

May 2013

Enhanced Laboratory Learning: Mobile Laboratory Application

Evan Adams Safford
Worcester Polytechnic Institute

John Stephen Synnott
Worcester Polytechnic Institute

Nathaniel William Miller
Worcester Polytechnic Institute

Nicholas C. Morin
Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/iqp-all>

Repository Citation

Safford, E. A., Synnott, J. S., Miller, N. W., & Morin, N. C. (2013). *Enhanced Laboratory Learning: Mobile Laboratory Application*. Retrieved from <https://digitalcommons.wpi.edu/iqp-all/3333>

This Unrestricted is brought to you for free and open access by the Interactive Qualifying Projects at Digital WPI. It has been accepted for inclusion in Interactive Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.

Enhanced Laboratory Learning:

Mobile Laboratory Application

By

Nathaniel Miller

Nicholas Morin

Evan Safford

John Synnott

An Interactive Qualifying Project:

Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

Abstract

With technology increasing in prevalence each day the potential for its usage in educational settings such as laboratories increases. We continued the development of a iOS mobile application for deployment in WPI Biology laboratories to serve as an alternative to traditionally lab conventions. Special attention was made to delivering a well-designed and well-documented application to ensure the ease of expanding the application in the future.

Authorship

Introduction – Nathaniel Miller, Nicholas Morin, Evan Safford, John Synnott

Analysis of Present Application – Nicholas Morin

Similar Products – Nathaniel Miller, John Synnott

Methodology – Nathaniel Miller, Nicholas Morin, Evan Safford, John Synnott

Results – Nathaniel Miller, Nicholas Morin, John Synnott

Recommendations for Future Work – Nathaniel Miller, Nicholas Morin

Conclusion – Nathaniel Miller, Evan Safford, John Synnott

Table of Contents

1	INTRODUCTION	5
2	ANALYSIS OF PRESENT APPLICATION	7
3	SIMILAR PRODUCTS	8
4	METHODOLOGY	10
	INSTITUTIONAL REVIEW BOARD:	11
	INITIAL APPLICATION IMPROVEMENT:	11
	PROJECT REFOCUS AND APPLICATION REDESIGN:	13
	TECHNOLOGIES UTILIZED:	13
	<i>Xcode</i> :.....	13
	<i>Objective-C</i> :	14
	<i>XML</i> :	14
	<i>Source Control</i> :.....	14
	<i>WPI Group Account/Server Space</i> :.....	15
	<i>CGI</i> :	15
	<i>Perl</i> :.....	16
	<i>Web Interface</i> :.....	16
5	RESULTS	17
	BIOLAB APPLICATION:	17
	WEB INTERFACE:	28
	PLANNED SURVEYING:.....	31
6	RECOMMENDATIONS FOR FUTURE WORK	32
	BIOLAB APPLICATION:	32
	LAB BUILDER APPLICATION:	32
	WEB INTERFACE:	33
7	CONCLUSION	34
9	BIBLIOGRAPHY	35
10	APPENDICES	36

Table of Figures

Figure 1: Biolab Home Screen	17
Figure 2: Lab Selection Menu	18
Figure 3: Lab Home Screen	19
Figure 4: Lab Section Selection	20
Figure 5: Text and Images	21
Figure 6: Questions	22
Figure 7: Tables	23
Figure 8: Video	24
Figure 9: Data Exportation	25
Figure 10: Menu Popup	26
Figure 11: Web Interface Main Page	28
Figure 12: File Upload Information Page	29
Figure 13: Lab File Management Page	30
Figure 14: Management Action Information Page	31

1 Introduction

The focus of this project is to review the current state of mobile laboratory applications in regards to their integration and utility in structured learning environments, and in addition work on the development of such an application. In doing so, this project hopes to uncover a better understanding of how mobile technologies can be used to enhance laboratory procedures. Ideally, technology should be able to benefit a classroom or laboratory environment without providing distractions or unnecessary challenges for the students, teaching staff, or professors. However, introducing or implementing technology into educational settings such as labs or classrooms may be viewed as having too steep a learning curve or being too time consuming to utilize effectively. As last year's group pointed out in their report, "when technology is used in the classroom, for presenting new information, collecting homework assignments or administering tests, there is often the fear that the technology will take away from the educational objective" (Claretti et al., 4). The question then arises whether that has to be the case or can technology be implemented effectively and in such a manner than it improves learning. Today, mobile technology exists nearly everywhere and is only becoming more and more prevalent. This can be seen in the widespread adoption and usage of cell phones, portable media players, laptops, tablets, and devices such as the Apple iPad. Such devices are used for countless purposes including communication, browsing the Internet, and entertainment. With these mobile technologies becoming so ubiquitous in society, using them to facilitate education should be explored.

The laboratory environment and labs are an integral part of the curriculum for the natural sciences and for many other areas of study in higher education. This is especially true at WPI, where there is a heavy focus on project and practical work. For the natural sciences, labs are the hands-on practical experiences and challenges that are presented to students in many courses in order to both teach important topics and to hopefully stimulate students' interest in the subject matter. For some courses, time spent inside the laboratory far outweighs that of time spent in lectures or doing other coursework. In this model, lecture is generally used to teach concepts and topics as a whole, and then the lab is usually used to teach more in-depth topics through hands-on, active, and structured learning. With so much time spent in working in the lab, the laboratory process should ideally be streamlined in order to ensure maximum efficiency and to prevent hindrance of learning through confusion or lack of resources.

However, can technology be utilized to streamline or improve laboratory learning? With any new piece of technology, there are certain user expectations associated with it. In the case of mobile technology or a mobile application, users expect the technology to truly be mobile and to not force the user to conform to the application, but for the application to assist them, and make their work easier. For example, users might want to be able to view and review lab procedures from locations outside the lab itself. Or they might want to have the ability to work on and edit the information gathered during the lab outside the laboratory. It is our belief and the belief of last year's group that a well designed and well planned mobile application for usage in the laboratory will "allow students to focus on the learning experience and will engage them" (Claretti et al., 4). Another IQP group did a project focused around designing an iOS application for laboratory usage and they concluded that by "streamlining the laboratory materials and resources, instructors and students would be able to jump directly into the lab work" (Erikson et al., 3). Condensing and centralizing the lab process into one application may eliminate the need for traditional printed lab handouts and lab notebooks. In doing so, the lab process may be simplified and laboratory learning may be improved.

The motivation behind our group's continued work on this application is twofold: first, the group's personal interest in utilizing technology in order to facilitate classroom learning, and second, to build upon the groundwork and research carried out by the previous year's team on the benefits of technology in education. In the background section of their paper, they focus on both the benefits and adaptation of technology in education. Through their research into those subjects, the authors come to the conclusion that "developing an application for a tablet will not only increase the value of the education, but the value of the student as an employee in the continually evolving business world" (Claretti et al., 5). We believe that their assessment and conclusions are still relevant to this project and serve as continued motivation for work and improvement on developing a mobile application for usage in a laboratory educational setting.

The goals set forth are to explore a subset of relevant applications on the market and analyze their features in order to determine whether those aspects are useful and why. In addition, the current mobile application in development for the WPI Biology department will be evaluated for potential improvement and subsequently be built upon. The objective is to determine what functionality a laboratory application would benefit most from. This will be achieved through an analysis of a few mobile applications that implement attributes relevant to a laboratory software

setting. In general, in order for a mobile application to be successful in an academic setting it ideally would satisfy at least the following criteria:

- Intuitive user interface
- Easy to learn and to explain usage to others
- More efficient than conventional methods (e.g. notes on paper, lab handouts)
- Provide structure to the material presented
- Offer help to the user if needed
- Have consistent appearance and feel

2 Analysis of Present Application

An analysis of the current version of the laboratory application was conducted in order to get a sense of what features need improvement. The developers of the current application gathered feedback last year about the application's strengths and weaknesses. As a result, the task of determining which elements to focus improvements on was streamlined. Based on the previous years IQP team's report, two potential improvements to the current application would be changing the ways data is exported and allowing for the storage of content on each mobile device running the application.

