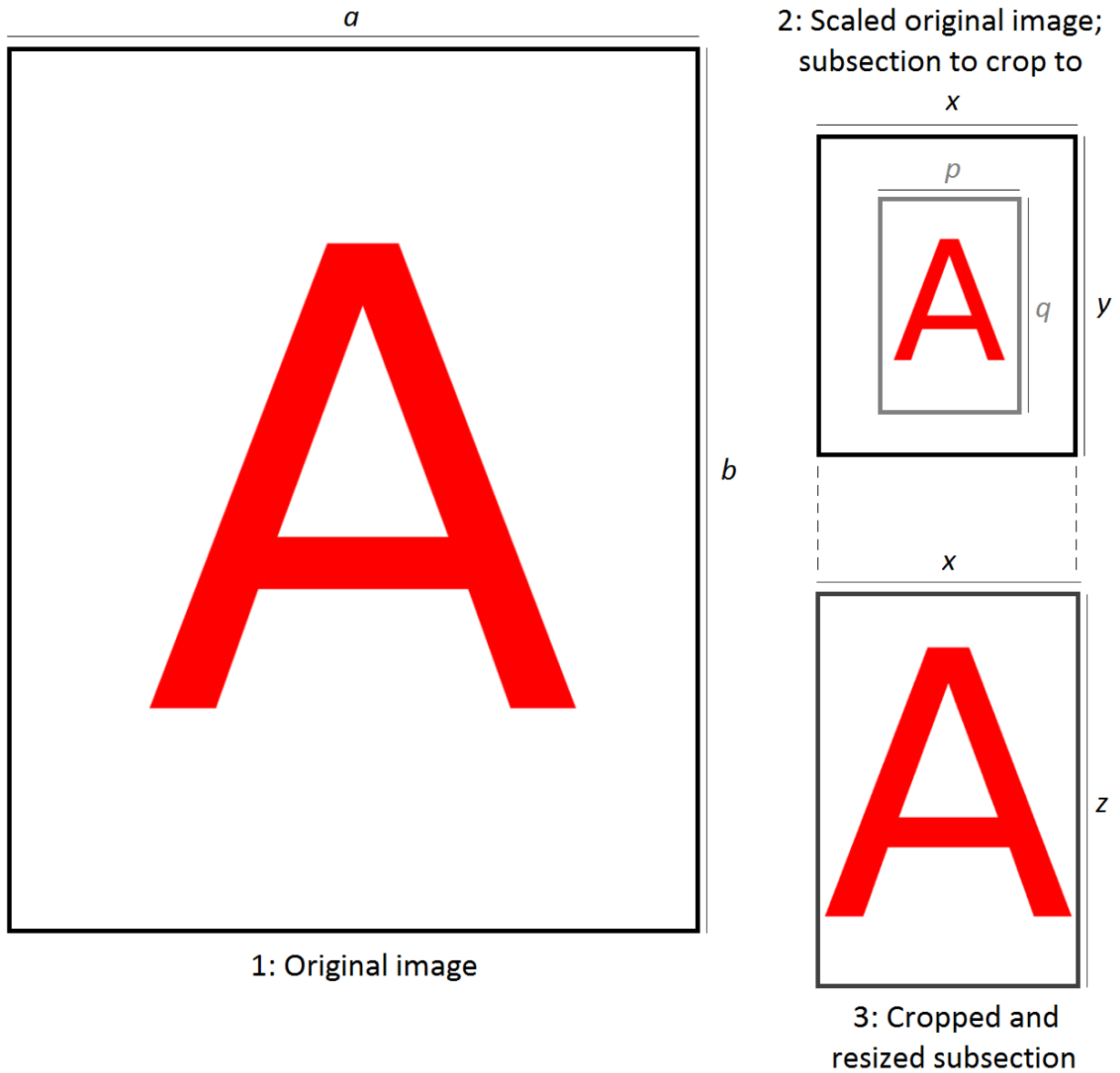
Initially, a static threshold value was used to achieve this effect, but this was insufficient for some images. Scanned from acetate sheets, these images could have areas with very light colouring, faint regions, or faded lines. Adjusting the static value to accommodate these images resulted in poor performance when processing parts of, or all of, the background in some of the scanned paper sheets (due to the off-white/beige colour of paper used). The solution implemented was the addition of a dynamic threshold value, controlled via an adjustable slider, set to a reasonable default value (the previously chosen best-guess static value). This base value catered for the

majority of images, but in any edge case where manual tweaking was required, a simple method existed to do so. This allowed for quick and easy preprocessing of any scanned image.

Of the tools created, the first implemented was a cropping tool, which could be used to remove excess whitespace from the border of the scanned image, or pick a specific diagram from a scan with multiple diagrams drawn on one sheet. The user interface for this involved simply selecting two points on the image (the diagonals of a rectangle), thus creating a bounded region which could then be cropped to. A predetermined horizontal resolution for images viewed within the smartphone app having been defined, (allowing the vertical resolution to be dynamic, with the smartphone app capable of vertical scrolling as needed) the editor would initially scale the images loaded to a set width, providing a working copy that wo. Upon cropping an image, the cropped subsection would then need to be enlarged to re-fit this pre-defined width (see Figure 2-2 below).

Unfortunately, doing these resizing steps in the simplest manner (scaling the original image down to the correct width, then enlarging a subsection to this width when cropping) resulted in a sharp deterioration of image quality on each crop. Given, then, the relatively high resolution of the original images compared to the final sized image, the solution here was simple: maintain a copy of the original image in memory, and upon cropping use the coordinates of the desired image subsection to reference against this original image. The chosen section of the original image (which would still be the highest possible quality, not having yet been downsampled) could then be scaled to the appropriate width and used to overwrite the working copy. This added some complexity: if multiple consecutive crops took place, an offset would need to be calculated each time to ensure the correct section of the original image was used.

Next, an eraser tool was required, to cut out sections of the diagram containing only text, or remove any persistent artefacts or other flaws in the source material after the preprocessing stage. This was implemented as a simple tool using a complex-shape drawing function available in the OpenCV library. This tool allowed an unlimited (within realistic computational limits) number of points to be selected, each one adding a vertex to an area that was to be erased. The area would then be filled with transparency, the background for the image, effectively erasing that section. The image subject to modification here was the original image (as described in the crop tool description), rather than the scaled and potentially cropped working copy of the image. Upon an erase being committed, the working image would therefore need to be replaced with a new cropped and resized copy of this modified original image, to produce the correct result. This method, modifying the base image and refreshing the working copy, allowed consistency to be maintained between multiple crop operations when an erase operation had taken place. The alternative, only modifying the working copy, would result in any erased sections of the image regenerating upon a later crop operation taking place due to the implementation of the crop tool.



a : original image width
 b : original image height
 x : predefined width to be maintained
 y : height scaled to match $x:y$ ratio to $a:b$ ratio
 p : width of area to crop to
 q : height of area to crop to
 z : with p scaled to maintain x , q is scaled at the same ratio, producing z , the height of the resized cropped area

Figure 2-2: Crop tool: method used to maintain a predefined width; steps numbered so final result is "3: Cropped and resized subsection"

The overlap of the tools here added complexity to both, and care had to be taken to ensure interactions between the two functioned as expected. During the development of these two tools, an undo-redo system was simultaneously implemented, to provide a standard safeguard against user error. This turned out to be a challenging feature to implement, for numerous reasons. The basic logic and functionality of the system took careful consideration to properly implement, given the aforementioned interactions between tools, and attempting to minimize computation done (even at this early point in development, the application was performing slowly when carrying out certain tasks, the burden of processing high-resolution scans each time a tool was used defeating the hopes of a relatively real-time interface).

At this stage in development, a worrisome issue emerged. After performing a short series of operations, the application would crash, reporting a 'memory limit exceeded' error. This stemmed from the way the high-resolution scans were being stored in memory when the program was running. While these scans, stored on disk as simple PNG files, were rarely more than a megabyte in size, the unpacked four-channel colour arrays being maintained in memory were far larger. Members of the first scan set, being used for initial testing, produced roughly nine million array entries when unpacked: a not insignificant amount of data, roughly thirty-six megabytes per image.

While individually these arrays would not be enough to cause memory issues, the storage of operation history for the erase tool, used by the undo-redo system, had been implemented in a completely naïve manner: a snapshot of the original image was saved both prior to the erase operation and following the erase operation. Either of these saved snapshots could then be restored on an undo or redo. As these copies were based on the original high-resolution scan kept in memory, each was tens of megabytes in size. A single erase operation, storing two copies of an image, would therefore quickly devour available memory.

In addition to this, a temporary memory overhead was incurred during any erase operation. An amount of memory of similar magnitude to that of a loaded image, tens of megabytes, had to be available in order for an erase operation to complete successfully. These costs quickly accumulated during attempted modification of an image, especially if multiple smaller regions required erasing, and appeared to be the source of the memory errors.

As a temporary solution to the problem, the undo-redo system was disabled, awaiting future re-implementation, and development continued. Time was taken, however, to attempt to optimize both the rendering system and the erase tool, to minimize memory overheads, and increase execution speed. For rendering, this involved developing a more intelligent approach, preventing multiple successive rendering operations from repeating unnecessary work. To achieve this, a cached copy of the currently displayed layer was maintained, which could then be marked as "dirty" and only updated as necessary, greatly reducing the rendering computation required in some situations. Optimization of the erase tool was simpler, and involved some low-level adjustment to the code to minimize data copying operations.

With the layered format of the pages, a tool to manipulate how the images stacked on top of one another was sorely needed, to compensate for offsets generated by the original scanning process. This tool was implemented and a user interface created to allow a layer to be displaced by clicking and dragging. This process could be repeated independently per layer, providing a simple method to line up any number of layers. This was implemented as a simple dynamic offset for each layer (other than the first layer, which was maintained as an immovable base) allowing the layers to be positioned as needed. Care had to be taken to ensure proper interaction with the cropping tool, and that when a cropped layer was moved any region that would otherwise have been “out of bounds” of the cropped zone was loaded and made visible (see Figure 2-3 below).

Additionally, an option to switch between viewing the entire image stack and just the currently selected image was added, to make editing diagrams with a high number of layers easier. This had the additional benefit of speeding up rendering when only working on a single layer, saving the processing time that would have been spent on any other visible layers.

At this stage in development, the basic functionality of modifying and preparing images for a single page was largely complete, lack of implementation of a working undo-redo process aside. Development efforts were then switched to implementing a system for annotating the layers, to create a solution capable of adding titles, text labels, and other information.

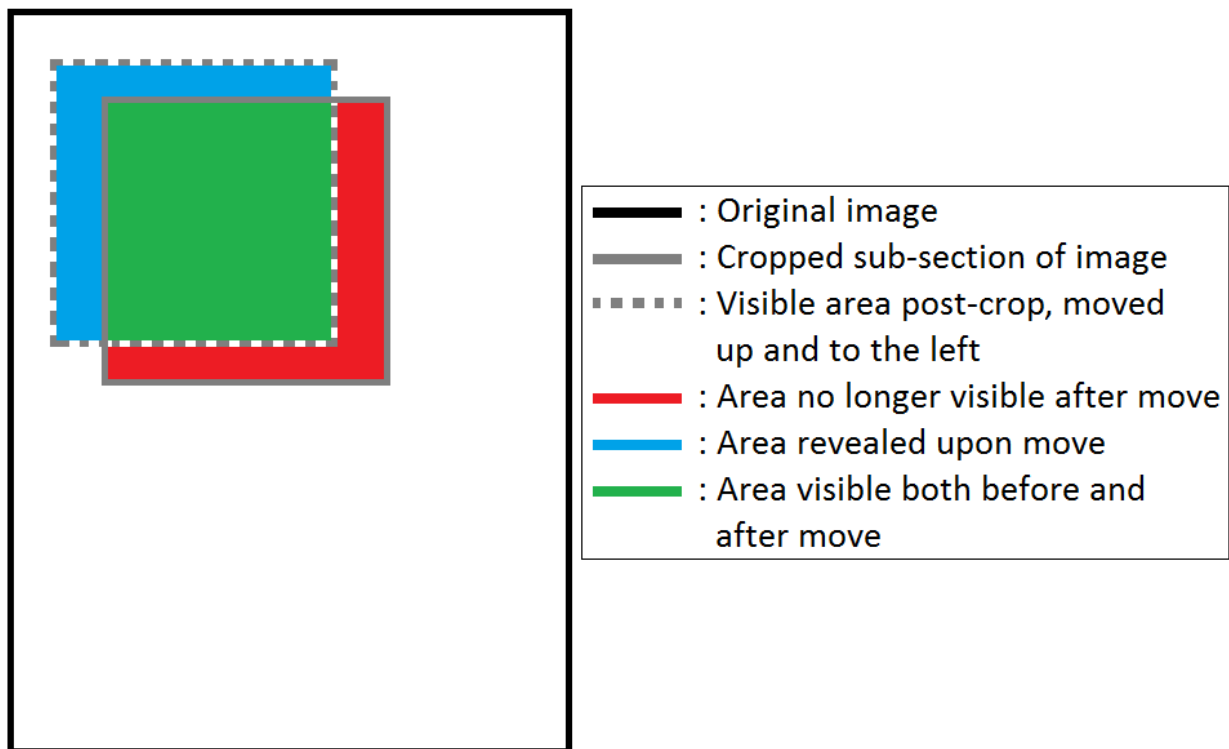


Figure 2-3: Crop and move-layer tool interaction

In retrospect, the best solution for labelling the diagrams would have involved web application code, thus allowing the user to see the final appearance of the label as it would render on the mobile device. Instead, the labelling method was implemented as a component of the main Python-based editor application. This had the advantage of simplifying development, having all of the package preparation available within one piece of software. The downside to this was that accurate and readable labelling was harder to produce, due to differences between rendering styles of the OpenCV library and a web rendering engine.

Implementing a system to perfectly recreate the web application rendering of a label in OpenCV would have been extremely time consuming, and perhaps one could never hope to accurately emulate browser rendering using the tools available in the Python library. Instead, a simple text field and coordinate point selector allowed the user to input the text and location of a label, respectively. While this basic solution was generally sufficient for purpose, being unable to view the resultant diagram and image combination that would be generated in the web application created situations in which placed labels could overlap, be positioned inaccurately, or simply not fit on the image. Due to the density of labels on some of the more complex diagrams, an additional feature was implemented later into development, to allow a label to be flipped horizontally, enabling better utilization of the available space over each image. A small modification to the smartphone app was additionally required to utilize this.

With the editor now able to annotate diagrams, an exporting tool was implemented, which dumped the finalized images to disk, with a JSON file containing any related textual information, organized in a specific directory structure. This would later be expanded to export multiple pages, using the same general principal.

To allow for multiple pages, however, the memory problem that had remained largely unaddressed during the earlier stages of development had to be solved. Even with the naïve undo-redo system disabled, the memory requirements present in loading multiple sets of high-resolution images in preparation for editing was causing the same memory problems previously encountered.

The obvious solution here was to only have the sub-set of images within the current page loaded, keeping the rest cached on disk to be swapped into memory as required. Due to the underlying NumPy-based array storage utilized in OpenCV, this solution was relatively simple to implement. The NumPy library included various methods for dumping array data to disk, so the problem was solved by simply writing the data of inactive pages locally and reloading this data when required. This method incurred a slight delay upon page transitions within the editor, but given the logically disconnected nature of these unrelated pages the small delay was acceptable.

To implement a similar solution by instead exporting the arrays as (for example, PNG format) images and re-loading them using OpenCV itself was considered, but as the OpenCV image loader discarded the alpha channel of any image (unacceptable behaviour due to the

transparency-heavy nature of the images to enable layering) this was not a valid option. An additional problem with converting the arrays to images for storage (and back again when reloading) lies in the potential degradation of quality due to compression losses when converting the images to and from common formats.