The first major improvement for the application would be the way it exports lab data gathered by the application, such as answers to questions or table data. Currently the application exports data by sending it in the body of an email. This is not an ideal format when considering potentially lengthy labs with large amounts of data collection. It also does not allow for transferring tables of information into other software such as Microsoft Excel. One possible manner of improving the exportation system would be to allow users to specify the format of their data and for it to be sent as attachments to email. Another potential solution for exporting data would be to simply parse the lab data into multiple documents such as Excel spreadsheets for tables and text files for short answer and essay questions. Both of these solutions would allow users to have more control over the format of the data while still speeding up the lab writing process. Another more elaborate solution would be to add the lab data into a generic lab report template allowing for a large amount of the lab report to be generated directly from the lab worksheet.

In addition to not sending lab data in a user-friendly format, the application does not allow for the lab data to be exported multiple times. “Once students [click] on export data they are required to export the data and cannot go back to the lab to change any data.” (Claretti et al., 30). This could lead to a lot of aggravation amongst students using the application if they accidentally clicked on export data. It would also be useful for students to be able to go back and collect more data and re-export if they realized a mistake had been made. It would be better for students to be able to click export and return to the lab and for students to be able to export their data multiple times if necessary.

The second major improvement for the current application would be to allow all of the labs to be stored locally on the device so that an internet connection would not be required to work on a lab. Currently the application must dynamically load each lab whenever a user wants to work on it. This requires an internet connection meaning that the lab application is not able to be used out in the field collecting data where it could be the most useful. Ideally the application should store a version of each lab on the device so that the user can load the lab anywhere. To further the improvement, a button could be added to the application to allow for last minute lab updates if any updates were needed.

3 Similar Products

The current application market for mobile devices is booming and continues to grow as mobile technologies become more mainstream and prevalent in society. Such mobile applications, “referred to as software systems operating on mobile devices, are evolving rapidly, making ubiquitous information access at anytime and anywhere a true reality” (Zhang, 2005). This raises the question of whether there are current applications or software that fit the needs and goals of a mobile laboratory application in a higher education setting? If there are, it would be useful to explore what kind of features and services such products offer. Doing so will provide insight into what kinds of aspects would be helpful in the current application. Research was done into the current mobile application stores and markets in the hopes of finding similar lab software. Through this search, it was discovered that while there are applications that offer some similar features that there are no applications specifically designed for use in college laboratory settings. This result is not surprising, as the market and demographic for such an application may be limited and not currently profitable. In addition, while higher education

institutions may have software for usage in the laboratory, those applications are likely internally developed and as a result are not publicly marketed or available. Notwithstanding, there are still applications on the market that do implement attributes that are pertinent to the current application in development and this section will focus on the evaluation of those similar products.

HUE Medscience--an online and mobile solutions company developed one such product, called Mylab. The premise behind Mylab is simple; it is “[...] a customizable app that lets you keep your lab data private and enables sharing of the content you want enabling full control of your files,” (HUE Medscience, 2010). The main features that Mylab provides are: protocols, inventory, notebook, catalog, calculators, orders, and Dropbox integration for data exportation. Of these, the notebook offers an intriguing and relevant service of data entry. In addition to using the notebook for data entry during a lab, HUE Medscience’s application enables the user to “search for data within the notebook by text or by dates,” as well as, “print or email selected pages to share results” (HUE Medscience, 2010). The ability for users of a laboratory application to search data that they’ve previously entered, as well as export their work in an easy and intuitive manner are features that are both useful and arguably necessary to such a piece of software. Efficient data entry, review, and exportability will be crucial if a mobile application is to be more effective and worthwhile than conventional laboratory methods of using handouts and physical notebooks. Mylab falls short however in that it does not offer a means of creating or displaying customized documents in such a way that would facilitate the app’s usage in an education laboratory setting. A lab application needs to be able to organize and display many different kinds of lab materials and present them in a straightforward and effective manner. Lab instructors need to be able to create a lab and then be able to easily make it available from within the laboratory application. Mylab does not offer such important functionality.

Mekentosj Inc. offers a piece of software called Papers, which they market as an application for organizing, viewing, and interacting with personal collections of research. Papers, available for Mac, PC, as well as iOS platforms, facilitates the importing, organizing, reading, annotating, searching, and syncing of a digital research library consisting of PDFs (Menkentosj Inc., 2012). The ability to import and display text based documents and materials is a relevant feature to a laboratory application. In order to provide the lab description, methods, procedure, and other details, it is necessary that text be displayed properly in the application. Similar to Mylab, Dropbox integration is implemented to allow importation of documents and files. While

Papers offers the ability to import, view, and display text documents it does not offer some of the features that would be necessary or useful in a laboratory application. For example, Papers does not support editing existing documents. The capacity to make edits, corrections, and additions to lab materials presented within a lab application is a feature that would be useful to lab instructors and those designing the labs. In addition, Papers does not support the displaying or manipulation of other relevant media such as images, videos, audio, tables, graphs, or diagrams. A lab may be filled with multiple media types and as such the ability to both display and interact with those materials is a feature that a lab application should support.

Not all lab related applications are restricted to iOS. Some have been developed for other mobile operating systems like Android. One such application is LabBook by PsiCode. LabBook is similar to some of the other applications that have been discussed; its primary purpose is to facilitate the recording of data acquired while performing experiments. Unfortunately there does not seem to be any way of using it to guide students through the lab, only to help them record the data. Like other applications, LabBook allows the user to upload their recorded data to Dropbox. The makers of LabBook also provide their own online storage which they say will make it easy to search, share, and read remotely. LabBook also boasts the ability to take photos, video and voice memos, all of which are very useful for speeding up note taking and note accuracy. In addition, it provides some useful features for helping students with the lab, like a barcode scanner, unit conversions, and solution calculator. Simple tools like these, which are also easily available, may help students progress through labs faster.

There are many applications that relate in some form to lab work, but there are none that have the collection of features that would be needed to be effective for both guiding students through a lab, as well as helping the students record the data acquired while performing experiments. What follows is a list of features that would be helpful, or required for an application to be used for every lab, in no particular order: image, audio and video capture, multiple convenient methods of exporting data, easy data entry, intuitive interface, central update location, local lab information storage, and the ability to easily create new labs.

4 Methodology

We began our project by looking at the project report of the IQP group that developed the first iteration of the application, as well as literature that we gained from our research sessions in

the library. Our initial research focused on finding similar applications, whether for usage within or outside laboratory settings. By focusing our research on applications and software not intended specifically for laboratory usage, the goal was to increase the quantity of material found. Our definition of a similar application was based around features we pinpointed from the current laboratory application developed for the Biology Department, for example: data entry and exportation or the displaying of various media types. Similar applications were looked into for two primary reasons: the first being to see if any had other features that might prove relevant to the application in development, and second being to see if there already exist applications that fulfill the needs of what the current application is trying to accomplish. Throughout this paper we will refer to the mobile iOS application in development as the Biolab Application and the Java application developed last year and used to generate labs as the lab builder application.

Institutional Review Board:

Originally, we intended to survey members of the WPI community who had been given a chance to use the application. We wanted to know what they thought about the application's ease of use, appearance, and the functionality it provided. A sample of planned survey questions can be seen in Appendix A. We applied to the WPI IRB in order to be granted exemption and permission to carry out research and surveys on human subjects. However, once our focus shifted to redesigning the application, the plans to conduct surveys and research were dismissed from the scope of our project. Please see Appendix B for the IRB exemption approval document.

Initial Application Improvement:

In the second stage of this project we turned our focus towards taking the current application and improving the data exportation system. This feature was pinpointed by the previous year's IQP group that developed the application as the area that warranted the most improvement and refactoring. As we have previously discussed in the introduction, the application handled data exportation using an e-mail based exportation system. Data and information entered into the lab was put into an auto-generated e-mail, which was then sent to any valid e-mail address recipient. While this system achieved the basic functionality of lab data exportation, we found a number of issues. The old application design did not enable a user to return back to the lab once the data exportation process had been initiated. This removed the

ability for changes to be made to the lab after the export e-mail had been generated. In the application's previous state, once a user had chosen to export the lab data, he or she either had to send the data or risk losing all the lab information, as the data was not saved locally on the device. This was a major design flaw that presented potentially significant inconvenience to the application users, including loss of data, productivity, and time.