3 Evaluation and Feedback

3.1 Evaluation of App Against Published Literature

The app was designed to conform to the *Six Principles of Effective e-Learning* [8] (discussed in Section 1.2) where possible, and how this was achieved is discussed below.

“The multimedia principle - adding graphics to words can improve learning”:

In the app this took the form of labelled diagrams, naturally combining text with images. It should be noted that, for revision purposes, having just enough information presented and no redundant information shared between the multimedia elements is preferred. Due to the nature of the source diagrams being Livens' own revision material, there was little redundancy in the content presented in the app.

“The contiguity principle - placing text near graphics improves learning”:

As with the implementation of the previous principle, this was provided through the use of labelled diagrams. Due to the small physical screen space available on mobile devices, it was not possible to fit all of the information relevant to each diagram into an overlay or label on the diagram, and instead any extra text was made visible by tapping the diagram. An unfortunate consequence of the platform, but perhaps unavoidable.

“The modality principle - explaining graphics with audio improves learning”:

There was no audio feature implemented in the app, so this principle was not followed. This was partly due to time constraints, partly due to a lack of desire to have it implemented, and perhaps more importantly due to the intended use of the app. To have audio accompanying the diagrams would have prevented a user from quickly navigating through the app to pick out specific sections to revise, instead requiring them to wait and listen to an audio recording for each section. If it was implemented as an optional feature, it may have been of some very situational use, but would likely just be disabled by users (and would even need to be disabled by default to provide a better user experience to the majority of users) and would have been a wasted effort.

There is an argument, however, for providing an audio recording for the pronunciation of various anatomical names, which are often complex. This is a feature that could potentially be implemented if development on the app were to continue.

“The redundancy principle - explaining graphics with audio and redundant text can hurt learning”:

The app conformed to this principle, again by virtue of the pre-created source material; there was little to no redundant information present.

“The coherence principle - using gratuitous visuals, text, and sounds can hurt learning”:

The app had no decorative elements, having no place in a project of this type. Additionally, the storage space required by the app when installed on a smartphone was to be kept to a minimum, further preventing any decorative elements from being added.

Any textual information presented was kept to a bare minimum, in the style of Livens’ original notes. These notes were Livens’ own distilled version of the material she would need to revise, and therefore no extra text was present beyond the required information in a basic and pure form.

“The personalization principle - Use conversational tone and pedagogical agents to increase learning”:

There was a lack of specific tone used in the text of the app, as any text was based on a tweaked and minimized copy of the notes Livens had provided (as described above).

3.2 Evaluation of Goals Against Specification

To evaluate the app against the specification, Livens, who proposed the project, was provided with a copy to test. As the app was essentially an implementation of her vision, this seemed the most reasonable way to carry out an initial assessment. Livens was then interviewed, this interview consisted of a set of specific questions to get Livens’ opinion of the final app compared to her expectations, followed by a subset of the questions used in a survey (discussed in Section 3.3 and Section 3.4, and conducted simultaneously to Livens own testing) to gather more general feedback about the look, feel, and performance of the app. A transcription of the full interview is available in the appendix [A2], where Livens’ response to the app was strongly positive. Overall, she was satisfied with the implementation of the app, the only complaints being minor issues with the presentation of some information.

The first issue she raised, “[for] any individual numbers that come up it would be better if the information was individual and not as a list,” was in reference to a choice made to combine multiple identical labels into a list to save screen space (see Figure 3-1 below).

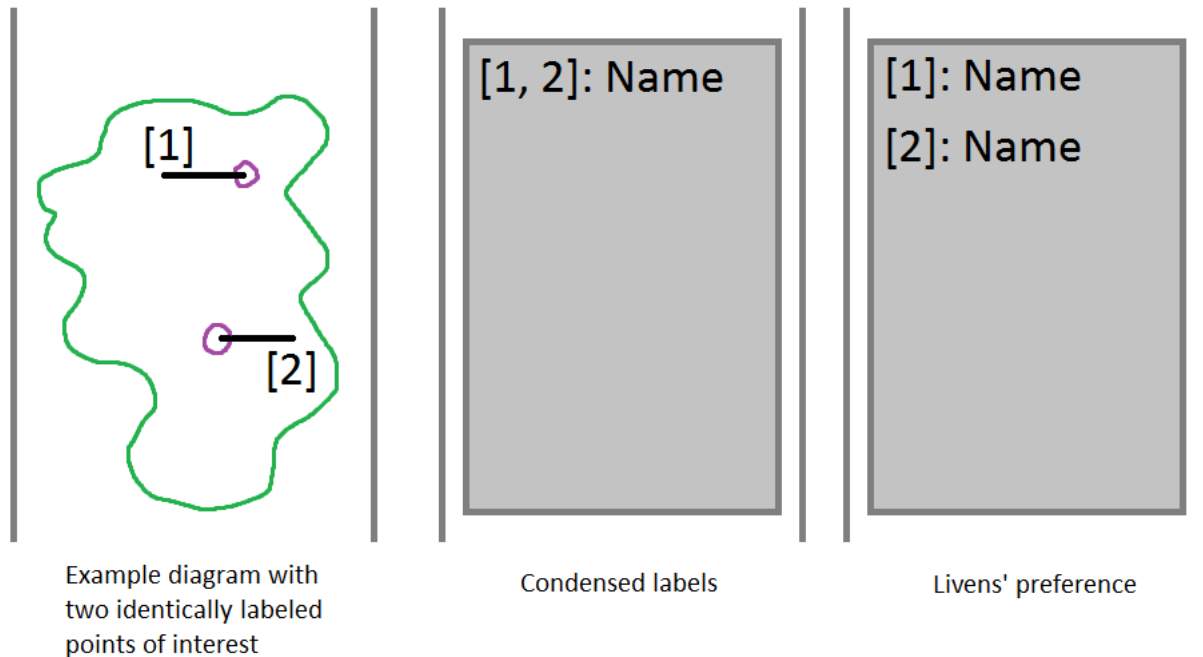


Figure 3-1: Information density versus clarity: originally the “condensed” style was used; this was switched to match Livens’ preference

In retrospect, condensing the labels in this manner made it harder to instantly see the name of a labelled feature, adding an extra cognitive step which could impede the revision process, if only slightly. Nonetheless, this goes against the general design principles for the user interface of the app, and so was later corrected. Unfortunately, this was not considered during development, and therefore the version with “condensed” labels was provided to testers, perhaps negatively affecting the user experience of the test group during the feedback collection period.

The second issue Livens raised, “Some of the pictures (very few) may have benefited with being done in more layers,” was a consequence of the minimal coordination between Livens and me during development. The app was an attempt to produce a replica of the diagram source material that Livens provided, and therefore the number of layers was directly related to the number she had used when creating the original hand-drawn diagram set. As Livens noted, this did not necessarily translate perfectly to a mobile app with far less viewable area available compared to the A4-size originals.

Ideally, the development process for the content of the app would have involved Livens herself producing a custom set of diagrams, to allow precise specification of layering (among other features) and ensuring the best possible final product. This would have placed a great burden on Livens, however, and overall was not deemed necessary. Alternatively, were Livens and I fully dedicated to working on the app, or the deadline for completion more relaxed, a complete review of the content could have been carried out to catch and fix such issues. Overpopulated layers could have been split into multiple parts, Livens’ guidance ensuring additional clarity without any

loss of information, sense, or meaning; something I could not have hoped to guarantee, having no prior knowledge or understanding of the material.

Beyond these two issues, Livens suggested a “self test function” which could be potentially be added as an extra feature. This was something (as previously mentioned) discussed at the outset of the project: a component that could be implemented beyond the requirements of the specification, were ample time available. Unfortunately, this idea was not realized, as the base specification took as long, if not longer, to develop than originally anticipated.

3.3 Overview of Feedback

Feedback on the app was collected by means of a short web survey, in which testers were presented with a series of statements and asked to define their agreement with the statements on a scale from “strongly disagree” to “strongly agree.” Additionally, comments could be made on any statement, if a tester wished to give a more full opinion, specify the reasoning behind their choice, or provide further detail that the survey format would otherwise not permit. Ideas for any additional features a tester wished to see implemented were gathered, and lastly each tester was asked if they would consider using an app of this style, or would recommend it to a fellow student. The full list of survey questions is visible in the appendix [A3].

The results from the survey, specifically the statement agreement questions, are shown in Table 3-1 below. The “Total agreement” column simply shows the percentage of responses that were “Agree” or “Strongly agree” for a given question, whereas the “Rating” column is a weighted average based on weights assigned to each potential response [30]. The weights were as follows:

Strongly disagree: 1

Disagree: 2

Neither agree nor disagree: 3

Agree: 4

Strongly agree: 5

The “Rating” is then calculated as follows, where:

w = weight of answer choice

x = response count for answer choice

t = total responses

$$\frac{w_1x_1 + w_2x_2 + \dots + w_nx_n}{t}$$

In general, the survey response was reasonably positive, the overall average of the “general feedback” ratings being just over 3.8 (not including the ‘negative’ Statement 13), falling close to the weighted value assigned to “Agree.” This, however, implies a lot of room for improvement, and the specific issues highlighted by the testers will be discussed in Section 3.4, along with any meaningful comments left by the testers. In addition, the “Total agreement” column highlights some worrying issues with the quality of diagram labelling (which will be explored in detail), and the appearance of the app.

Table 3-1: Survey responses (without comments)

Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree	Total agreement	Rating
General feedback:						
<i>1: The material/content was useful for revision purposes</i>						
0	0	1	2	2	80.00%	4.2
<i>2: The material/content was presented in an easily readable way</i>						
0	1	0	3	1	80.00%	3.8
<i>3: The material/content was organized in an easily navigable way</i>						
0	1	0	3	1	80.00%	3.8
<i>4: The material/content was partitioned into sensibly sized chunks</i>						
0	0	2	0	3	60.00%	4.2
<i>5: The accuracy of the material/content was sufficient</i>						
0	0	0	4	1	100.00%	4.2
<i>6: The quality of the diagrams (accuracy aside) was sufficient</i>						
0	0	1	4	0	80.00%	3.8
<i>7: The quality of labeling of the diagrams (accuracy aside) was sufficient</i>						
0	2	1	2	0	40.00%	3
<i>8: The app was aesthetically pleasing</i>						
0	2	2	0	1	20.00%	3
<i>9: The app was intuitive (sensibly organized; easy to learn; actions performed did not have surprising outcomes)</i>						
0	0	1	2	2	80.00%	4.2
<i>10: The app was easy to use (once familiar)</i>						
0	0	0	4	1	100.00%	4.2
<i>11: The app was pleasant to use</i>						
0	1	1	2	1	60.00%	3.6
<i>12: The app did not punish user error; the app made it easy to recover from user error (e.g. pressing the wrong button did not cause anything disastrous to happen)</i>						
0	0	1	2	2	80.00%	4.2
<i>13: The app should have been more dynamic/customizable</i>						
0	1	1	2	1	60.00%	3.6
<i>14: The performance of the app was sufficient (it did not feel slow or sluggish during use; it did not freeze)</i>						

0	1	0	3	1	80.00%	3.8
Personal feedback:						
<i>17: I would use an app of this kind (not necessarily this app, but one with a similar function and design)</i>						
0	0	1	2	2	80.00%	4.2
<i>18: I would recommend an app of this kind (not necessarily this app, but one with a similar function and design) to a fellow student</i>						
0	0	0	3	2	100.00%	4.4

3.4 Detailed Breakdown of Feedback

Here, the feedback for each statement is discussed, and any meaningful comments related to that statement are explored.

1: The material/content was useful for revision purposes

A general, and perhaps unsurprising, level of agreement was present here, showing that the material would, in fact, be useful in a revision context.

The singular “neither agree nor disagree” response was made by a tester who commented that they preferred their revision material in a different style, and so would not generally find such an app useful.

2: The material/content was presented in an easily readable way

A reasonably positive response, but problems were identified by the testers here in relation to the presentation of diagrams and labels.

One tester commented that the ability to toggle the display of labels annotating a layer, to only view the diagram underneath, would be beneficial; this was an excellent suggestion and a feature never considered during development. This feature was therefore implemented in response to this feedback.

Another tester commented that colour-coding the labels would greatly enhance the clarity of some annotated diagrams. This was potentially the greatest oversight made during development: the original diagrams had coloured labels, and this information was stripped away in the conversion process, for no practical reason. Correcting this most blatant problem would require a rework of the app itself, a pass over the content currently available in the app (produced by the editor application) to add colour information, and a redesign of the editor application to incorporate a means to define colour per label. As such, implementing this feature is no small task, and was therefore left as further work to be completed if the project was to see continued development.

3: The material/content was organized in an easily navigable way

Again, a reasonably positive response here, the single “disagree” response coming from a tester who encountered technical issues in running the app. As the survey was web-based and anonymized, there was, unfortunately, no direct way to follow up on this issue and discover the exact problem. In order to properly evaluate (and hopefully then solve) technical problems, an in-app method of collecting error data, feedback, phone specification, user’s Android version, and any other relevant data would likely be required. This is certainly an area that would warrant further investigation and development were work on the project to continue. Additionally, a larger test group could have helped to identify similar issues, and perhaps assist in pinpointing their causes.

One tester commented that the interface was inconsistent, and the text on one of the buttons unhelpful. This was an accurate assessment, and the problem has since been fixed.

4: The material/content was partitioned into sensibly sized chunks

An overall positive rating from the responses here, but some problems were identified. One tester, again, experienced technical difficulties, unfortunately the same situation described earlier prevented these from being resolved. Another tester commented that, similar to Livens’ opinion from the interview in Section 3.2, some layers contained far too much content and would have been better split into multiple less-populated layers. The challenges involved in solving this issue are described in Section 3.2, but again, this problem remains unresolved and therefore would be important to tackle if development were to continue.

Another comment, related to this statement but given in response to another, concerned one tester’s preference for diagrams separated into layers organized into distinct categories; for example a layer for bone, then a layer for ligaments. This would seem a reasonable expectation, but again, the content available in the app was defined by Livens’ original diagrams: discussion and coordination would be required, alongside a review of the material presented, to decide whether diagrams could be split into layers in a more logical fashion. If the aforementioned layer subdivision changes were to be made as part of future development, simultaneously investigating this issue would seem sensible.

5: The accuracy of the material/content was sufficient

A good level of agreement here, and no specific issues were raised by the testers. It is therefore unclear if the number of “agree” responses as compared to “strongly agree” was purely because the testers did not evaluate the complete set of content available in the app (which, admittedly, would have been very time consuming) if some small errors were found, or if some other problem was present. Further collection of feedback could be warranted here to ascertain the exact nature of any issues.

6: The quality of the diagrams (accuracy aside) was sufficient

Again, a good level of agreement here, with no specific comments from the testers as to what could be improved. Perhaps the nature of the content in the app, being electronic versions of

Livens' hand-drawn images, meant the quality was insufficient for the testers to "strongly agree" with this statement.

The single "neither agree nor disagree" response was, again, from the tester experiencing technical difficulties, as previously discussed.

7: The quality of labeling of the diagrams (accuracy aside) was sufficient

Here we have the first set of responses which are generally negative. The testers had issues with the style of the labels, the lack of colour (as mentioned earlier), and the density of the labels on some diagrams causing overlaps which made reading them difficult.

These problems largely stem from the choice, made relatively early in development, to label the diagrams within the image editing program, and not in a separate program powered by the same web rendering engine used in the final app (as described in Section 2.5). The feedback here makes it clear that this was a poor choice, and the lack of quality labelling hampers the user experience and usability, reducing the value of the app significantly. Developing a standalone web application to label the diagrams would require a large amount of development time, but is likely the only reasonable way to alleviate these issues. If work on the project were to continue, creating this labelling tool would be a high priority task.