In order to properly improve upon the application, specifically the data exportation system, we had to become familiar with the application code base and fully understand how it all fit together and worked. Through the methodical process of looking at the code base and figuring out the details of how it all worked, we discovered and pinpointed numerous design and technical issues within the application. While some of the problems we found were minor, a number of the problems turned out to be critical bugs in the application's design and code. The following table lists the major issues found, including the source and description of the issue:

Issue Source	Description
Lab builder application	Lab builder XML file generation does not properly include <EXPORT> tags to designate the data exportation section of each lab.
Biolab Application	Design flaw that expects an empty <ENTRY> tag in the lab XML file, which is not generated by the lab builder.
Biolab Application	Memory leak associated with the improper management and reuse of application views/tabs.
Biolab Application	Application crashes triggered by: <ul style="list-style-type: none"> - Swapping between lab sections too rapidly - Executing the data exportation system and returning the lab interface - Icon downloader system - Other unknown sources
Biolab Application	Embedded videos in labs do not load properly.

Biolab Application	Data from different views/tabs bleeds through into other views.
Biolab Application	Lab section text gets cut off within views.
Biolab Application	Memory management issues.
Biolab Application	Data entry information is not saved and can be lost easily.

Project Refocus and Application Redesign:

Through a group decision and subsequent approval by the project advisors, we decided to refocus the project and redesign the application. This decision was based on a number of different factors: the amount of issues found in the version of the application that we were provided, the amount of time we spent fixing or attempting to fix some of the issues, and the amount of time and rework we estimated it would take to fix all the issues found before any improvements could even be made on the application. With this decision came a couple of major changes to the goals of the project, namely: we would no longer be trying to improve the data exportation system, and we would no longer be carrying out surveys of the WPI community in regards to improvements. Instead, the new goals of the project became: redesign and develop the application to match the core functionality of the previous group’s application. However, in doing so, the hope of the project was to create a well-designed, well-documented, bug-free, and truly functional application for laboratory usage.

Technologies Utilized:

Xcode:

Apple’s proprietary IDE Xcode was utilized in this project in order to design, develop, and test the iOS application. We were provided with the Xcode project files from the previous year’s group, which we then brought back into Xcode in order to work on the code base. When the decision was made to redesign the application, we created a new project within Xcode and used the previous year’s project code as a reference.

Objective-C:

Objective-C is the programming language used to develop Apple's device operating systems as well as to develop applications, utilities, and games for both OSX and iOS systems. In order to facilitate development of this mobile laboratory application, the group had to become familiar with the language and how to use it within the context of building an application. This process was an iterative one, involving looking at Apple's developer resources, as well as tutorials, and simple hands-on testing and experimenting using the language.

XML:

XML, or Extensible Markup Language, is a markup language that is used to create schemas or definitions of groups of rules for document organization and encoding. The lab builder application developed by the previous year's team exports XML files which contain the data that represents a lab. The XML file is then parsed by the Biolab Application and is displayed. We continued to make use of XML and even extended its usage in our redesign of the iOS application and the other server space infrastructure to support the lab application.

Source Control:

Source control, also called version control or revision control, at its most fundamental level is the management of changes to data. This data can be files, documents, web sites, entire applications, or any number of other things. The purpose of source control is to organize the modifications made to information or data. Within the scope of this project, source control was used to organize and manage the changes and revisions made to the application and code base. Source control is a very common and widely used tool in software development and computer systems as a whole. In addition to keeping track of the different versions and changes made to the data, source control and source control management systems usually enable the users to revert to previous revisions. This feature is an invaluable tool in software development, where the need to roll back a change made to the code base is useful. For example, if an issue or major bug is introduced into the code, the ability to revert to a version before the issue can save time and resources. Source control also serves to provide a back-up system of sorts for the information being managed.

We chose to use WPI's supported source controller management system, named FusionForge, and which can be found here: <http://fusion.wpi.edu/>. Through this system, repositories can be requested and managed for projects or classes. A repository for the project, named IQP Mobile Laboratory Application, was requested and approved and was used to provide version control for the application code base throughout the project.

WPI Group Account/Server Space:

In order to facilitate the long-term and persistent storage and hosting of files, scripts, and other data required for the application it was determined that server space would be required for the project. Investigation was done into the best option to provide storage for the project that could easily be accessed, as well as could be reused for future projects or developments to the application. It was decided that the creation and usage of a WPI group account would fulfill the needs and requirements of our project. The group accounts offered by WPI can be used by clubs, groups, or academic project teams who desire a single login account that can provide web space and data storage in a central location for the group.

The group account was created using the web-interface provided by WPI (https://www.wpi.edu/cgi-bin/Pubcookie/project_support). The group account login was chosen to be biolabapp and is directly tied to the IQP project through the system. The storage space provided by the group account is identical to the personal file shares each student is provided on the filer.wpi.edu server. This storage can be accessed by either using SSH and connecting to the CCC server (ccc.wpi.edu) or by mapping a network drive to the account as one would do to access their personal file share.

The group account can be managed using WPI's group account management interface, located at <https://www.wpi.edu/+groupaccount>. Through this interface, the ownership of the group account can be changed. The ownership of the group was transferred to one of the project advisors once the project ended. Should another group wish to work on the application and need access to the group account, the ownership can be changed again at any point to facilitate that.

CGI:

CGI, the Common Gateway Interface, is a protocol or specification that is used by web servers in order to use executable files, such as scripts, to generate content for web display. CGI

is supported on WPI's main and user servers, which is why we chose to use CGI over alternatives such as Ruby, or PHP (which is specifically banned from usage on WPI servers). CGI was utilized among other web technologies such as HTML and CSS in order to create the simple web interface for our project's file server.

Perl:

Perl was the language that we used to write the executable files that were run on the server. It was used to accept files that were uploaded by users, store the uploaded files, and tell the application running on the iPad which labs were available, and where to look for them.

Web Interface:

The web interface created for this project was designed to enable the easy upload of lab XML files generated by the lab builder Java application to the biolabapp group account file server. The interface was also designed to support browsing and deleting of files already stored on the server. Lab creators and professors will utilize this interface to upload new labs to the file storage server for syncing with the mobile lab application. In order to ensure the security and integrity of the interface itself and the data on the server, WPI's Central Authentication System (CAS) was enabled. As a result, only members of the WPI community will be granted access to the web interface.

5 Results

Biolab Application:

The Biolab Application was designed to be as simple to navigate as possible in order to encourage efficient usage. The application has a home screen as shown in Figure 1, which states a quick welcome and brief instructions on how to get started with a lab.