8: The app was aesthetically pleasing

Another relatively negative set of responses here, which was not unexpected. The app is stylistically simple, perhaps having seen less beautification than the testers would expect from a modern app. One tester did comment, however, that while not particularly aesthetically pleasing, the utilitarian style did not generate any issues during using of the app. There is no reason the app could not be more visually appealing, development time was simply focussed elsewhere, and improvement could be made here fairly easily given a reasonable time investment. In general, this is another area for potential future development.

9: The app was intuitive (sensibly organized; easy to learn; actions performed did not have surprising outcomes)

A largely positive response here, the single "neither agree nor disagree" response from the tester experiencing technical difficulties. No specific comments were made, but any improvements to the user interface, appearance-only or otherwise, would likely increase tester satisfaction in this area.

10: The app was easy to use (once familiar)

Strong agreement here, again without specific comment from the testers. As discussed in evaluation of the previous statement, general tweaks to the interface could potentially draw stronger agreement from the testers.

11: The app was pleasant to use

A fairly mixed response from the testers here, with no specific issues commented on. Presumably, since all but one tester reported that the app performed sufficiently well, any issues found here

were more to do with general look and feel of the interface. Again, various aspects of this could be tweaked or improved were development to continue.

12: The app did not punish user error; the app made it easy to recover from user error (e.g. pressing the wrong button did not cause anything disastrous to happen)

A positive response was given to this statement, with no specific comments. This can perhaps be explained by the fact that the app had no real state-changing actions (although would automatically save the page the user was last viewing, to re-load it when they next open the app) so generally there were limited ways in which the user could negatively affect their own experience. Perhaps the most “disastrous” mistake the user could make would be accidentally resetting the app, but so long as they had internet connectivity, they could simply re-download any lost content.

If a user were to navigate away from the page they intended to be viewing, however, there was not, for example, any way for them to backtrack through a history of the pages they had viewed, which could potentially be implemented as a feature during future development. Perhaps implemented similarly to the style of a web browser’s back-forward page navigation, this could improve the user experience by compensating for potential user error.

13: The app should have been more dynamic/customizable

In the only statement where an agreement response a negative one, testers generally wanted more customizability. Given the completely static nature of the app interface, this is perhaps unsurprising. The single “disagree” response came with a comment stating the app had no need to be more customizable, and that implementing other features or fixes would be more meaningful improvements. Nonetheless, certain options (for example, changing text sizes, a ‘night reading’ mode, changing zoom levels) could add a lot to the user experience, and could be a target for further development.

14: The performance of the app was sufficient (it did not feel slow or sluggish during use; it did not freeze)

A generally positive response was had here, with the tester experiencing technical difficulties providing the singular “disagree” response. One tester commented that the app was slow to load the first diagram, but beyond that performed well. This could perhaps be the reason the testers did not “strongly agree” with the statement.

Besides the lackluster startup time, which was essentially unavoidable, one change that could be implemented (not truly improving performance, but rather improving user-perceived performance) would be to add transition effects when swiping between layers. Currently, the app waits until the user completes the swiping motion before switching layer, showing no visible feedback for the duration of the swipe. This is not a particularly pleasant effect; a time period exists where the user has no indication of whether or not the app is responding to their input. This feedback effect could be a helpful feature to add if development on the project were to continue.

15: Are there any features that could be added that would improve the app?

The first of two open-ended questions, testers had the opportunity to suggest any additional features they would have liked to see in the app. Besides various improvements mentioned previously, and suggestions to implement material specific to courses testers were studying, there were some interesting suggestions.

Firstly, one tester suggested the content would be better organized into sections, with per-section quizzes to test knowledge acting as revision checkpoints to ensure a user had thoroughly revised the contents of one section before moving to the next. Both parts of this proposal, sectioning and quizzes, appear to be reasonable suggestions that could potentially be implemented in the future.

On the concept of sectioning, Livens' original material did not have any granular division, simply separate folders per animal. Breaking this mass of content into logically organized sections could certainly help revision efforts, as currently the diagrams are listed, by name, as one full set, which could be overwhelming and difficult to navigate.

The addition of revision quizzes then carried out over these smaller sets seems logical, allowing the user to focus on and revise a sensibly sized chunk of material without having to wade through the entire diagram set to find some piece of information they had forgotten or misremembered. As previously mentioned, the idea of implementing some form of revision quiz functionality was discussed, but unfortunately never realized, and refining the concept (were it to be eventually implemented) with the use of sectioning seems a sound suggestion.

In general, a revision quiz feature would appear to be in high demand, and some implementation of this should therefore be prioritised, if development were to continue.

Another feature suggested by a tester was the ability to view and compare two different pages side-by-side to allow easy comparisons between, for example, types of joint. While this seems like a conceptually useful feature, it would likely be hard to realize due to the size of the screen available on most smartphones. Were a version of the app to be made to target tablets, this could certainly be a useful feature, and could therefore be implemented.

16: Were there any problems (aside from any previously mentioned) experienced when using the app?

The second open-ended question, which elicited no response.

17: I would use an app of this kind (not necessarily this app, but one with a similar function and design)

A generally positive response to the idea of the project as a whole was an encouraging sight, with the singular "neither agree nor disagree" response from a tester who commented that they preferred to make and use their own style of revision material, and so would not have a use for such an app.

18: I would recommend an app of this kind (not necessarily this app, but one with a similar function and design) to a fellow student

Again, a positive reaction from the testers, which would again indicate that an app of this type has potential for success. No further comment was made by the testers.

3.5 Evaluation of Goals Against Feedback

Overall, the response from the testers was a largely positive one. Several problems were identified, the most critical being the poor labelling of diagrams, as well as lesser issues such as the lackluster look of the app. The testers, however, agreed that an app of this nature would be a useful revision tool, and one they would recommend to their peers, certainly a promising result (if perhaps an expected one, given the popularity of Livens' original diagram set).

Nonetheless, a significant number of problems with the final version of the app would need to be solved for it to be truly appealing and useful as a revision tool. Considerable time would have to be spent developing and improving both the smartphone app and the editor application; yet more time would need to be devoted to creating a web application capable of accurately labelling diagrams to replace the tool contained in the editor application. A review and refinement of the content available within the app would be necessary, and ideally the diagram set should be reworked to suit the medium, to provide the best end user experience.

Despite all this, the project showed potential, the concept demonstrated being desirable to the target audience. Were development to continue, the app could certainly be improved and refined to become an invaluable tool to veterinary students everywhere.

4 Conclusion

4.1 Overview of Unsolved Problems and Areas for Further Development

The following is a list of the issues and areas for further development presented throughout the report, in approximately the order they were first discussed.

Images should be saved as appropriately formatted image files within the smartphone app (Section 2.4)

This was a curious issue that halted development, whereby images downloaded from the repository system within the app failed to save as true PNG files, seemingly due to a character encoding issue. A proper solution could not be found within a reasonable time, so instead a workaround was implemented, to save images as a string format using a Base64 encoding. The downside of this workaround was a slightly higher storage space requirement for images saved within the app, compared to a proper encoding. While this had very little impact on the overall performance or functionality of the app (and so the workaround remains in the final version) further investigation is certainly warranted here, as it seems absurd that the most popular file-writing plugin for the Cordova platform would be unable to save image files correctly.

Some layers could be subdivided, and potentially re-organized to have a more logical division of information (Section 3.2, Section 3.4; Question 4)

The sub-division of layers is discussed in some detail in Section 3.2, but to summarize: the conversion process from A4 diagrams to smartphone-screen compatible diagrams increased the density of labelling, which, according to feedback, had the unfortunate effect of cluttering some diagrams beyond comfortable readability.

An effort to review the app content to identify affected pages and sub-divide these appropriately, with the assistance of someone familiar with the content to prevent any information from being mistakenly lost in the process, would likely be needed to correct this issue.

Additionally, while carrying out such a review, ensuring the division of information between layers was appropriate and logical would be a sensible strategy.

A revision quiz or self-test mode could be added (Section 3.2, Section 3.4; Question 15)

An additional feature suggested by both testers and Livens herself, some internal means of assessing a user's knowledge could provide a useful tool to aid revision. In addition to this, sectioning the diagram set into smaller related collections, which would then allow a narrower area of knowledge to be tested, could be beneficial.