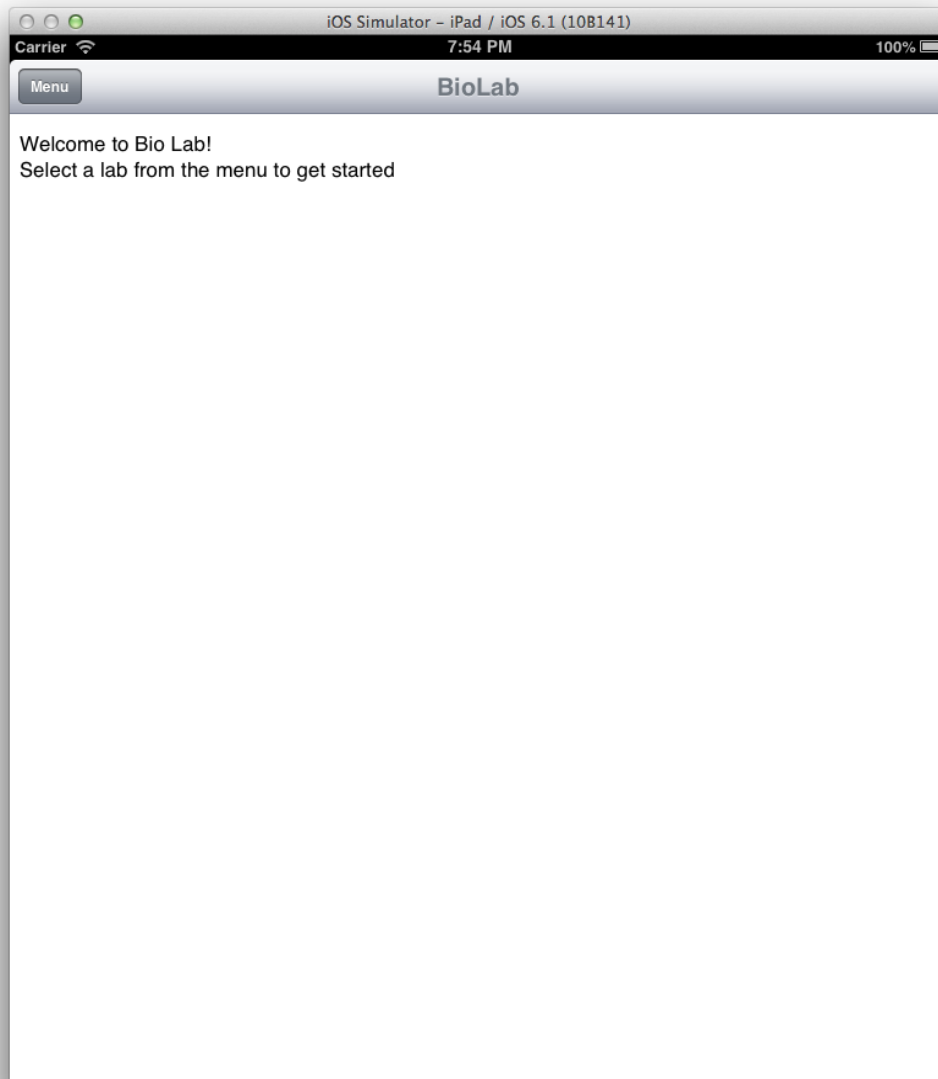


Figure 1: Biolab Home Screen

From here a user can press the *Menu* button to select which lab he/she would like to view or work on. This menu can be seen in Figure 2.

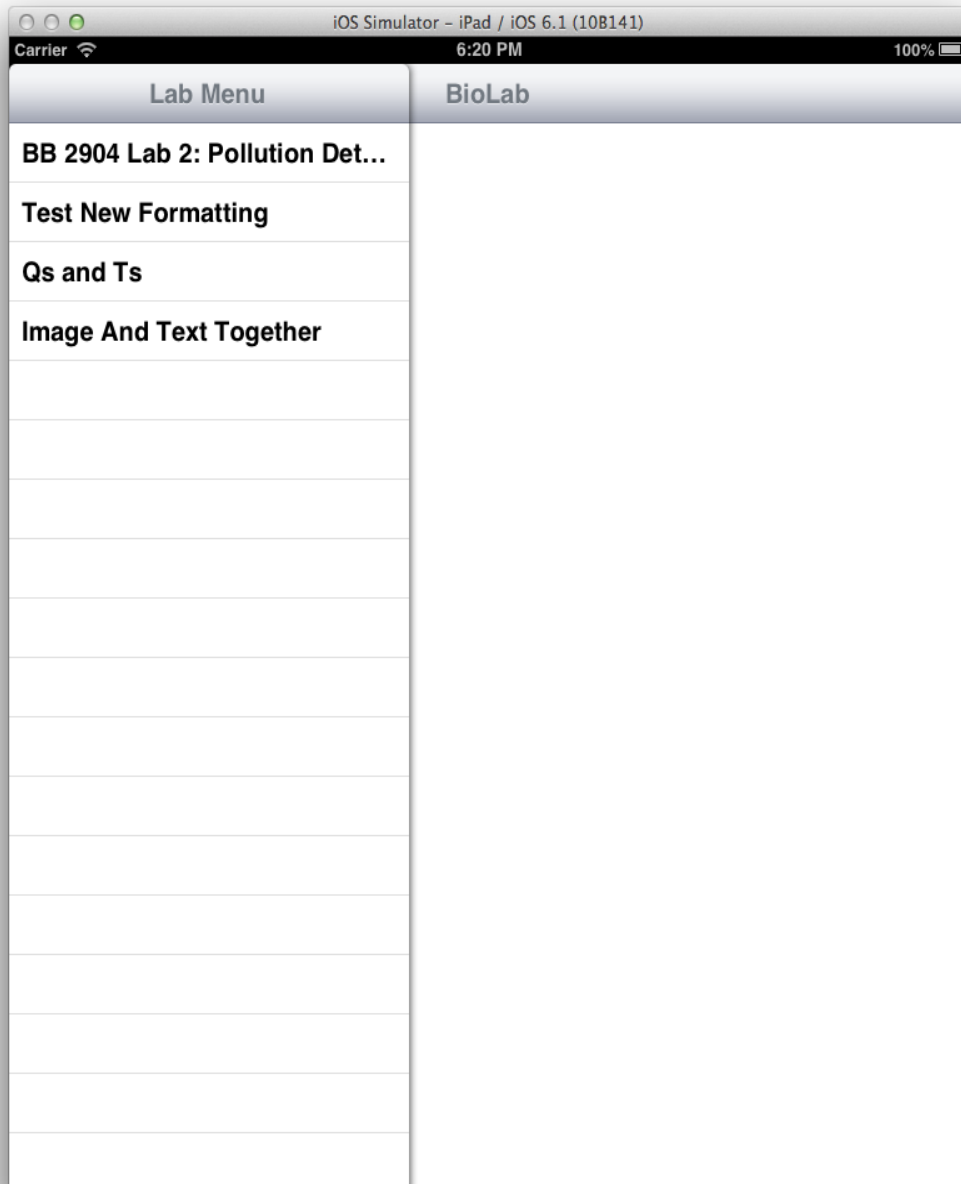


Figure 2: Lab Selection Menu

Once a user selects a lab, the home screen of the selected lab is loaded on the screen. This screen displays the name of the selected lab and updates the title shown at the top of the screen from “BioLab” to the name of the lab. Once this screen is loaded, an additional button is added to the top of the screen, which allows a user to navigate back to the home screen. This can be seen in Figure 3 below.

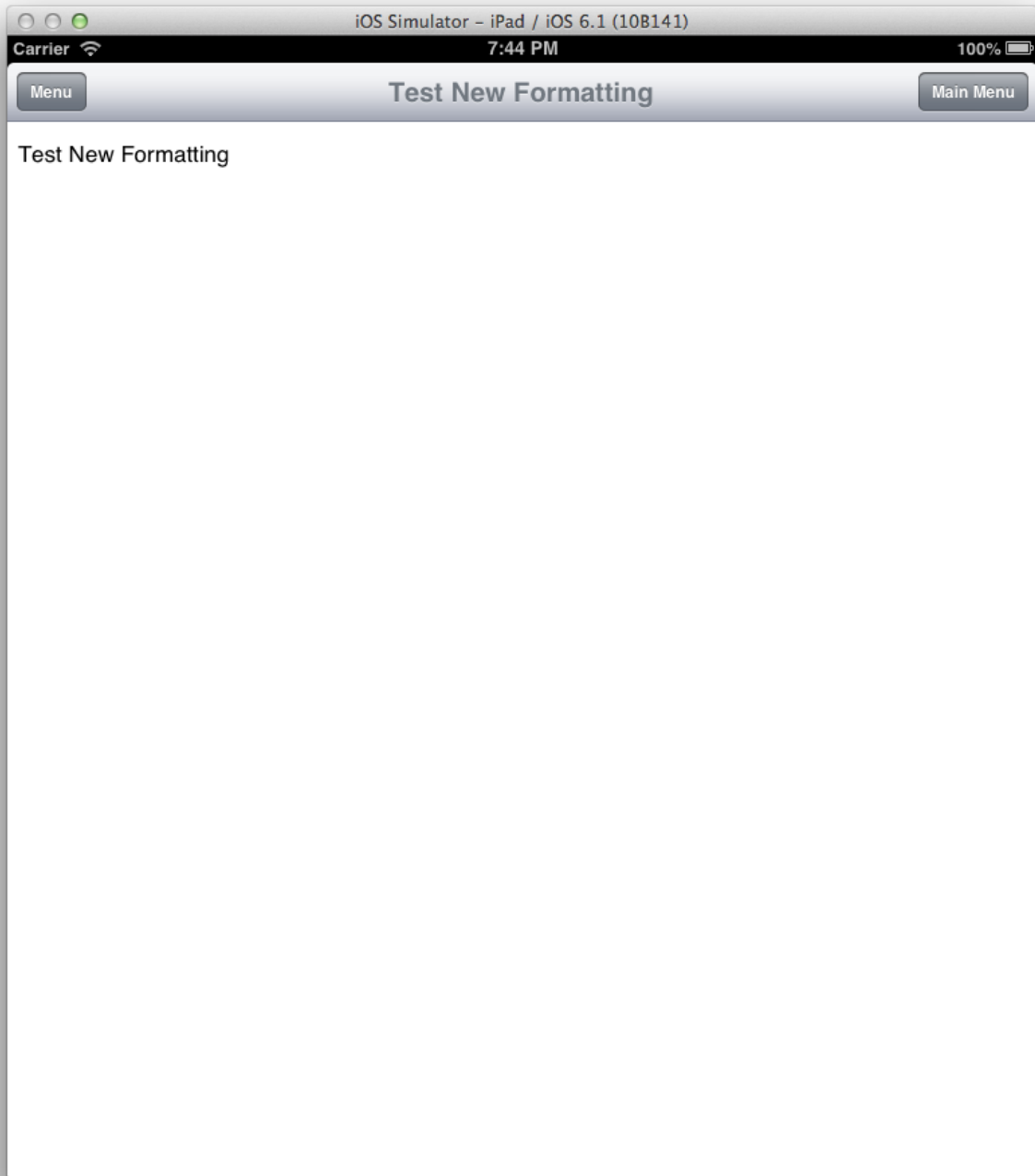


Figure 3: Lab Home Screen

From here, a user can press the Menu button again to choose to navigate between the individual sections of the lab as seen in Figure 4.

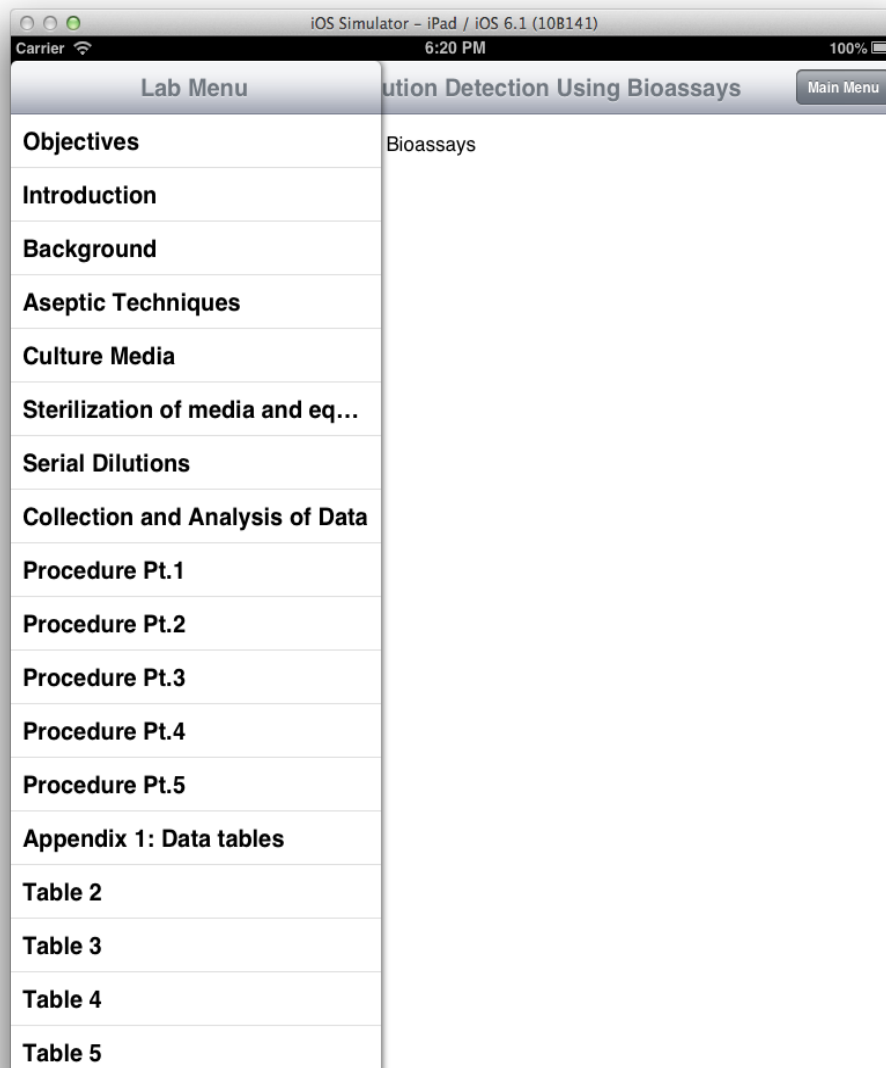
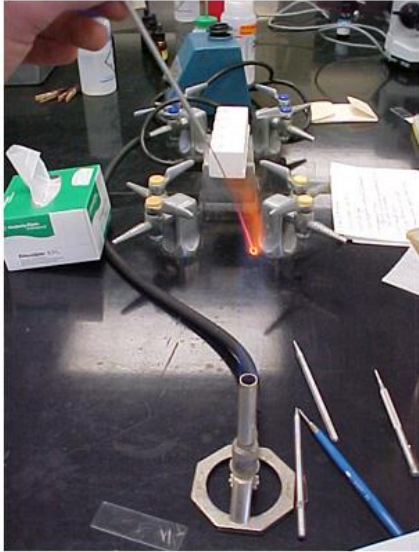


Figure 4: Lab Section Selection

Once a section is chosen, it is loaded on the screen. Loading a lab section also updates the title at the top of the screen to show the name of the lab and the section of the lab currently selected. If a section contains a question or a table that needs to be filled out, a user simply needs to tap the section he/she wants to edit in order to bring up the on screen keyboard and add data to the lab. The section loaded in Figure 5 shows examples of text and images, the section in Figure 6 shows questions and text input, the section in Figure 7 shows tables, and the section in Figure 8 shows video.

iOS Simulator - iPad / iOS 6.1 (10B141)
Carrier 6:21 PM 100%

Menu BB 2904 Lab 2: Pollution Detection Using Bioassays : Asepti... Main Menu




3. It is also frequently useful to inoculate liquid media in tubes, either from a colony on a Petri plate or from a tube culture. These transfers are handled in the following manner: the tubes to be inoculated are held in the left hand and the inoculating loop in the right hand (reverse this if you are left-handed). The caps or plugs of the tubes are removed and held between the fingers of the right hand (see Figure 4) throughout the entire procedure. The caps should not be placed on the bench top, and should be replaced on the tube immediately after the transfer is completed. Many technicians pass the top of the open tubes through a flame to sterilize the outside of the opening before and after inoculation. The wire transfer loop is then flamed and used to pick up the inoculum (if the inoculum is from a colony on a petri dish this step will precede uncapping and flaming the tubes) and the bacteria are either swished off the loop into liquid medium, or if an agar slant is to be inoculated, the loop is inserted to the bottom of the slant and then zigzagged over the surface to the top. Either way, no visible mass of bacteria should adhere to the loop at the end of the procedure, and the loop should be flamed before being put away in its holder. An experienced technician can inoculate 4 or 5 tubes at a time by gripping the tubes between the fingers of one hand, while holding their caps between the fingers of the other hand. Note the location of the caps - in the right hand, held between the fingers. Also note that the flame is nearby, ready for use as soon as the inoculation is completed. The openings of both test tubes should be flamed before the caps are

Figure 5: Text and Images

iOS Simulator - iPad / iOS 6.1 (10B141)
Carrier 6:33 PM 100%

Menu BB 2904 Lab 2: Pollution Detection Using Bioassays : Proced... Main Menu



1. Obtain one Luminescence plate for each member of your group as well as a master plate of previously streaked bacteria. 2. Each person should streak a plate by following the directions in the background above. 3. Be sure to label your plate with your name, the date, and the organism on the back. There is a reason for labeling the back, what do you think it is?