Audio recordings for pronunciation of anatomical terminology could be added (Section 3.2)

A feature present in various anatomical reference apps, adding a recorded set of pronunciations for various terms and names throughout the app could be helpful, if not particularly impactful. Implementing this feature would require a large time investment from a narrator, as well as specialized recording equipment to ensure a high-quality set of recordings. Beyond this, adding such a feature would, unfortunately, increase the storage size requirements of the content available within the app, perhaps by an unacceptable amount. Further examination of the benefits of term narration against the costs would have to be made to determine whether or not such a feature should be implemented, but it would certainly be possible.

Missing colour information should be restored to labels (Section 3.4; Question 2)

A stunningly obvious feature that was completely overlooked in development, the usage of colour in labels in the smartphone app would greatly enhance the readability of many diagrams. The original diagrams used coloured labels throughout, but this information was lost upon converting the diagrams to a digital form. This is a clear oversight, and should be corrected immediately if development were to continue.

In-app error data and feedback could be collected to identify issues on untested hardware and software versions, or other more general issues (Section 3.4; Question 3)

As detailed in Section 3.4, one tester experienced technical problems when attempting to run the app, which severely crippled the app's functionality. While this may have been unavoidable (some user-side problem with the tester's smartphone), a mechanism to collect error data to attempt to identify and then resolve any such technical problems would be greatly beneficial.

The ability for a user to provide feedback within the app could be a similarly useful addition from a development standpoint, allowing any user to comment on their experience, potentially exposing problems or providing inspiration for further features. While there is no guarantee a user would provide useful or even meaningful feedback, having at least the option for them to do so would surely produce some benefits.

The method of labelling diagrams should be revised, as it produced poor results (Section 2.5, Section 3.4; Question 7)

An issue highlighted by testers, the labelling of the diagrams in the smartphone app was of poor quality. As discussed in Section 3.4, this was likely due to the nature of the method of diagram labelling. An extremely limited component of the desktop editor application, this labelling feature should be replaced by a web application capable of labelling and then rendering content as it would appear on a smartphone, allowing accurate labelling to take place.

The look and feel of the app could be improved (Section 3.4; Question 8)

As mentioned, the look of the user interface of the app was one of barebones simplicity, and this could certainly be improved. The web technologies powering the app provide practically limitless potential for customization here, were a developer skilled in graphic design to spend time polishing the interface.

Besides the interface, the diagrams within the app were of perhaps substandard visual quality. While adequate and accurate enough, these diagrams could be redrawn, preferably directly into an electronic format instead of relying on scanning and conversion which can result in a drop in quality.

A history of viewed pages, with the ability to navigate between them could be implemented (Section 3.4; Question 12)

A common feature in applications (smartphone, desktop, or otherwise) the ability for a user to quickly navigate through a history of the pages they have viewed would be a simple and helpful addition to the app, improving the user experience.

The app could have customization options (Section 3.4; Question 13)

While not a necessity, if implemented in an appropriate way as to not clutter the interface of the app and harm the user experience, options to allow a user to customize the appearance of the app could be beneficial. Simple options such as adjusting text size, overall zoom, or a night reading mode with a darker colour scheme could be helpful.

Transition effects when swiping could be added to provide missing feedback for user input (Section 3.4; Question 14)

A potentially missing feature that was not considered during development, and could help the user in learning and using the app, by providing clarity and a link between certain inputs and actions. Implemented poorly, however, it could be an annoyance, and potentially reduce performance: care

would have to be taken to ensure such a feature was an improvement to the user experience, and not a detriment. Perhaps the ability to disable the feature as a customization option could therefore provide some insurance, were the effects to be implemented.

4.2 Final Outcome of Project

Overall, the outcome of the project was one of modest success. Livens, who proposed the project, was satisfied with the final app that was produced; the app matched her vision of a conversion of the hand-drawn diagram set she had created into an app to aid revision for other vet students.

During the period of feedback collection, testers identified various issues with the app, some of which required only simple tweaks which were easily implemented, but many others would require a large investment of development time and continuation of the project to rectify.

The concept of the app on the whole, however, was well received: a promising result that should provide motivation either for further development of this project, or for other revision tools similar in nature to be produced.

4.3 Acknowledgments

Thanks go to Amy Livens, for firstly providing the motivation and materials for the project, and then testing and evaluating the final product, as well as orchestrating communication between myself and members of the Veterinary Medical School here at The University of Edinburgh.

Thanks go to Professor J Douglas Armstrong at The University of Edinburgh, who oversaw the project, providing support and advice throughout the duration, as well as encouragement and guidance where needed.

Thanks to Doctor Jessie Paterson, who was the driving force behind the assembly of the group of testers, a feat which made the collection of feedback possible.

Thanks to the anonymous testers themselves, surprising difficulty was encountered in finding willing candidates to test the app and provide feedback, making these contributions all the more valuable.

Finally, thanks to my family and friends; staff and peers at The University of Edinburgh; and anonymous strangers on the internet, who each contributed in their own way to make the project possible.

Bibliography

[1] Colorado State University. Veterinary Education Tools - Virtual Canine Anatomy.

<http://www.cvmb.colostate.edu/vetneuro/>, accessed 17th Feb 2016.

[2] Adobe AIR and Adobe Flash Player Team Blog. An Update on Flash Player and Android.

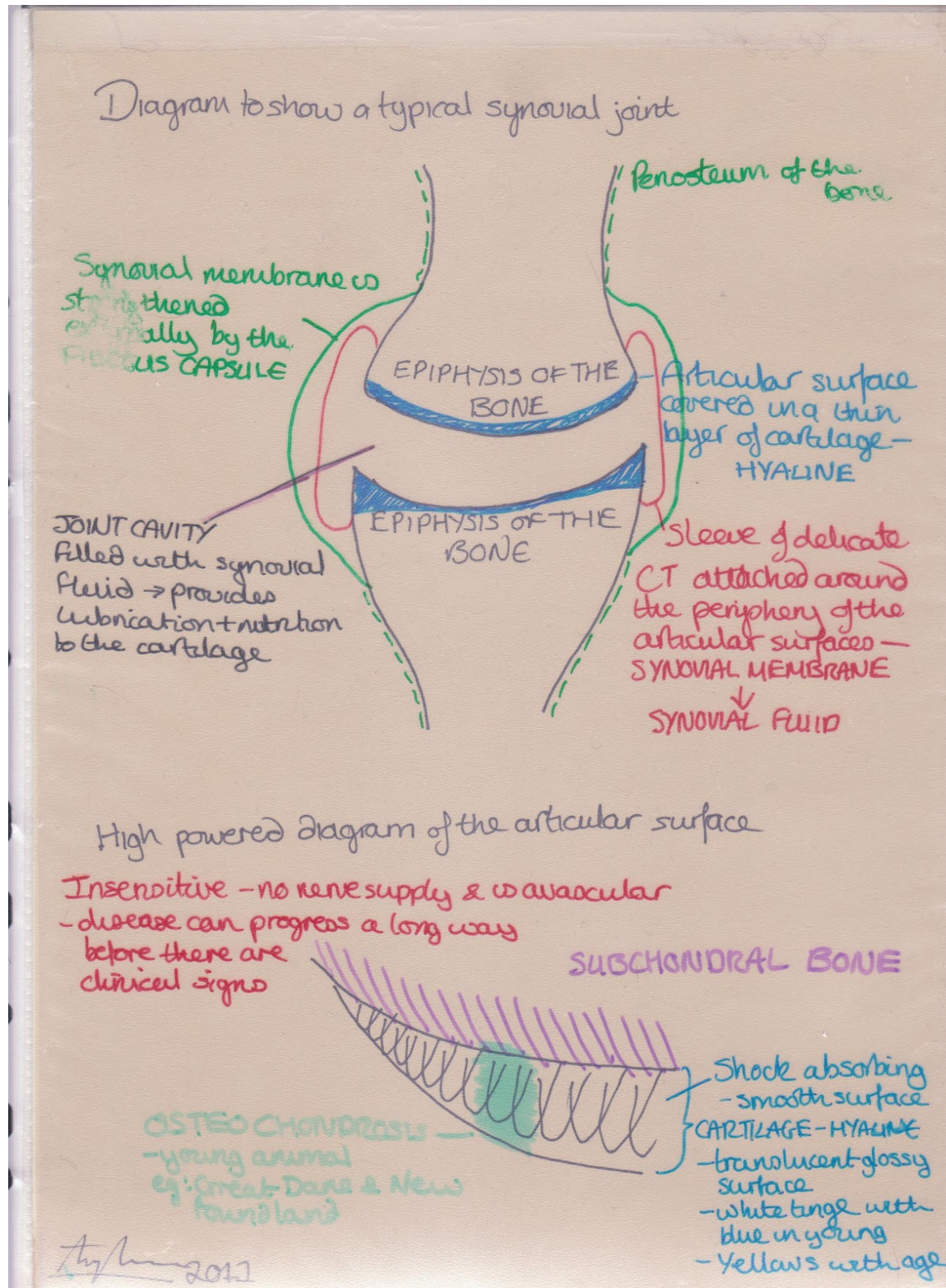
<http://blogs.adobe.com/flashplayer/2012/06/flash-player-and-android-update.html>, accessed 18th Mar 2016.