There is a reason for labeling the back, what do you think it is?

4. Place your plate back up in the 30°C 5. Check your plate in 24 and 48 hours to look for growth of bacterial colonies. a) Does your plate resemble the master plate? b) Did you get isolated colonies? c) Are there any spots or colonies that appear odd that might be contamination? The digital camera will be available if you would like to record your results photographically.

Does your plate resemble the master plate?

Did you get isolated colonies?

Figure 6: Questions

iOS Simulator - iPad / iOS 6.1 (10B141)
 Carrier 6:31 PM 100%

Menu BB 2904 Lab 2: Pollution Detection Using Bioassays : Table 2 Main Menu

Table 2:Daphnia Survival after Exposure to (solution name)


Tube/dilution	Fractional dilution	% Concentration	Actual Dil.	Actual concentration	# of surviving Daphnia	% survival
F	0-no dilution	100				
10 ⁻¹	1/10	10				
10 ⁻²	1/100	1.0				
10 ⁻³	1/1000	0.10				
10 ⁻⁴	1/10,000	0.01				
NC (negative control)	No product added	0				

Figure 7: Tables

iOS Simulator - iPad / iOS 6.1 (10B141)
Carrier 6:22 PM 100%

Menu BB 2904 Lab 2: Pollution Detection Using Bioassays : Proced... Main Menu

Part 1: Dilutions In the back of the room you will find 8-10 household products to test for toxicity. You will be making serial dilutions of 3 of these to be used in the two bioassays so that you can test and compare their toxicity. For the purposes of getting meaningful class data you will be assigned one of the products numbered 1-4 to test. You are free to choose any of the others that you want to examine for your 2nd and 3rd products. The use of serial or stepwise dilutions will enable you to make accurate dilutions to low concentrations using only small volumes of product and reagents. You can see a video demonstration of 10 fold dilutions at



You will be using micropipettors extensively this week. If you are unfamiliar with micropipettors or haven't used them for a while you should review the document on how to use them in the course info section of the website and in particular the demonstration video at Prepare 10 fold serial dilutions using the following procedure: 1. Obtain a sample of 3 different products to test. Two are your choice so pick something interesting to you, the other will be assigned to you. Also be sure you have a bottle of 3% saline (salt water) and a bottle of pond/spring water, these will probably be at your bench already. 2. Obtain 5, 1.5 ml microfuge tubes for each product you will dilute and label them with the product name or some code of your choosing using a lab marker. Five for each product. 3. Then label one of each set of 5 with F (full strength), 10-1, 10-2, 10-3, and 10-4 to indicate the dilution in each one. 4. Set the tubes in a microfuge tube rack in the above order. 5. Using a micropipettor add 1 ml of product (X) to the appropriately labeled F tube. This will serve as your stock solution so you can return the class stock to the front of the room and is not really a dilution. 6. Add 900 μ l of 3% saline to each of the 4 remaining tubes (10-1, 10-2, 10-3, and 10-4)

Figure 8: Video

Once a user has finished editing his/her lab, he/she may select the Export Data option from the section selection menu. This will create a new email with each table attached as a separate .csv (comma separated value) file and all questions attached in a .txt file. The table files can easily be opened inside of Microsoft Excel and similar spreadsheet programs, and the question text file can be easily opened inside of any word processing program. The export section can be seen below in Figure 9.

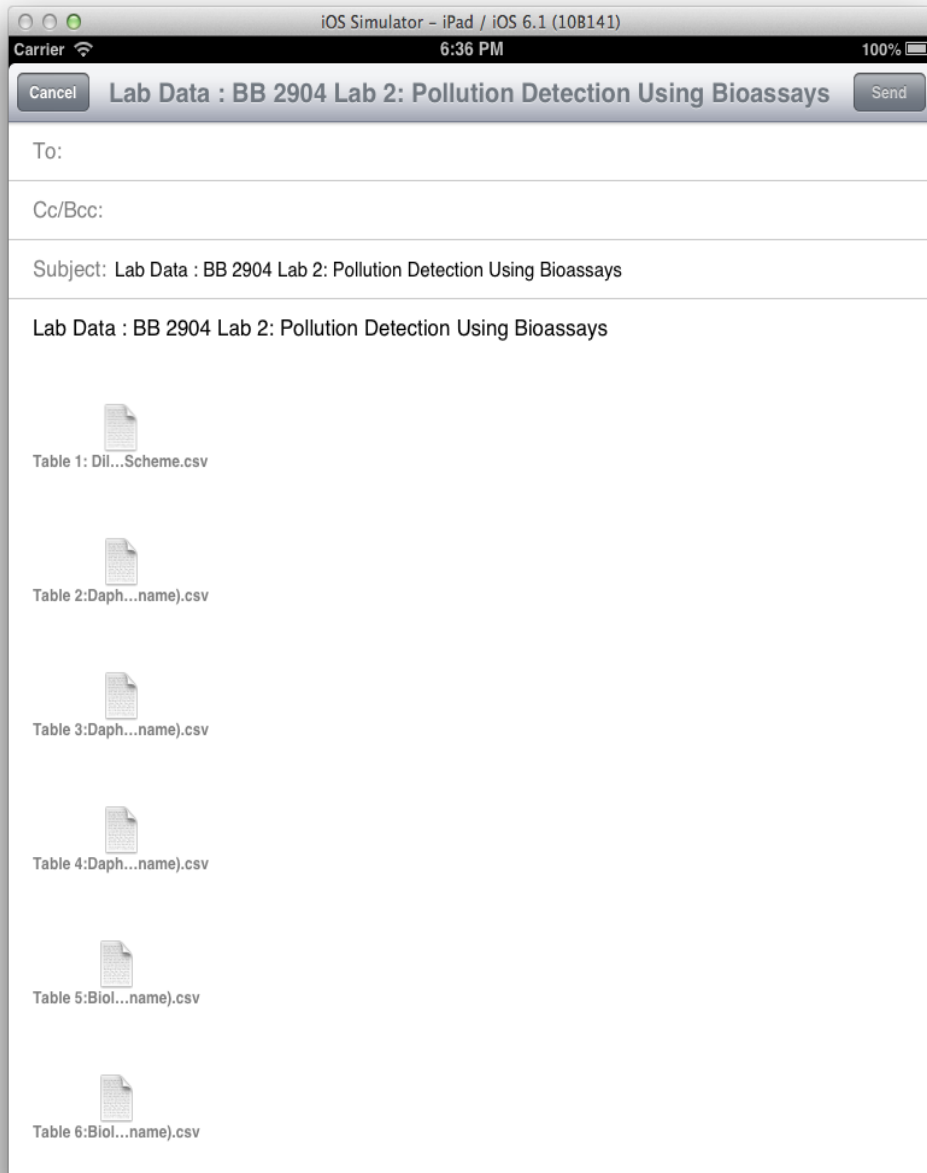


Figure 9: Data Exportation

Once a user has finished sending an email, or once the user has decided to cancel the email, the user is brought back to the lab. From here the user may choose to edit the lab more, close the application, or return to the main menu to select a different lab to work on. To return to the main menu, the user presses the *Main Menu* button in the top right of the screen and follows the on screen popup, in Figure 10, to return to the menu. The popup was added because once a user returns to the main menu, all data entered into a lab is lost.



Figure 10: Menu Popup

The main menu of the Biolab Application is loaded from the server space from a file formatted like the example in Appendix C. Each lab is then loaded from the server from files that look like Appendix D. The lab files are generated from the lab builder application created by last year's group.

The size of all data fields in the lab is determined by the orientation of the iPad when each tab is loaded. This means that unlike many iOS applications, the size of the lab data only changes when the tab is reloaded.

Web Interface:

The web interface was designed to enable lab creators, such as faculty or teaching staff, to upload lab files to a centralized server storage location. Lab files designed and generated through the Lab Builder Java application can be uploaded using the main file upload interface shown in Figure 11. Once a lab file is uploaded to the server, it is then accessible from the Biolab Application. In addition to file upload, the lab management page is also accessed from this main interface.



Figure 11: Web Interface Main Page

The file upload process was set up to be as simple as possible: a user accesses the web interface, selects the file to upload using the option on the page, and then clicks upload. Upon clicking the upload button, a new page is presented, which displays information about the upload. This information is either a confirmation of a successful upload, as shown in Figure 12, or an

error message if the file could not be uploaded. Potential causes for file upload failure include: the file exceeds the max file size of 5MB, the file is not an XML file, the file contains no data, or the XML is improperly formatted.



Figure 12: File Upload Information Page

Labs uploaded to the server through the web interface can be managed through the lab file management page, depicted in Figure 13. This page lists all the labs currently stored on the server and provides an option to delete the files. Once a file is successfully deleted from the server, it is no longer accessible from the Biolab mobile application. The list of lab files on this page is generated and updated based on the contents of a lab lister XML file hosted on the server, an example of which can be seen in Appendix C. This file contains an entry for each lab XML file currently stored on the server and direct URL to the file, which the Biolab Application uses to download the lab files for display and usage.

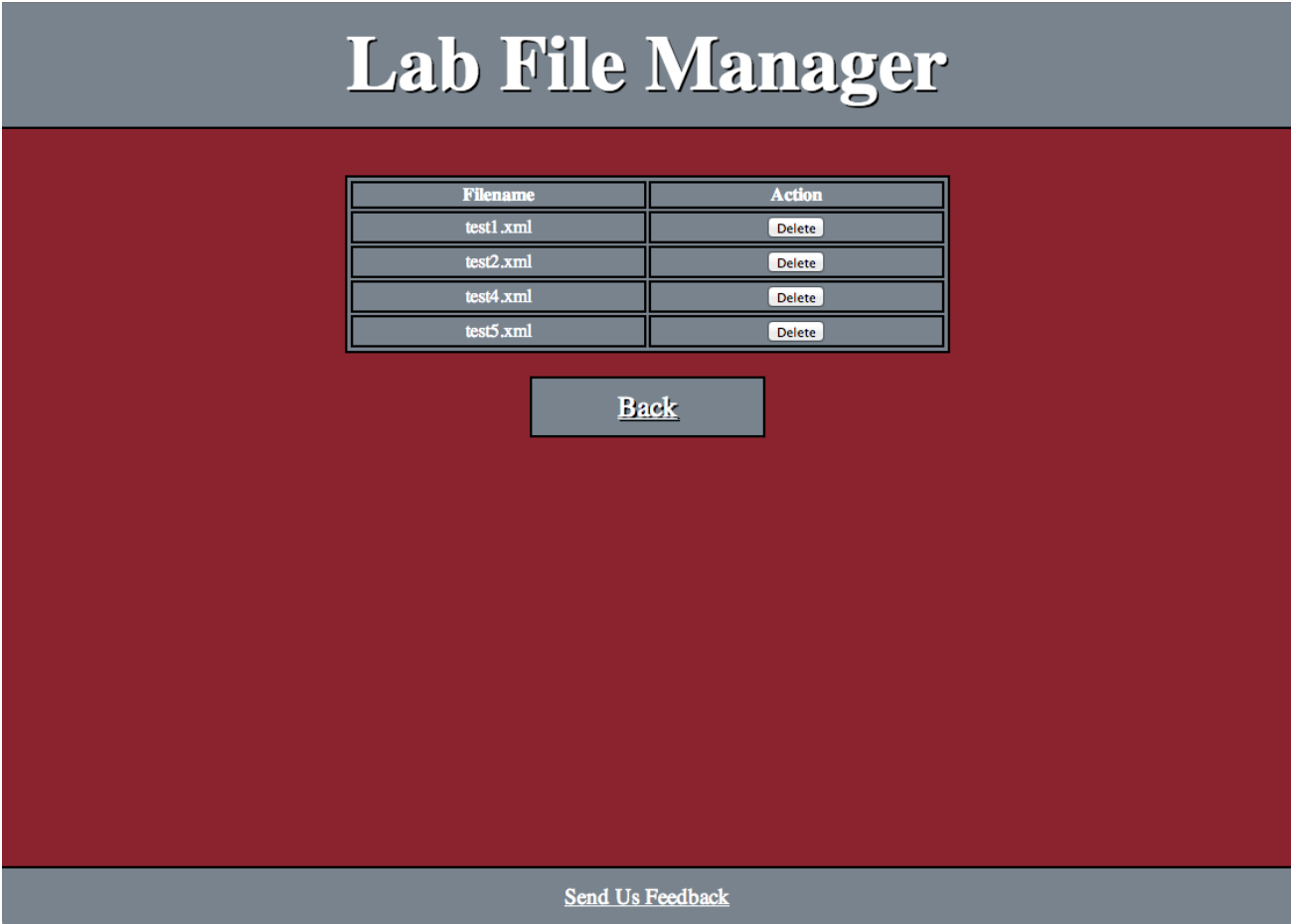


Figure 13: Lab File Management Page

Once a file is selected for deletion, a new information page is displayed, which confirms whether or not the file was successfully removed from the server. An example of this can be seen in Figure 14.

Manager Information

Deletion Information

Successfully deleted test1.xml.

[Back](#)

[Send Us Feedback](#)

Figure 14: Management Action Information Page

The goal of creating the web interface was to provide a straightforward and easy to access system to those making and administering labs. We believe the web interface achieves the aforementioned goal and in doing so prevents the need for those utilizing the interface to have to use other less easy-to-use methods, such as accessing the server directly via SSH.

Planned Surveying:

Included among our plans when starting this project was conducting surveys of faculty, teaching staff, and students on the usage of the application in order to gather information about whether the application met both our developmental goals as well as fostered improved learning. Using questions such as, “Yes or no, did the application foster a more efficient manner of lab data entry compared to traditional lab notebooks and handouts?” we intended to gather useful

and quantitative data about whether or not the application did or did not meet certain goals or criteria. A complete list of planned survey questions can be seen in Appendix A.

6 Recommendations for Future Work

Biolab Application:

We pinpointed a number of features that could be considered for improving and adding to the present mobile application. Below we have outlined these potential improvements or future endeavors.

A major area for potential future work is the data exportation system, which at its core, remains fairly limited. Although it allows for data to be easily imported into other programs such as Microsoft Excel and Word, it would be nice if the Biolab Application could format the data into a selection of lab templates so a student could have one file to open inside of a word processor allowing for faster lab report generation.

Currently, when the mobile lab application is exited, all the data entered is lost. This could lead to easy frustration if a student were to accidentally close the application before using the exportation section. The addition of local device storage or caching labs would be a useful feature, enabling the closing of the application without loss of work or time.

A step further than saving labs locally on the device would be to implement an account system for the application and store labs and reports associated with each account on a server for backup and accessing. This feature would be quite an undertaking and would require a lot of design and planning, in addition to acquisition of the required server storage space.

The ability to take pictures and or record videos from within the lab application itself would be a useful feature in a laboratory setting to enable users to document the lab.

Further improvements to the application's functionality could also include: implementing resizing upon iPad orientation changes, adding the ability to customize the application and individual lab home screens through changes to the XML, as well as bettering the overall appearance and aesthetics of the application.

Lab Builder Application:

Depending where this project goes in the future, the lab builder application may be more suited as an entirely web-based application. There are numerous advantages to making the lab

builder a web application, such as greater overall accessibility, easier integration with server storage, the ability to have built in file upload, and removing the need to have a desktop application in order to create labs. Shifting the functionality of uploading lab files to a lab builder web application would also potentially remove the need for a separate web interface to upload and manage lab files.