- [3] Biosphera - 3D Canine Anatomy Software.
<http://www.biosphera.com.br/e-veterinary-dog-anatomy.asp>, accessed 17th Feb 2016.
- [4] Google Play store. Animal Anatomy and Physiology.
<https://play.google.com/store/apps/details?id=code.alphonso.android.bookviewer.books.animalphysiology>, accessed 17th Feb 2016.
- [5] Google Play store. Dog Anatomy: Canine 3D.
<https://play.google.com/store/apps/details?id=com.realbodywork.doganatomy>, accessed 17th Feb 2016.
- [6] Google Play store. 3D Dog Explorer.
<https://play.google.com/store/apps/details?id=com.dg.dogexplorerres.android>, accessed 17th Feb 2016.
- [7] S. Kalyuga, P. Ayres, P. Chandler, and J. Sweller. The expertise reversal effect. *Educational Psychologist*, 38, 23-31, 2003.
- [8] Ruth Clark. Six Principles of Effective e-Learning: What Works and Why. *The eLearning Guild's Learning Solutions e-Magazine*. 10 Sep, 2002.
- [9] Wikipedia. Dual-coding theory. https://en.wikipedia.org/wiki/Dual-coding_theory, accessed 17th Feb 2016.
- [10] S. Kalyuga, P. Chandler, and J. Sweller. Incorporating learner experience into the design of multimedia instruction. *Journal of Educational Psychology*, 92, 126–136, 2000.
- [11] Android Studio Documentation. Android Studio Overview.
<https://developer.android.com/tools/studio/index.html>, accessed 18th Feb 2016.
- [12] Wikipedia. Android Software Development.
https://en.wikipedia.org/wiki/Android_software_development, accessed 21st Mar 2016.
- [13] WebView. Android Studio Documentation.
<https://developer.android.com/reference/android/webkit/WebView.html>, accessed 18th Feb 2016.
- [14] W3C HTML. <https://www.w3.org/html/>, accessed 21st Mar 2016.
- [15] Adobe PhoneGap. <http://phonegap.com>, accessed 18th Feb 2016.
- [16] Apache Cordova. <https://cordova.apache.org>, accessed 18th Feb 2016.
- [17] Python. <https://www.python.org/>, accessed 21st Mar 2016.
- [18] OpenCV. <http://opencv.org>, accessed 18th Feb 2016.
- [19] Python Image Library (PIL). <http://www.pythonware.com/products/pil/>, accessed 18th Feb 2016.
- [20] Pillow. <http://python-pillow.org>, accessed 18th Feb 2016.
- [21] Adrian Rosebrock. *My Top 9 Favourite Python Libraries for Building Image Search Engines*. PyImageSearch.
<http://www.pyimagesearch.com/2014/01/12/my-top-9-favorite-python-libraries-for-building-image-search-engines/>, accessed 18th Feb 2016.
- [22] JSON. <http://www.json.org/>, accessed 21 Mar 2016.
- [23] Numpy. <http://www.numpy.org>, accessed 18th Feb 2016.
- [24] The Python Wiki. TkInter. <https://wiki.python.org/moin/TkInter>, accessed 18th Feb 2016.
- [25] Hammer.js. <https://hammerjs.github.io>, accessed 18th Feb 2016.

- [26] Jake Archibald. *300ms tap delay, gone away*. Google Developers.
<https://developers.google.com/web/updates/2013/12/300ms-tap-delay-gone-away?hl=en>,
accessed 18th Feb 2016.
- [27] FT Labs. FastClick. <https://ftlabs.github.io/fastclick/>, accessed 18th Feb 2016.
- [28] Wikipedia. Base64. <https://en.wikipedia.org/wiki/Base64>, accessed 18th Feb 2016.
- [29] Wikipedia. Gaussian blur. https://en.wikipedia.org/wiki/Gaussian_blur, accessed 21st Mar 2016.
- [30] Survey Monkey. Rating & Ranking Average Calculations.
http://help.surveymonkey.com/articles/en_US/kb/What-is-the-Rating-Average-and-how-is-it-calculated,
accessed 14th Mar 2016.

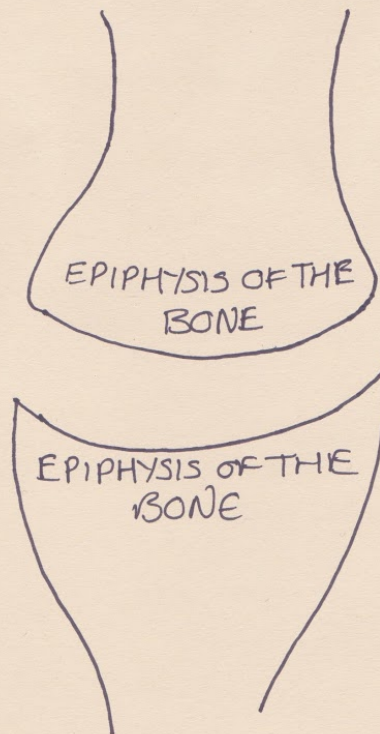
Appendix

[A1] Amy Livens. Diagram to show a typical synovial joint.

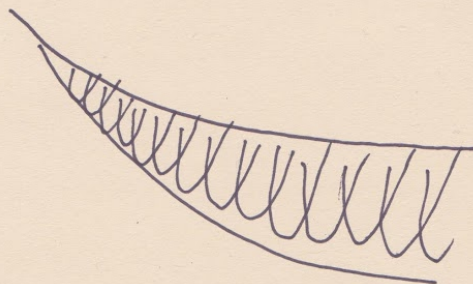


Combined base, first, second, and third layers

Diagram to show a typical synovial joint



High powered diagram of the articular surface




Aug/Sept 2011

Base layer



- Articular surface
covered in a thin
layer of cartilage -
HYALINE

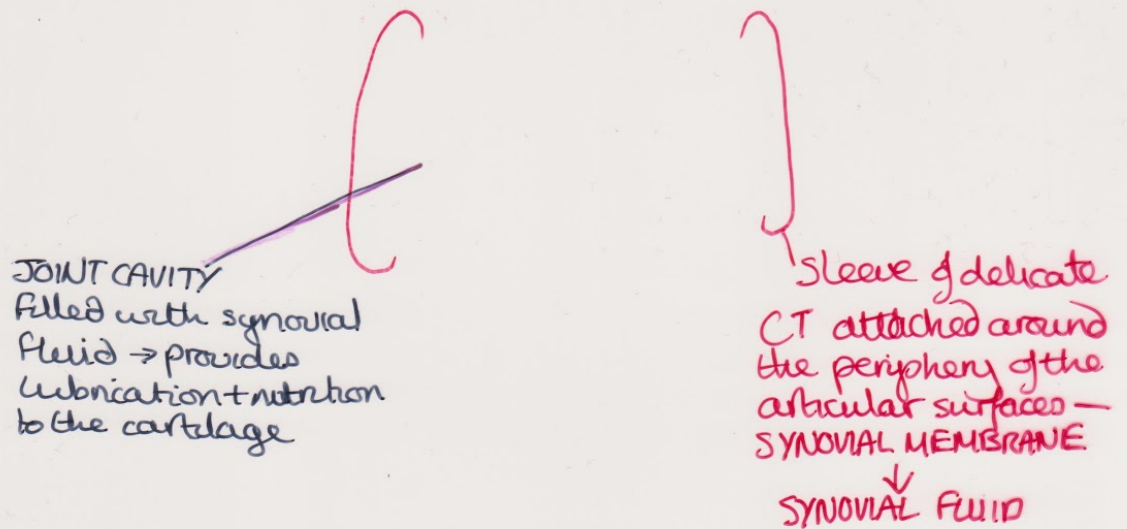
A hand-drawn diagram in blue ink showing two curved, parallel lines representing the articular surfaces of two bones. The lines are slightly irregular, suggesting a natural joint surface. The top line is slightly higher and more curved than the bottom line.



SUBCHONDRAL BONE

A hand-drawn diagram in purple ink showing a series of parallel, slightly curved lines representing the subchondral bone. The lines are closely spaced and follow a similar curve to the articular surfaces above them.


First layer

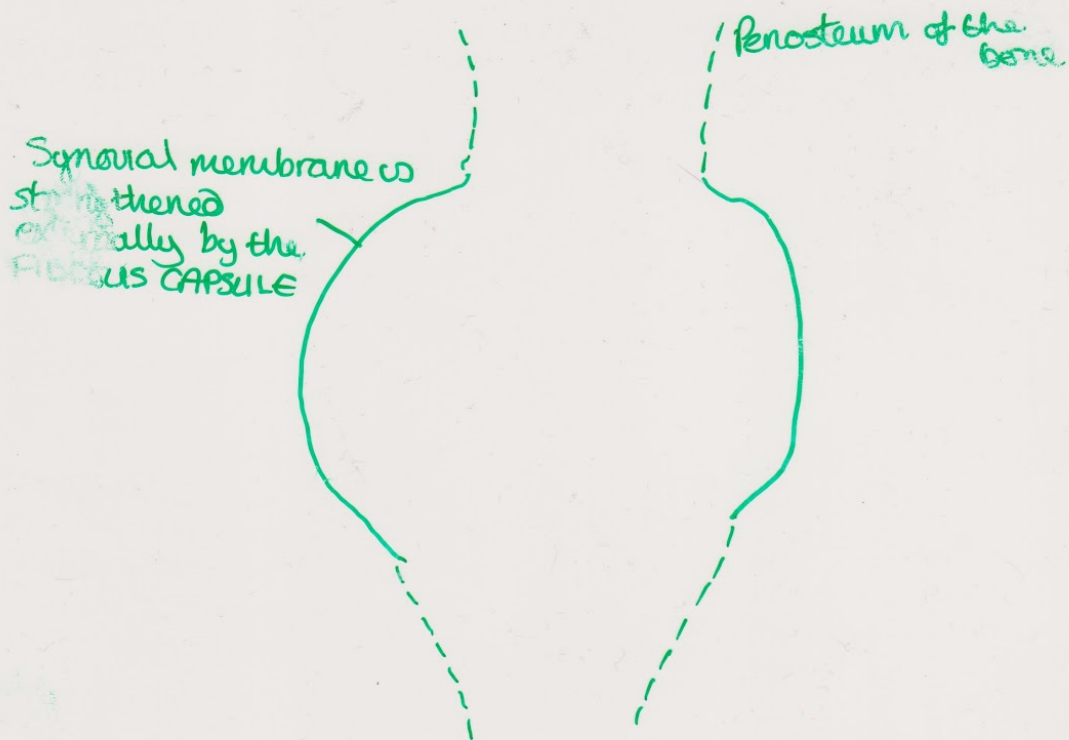


JOINT CAVITY

Filled with synovial
Fluid → provides
lubrication + nutrition
to the cartilage

Sleeve of delicate
CT attached around
the periphery of the
articular surfaces —
SYNOVIAL MEMBRANE
↓
SYNOVIAL FLUID

OSTEOCHONDROSIS — 
— young animal
eg: Great Dane & New
foundland



Insensitive - no nerve supply & is avascular
 - disease can progress a long way before there are clinical signs

Shock absorbing
 - smooth surface
 } CARTILAGE - HYALINE
 - translucent glossy surface
 - white tinge with blue in young
 - yellow with age

Third layer

[A2] Evaluation interview with Amy Livens

1: Was the principal goal of the project met? ("Build a smartphone/tablet app to help veterinary students with revision")

Yes

2: Was the app a reasonable realization of the original proposal/vision?

Yes

3: Was the app of a satisfactory level of completeness?

Yes it was complete and covered all the original documents

4: What changes to the current features of the app, if any, would you suggest?

I think that [for] any individual numbers that come up it would be better if the information was individual and not as a list. Some of the pictures (very few) may have benefited with being done in more layers.

5: What additional features, if any, would you have liked to see implemented?

Possibly a self test function could have been added so students could write their own answers and check their understanding

6: Would you use the app, as currently available? (If not, why not?)

Yes

7: Would you recommend the app, as currently available, to other students? (If not, why not?)

Yes definitely

Survey questions:

8: Was the material/content presented in an easily readable way?

Yes

9: Was the material/content organized in an easily navigable way?

Yes, it was very easy

10: Was the material/content partitioned into sensibly sized chunks?

Yes

11: Was the quality of the diagrams sufficient?

Yes, they very closely matched the originals

12: Was the accuracy of labelling of the diagrams sufficient?

Yes

13: Was the quality of labelling of the diagrams (accuracy aside) sufficient?

Yes

14: Was the app aesthetically pleasing?

Yes

15: Was the app intuitive to use? (sensibly organized; easy to learn; actions performed did not have surprising outcomes)

Yes very much so

16: Was the app easy to use? (once familiar)

Yes and it did not take long to learn how to use at all.

17: Was the app pleasant to use?

Yes

18: Was the app forgiving of user error; Did the app made it easy to recover from user error? (e.g. pressing the wrong button did not cause anything disastrous to happen)

Yes

19: Was the performance of the app sufficient? (it did not feel slow or sluggish during use; it did not freeze)

I have had no problems whatsoever whilst using the app, it has functioned perfectly every time

20: Were there any problems (aside from any previously mentioned) experienced when using the app?

None at all

21: Any other comments?

[None]

[A3] Survey questions

Section 1: general feedback questions

Answers of format: Strongly disagree/Disagree/Neither agree nor disagree/Agree/Strongly agree

1: The material/content was useful for revision purposes

2: The material/content was presented in an easily readable way

3: The material/content was organized in an easily navigable way

4: The material/content was partitioned into sensibly sized chunks

5: The accuracy of the material/content was sufficient

6: The quality of the diagrams (accuracy aside) was sufficient

7: The quality of labeling of the diagrams (accuracy aside) was sufficient

8: The app was aesthetically pleasing

9: The app was intuitive (sensibly organized; easy to learn; actions performed did not have surprising outcomes)

10: The app was easy to use (once familiar)

11: The app was pleasant to use

12: The app did not punish user error; the app made it easy to recover from user error (e.g. pressing the wrong button did not cause anything disastrous to happen)

13: The app should have been more dynamic/customizable

14: The performance of the app was sufficient (it did not feel slow or sluggish during use; it did not freeze)

Section 2: further comments

Answers of format: comments

15: Are there any features that could be added that would improve the app?

16: Were there any problems (aside from any previously mentioned) experienced when using the app?

Section 3: personal feedback questions

Answers of format: see Section 1 [A3]

17: I would use an app of this kind (not necessarily this app, but one with a similar function and design)

18: I would recommend an app of this kind (not necessarily this app, but one with a similar function and design) to a fellow student