Web Interface:

There are multiple areas where improvements could be made to the web interface, ranging from usability and additional features, to security.

The first would be adding further functionality to the labs management page, such as the ability to download, rename, or view lab files stored on the server within the web browser itself.

The second would be adding more security and restrictions. In its current state, any member of the WPI community with a WPI account has the ability to access the interface through CAS. Added security could be accomplished by limiting which users can access the interface itself through the authentication system, or even further by imposing user permissions in regards to files uploaded to the server. For example, only allowing the user who uploaded a lab file to delete it from the server may be functionality to consider.

The third would be to add additional metadata to each uploaded lab, such as date uploaded, the user who uploaded it, what class it is used for, what term and year it is intended for, and any other useful information wanted.

7 Conclusion

Overall, we believe that we have successfully achieved our goals of carrying out research and analysis of similar applications, as well as working on the development of a mobile laboratory application for the Biology Department at WPI. Although it yielded limited results, our initial research offered us the necessary background to understand what features and functionality a well-designed application would need to have or might benefit from having. In addition, information and insight provided by the previous year's group contributed greatly to helping us formulate the goals for our project and for the continued development of the application. This knowledge proved vital during our analysis of the previous group's application and in our subsequent decision to redesign the application. We believe that our decision to redesign the application was both necessary and beneficial in order to ensure that the application met our project goals as well as our developmental goals of being well designed, well documented, extensible for future work, and easy to use. In addition to the mobile application, we believe the creation of the web interface was needed in order to provide a straightforward system for those creating and managing labs to upload and store lab files in a centralized and stable location. Due to our decision to refocus the project, the core functionality of the Biolab Application has not changed greatly from last year's version. However, while we were not able to implement major visible additions to the application as we initially intended, we believe we have greatly improved the application and made it easier for future groups to move forward with development.

9 Bibliography

Claretti, Ennio, Joseph A. Everett, Benjamin J. Leone, Nikolaos Matthiopoulos, Christopher T. O'Connor, Max A. Saccoccio, and Michael D. Yeroshalmi. *Enhancing Science Laboratories with Technology*. Rep. WPI, 28 May 2012. Web. Nov. 2012. <<http://www.wpi.edu/Pubs/E-project/Available/E-project-052812-152914/>>.

Naismith, Laura, Peter Lonsdale, Giasemi Vavoula, and Mike Sharples. *Literature Review in Mobile Technologies and Learning*. Bristol: NESTA Futurelab, 2004. *Leicester Research Archive*. Futurelab, Dec. 2004. Web. <<https://lra.le.ac.uk/handle/2381/8132>>.

Lippi, G. "Laboratory Applications for Smartphones: Risk or Opportunity?" *Clinical biochemistry* 44.4 (2011): 273. Print.

Zhang, Dongsong. "Challenges, Methodologies, and Issues in the Usability Testing of Mobile Applications." *International journal of human-computer interaction* 18.3 (2005): 293-308. Print.

Wasserman, Tony. "Software engineering issues for mobile application development." *FoSER 2010* (2010).

"Mylab." *HUE Medscience*. HUE Medscience, 2010. Web. 2 Dec. 2012. <<http://www.huemedscience.com/products/mylab>>.

"Papers for iPad." *mekentosj.com : Software for Research*. Mekentosj Inc., 2012. Web. 2 Dec. 2012. <<http://www.mekentosj.com/papers/ipad>>.

Barton, Geraint, Derek Huntley, and Mark Woodbridge. "LabBook ." *LabBook*. N.p., n.d. Web. 1 Dec. 2012. <<https://labbook.cc/>>.

Erikson, John A., John D. Neary, Alexander G. Orphanos, and David A. Polito. *IOS Application Development for Laboratory Use*. *WPI Electronic Projects Collection*. Worcester Polytechnic Institute, 15 Mar. 2011. Web. 1 May 2013.

10 Appendices

Appendix A: Planned Survey Questions

Have you used the application before?

How easy was the application to navigate? (Please rate 1-5 with 1 being very difficult, 5 being very easy)

How well is the application organized? (Please rate 1-5 with 1 being not organized, 5 being well organized)

What, if anything, could be organized better?

Yes or no, does the application “look good”? If no, please explain.

Yes or no, were you able to easily enter data and other media into the application during usage? If no, please explain.

Yes or no, did the application foster a more efficient manner of lab data entry compared to traditional lab notebooks and handouts? If no, please explain.

Was the data exportation process straightforward and easy to navigate? (Please rate 1-5 with 1 being very difficult, 5 being very easy)

Yes or no, did the data exportation system enable you to export your lab data and report in a useful and usable format? If no, please explain how it could be improved.

What did you find lacking in the application’s way of exporting data?

What did you find particularly good about the exportation system?

What, if any, other data entry/export programs have you used in previous labs or in general? If this is the first of such program you’ve used, please say so.

What did you find useful and/or good about the application?

Yes or no, would you want to use the application for labs in its current state? If no please explain.

Did any part of the application stop working unexpectedly?

What did you dislike or think could be improved?

Did the application have any unexpected behavior?

Any other comments or observations?

Appendix B: IRB Exemption Approval

WORCESTER POLYTECHNIC INSTITUTE

Worcester Polytechnic Institute IRB# 1
HHS IRB # 00007374

22 February 2013
File:13-015

Re: IRB Application for Exemption #13-015 "Mobile Laboratory Applications"

Dear Prof. Buckholt,

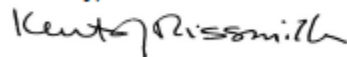
The WPI Institutional Review Committee (IRB) has reviewed the materials submitted in regards to the above mentioned study and has determined that this research is exempt from further IRB review and supervision under 45 CFR 46.101(b)(2): "Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures or observation of public behavior, unless: information obtained is recorded in such a manner that human subjects can be identified, directly or through identifiers linked to the subjects; and any disclosure of the human subjects' responses outside the research could reasonably place the subjects at risk of criminal or civil liability or be damaging to the subjects' financial standing, employability, or reputation."

This exemption covers any research and data collected under your protocol from 22 February 2013 until 21 February 2014, unless terminated sooner (in writing) by yourself or the WPI IRB. Amendments or changes to the research that might alter this specific exemption must be submitted to the WPI IRB for review and may require a full IRB application in order for the research to continue.

Please contact the undersigned if you have any questions about the terms of this exemption.

Thank you for your cooperation with the WPI IRB.

Sincerely,



Kent Rissmiller
WPI IRB Chair

Appendix C: Example Labs.xml File

```
<BIOLAB>
<LABFILE title="BB 2904 Lab 2: Pollution Detection Using
Bioassays">http://users.wpi.edu/~biolabapp/labs/FullLab.xml</LABFILE>
<LABFILE title="BB 2904 Lab 2: Pollution Detection Using
Bioassays">http://users.wpi.edu/~biolabapp/labs/FullLab2.xml</LABFILE>
<LABFILE title="Test Lab 0">http://users.wpi.edu/~biolabapp/labs/test_lab_0.xml</LABFILE>
<LABFILE title="Test Lab
1">http://users.wpi.edu/~biolabapp/labs/test_lab_1.xml</LABFILE>
<LABFILE title="Test Lab
2">http://users.wpi.edu/~biolabapp/labs/test_lab_2.xml</LABFILE>
<LABFILE title="Test Lab
3">http://users.wpi.edu/~biolabapp/labs/test_lab_3.xml</LABFILE>
</BIOLAB>
```

Appendix D: Example Lab XML File

```
<LAB title="Sample Lab 1">
<ENTRY title="Section 1">
<TEXT title="Text 1">Sample text 1</TEXT>
<INPUTTEXT title="Question 1">What is your name?</INPUTTEXT>
</ENTRY>
<ENTRY title="Section 2">
<VIDEO title="Sample Video">http://www.youtube.com/samplevideo</VIDEO>
<IMAGE title="Sample image">http://www.image.com/sample</IMAGE>
<TABLE title="Sample table">
<ROW>
<COLUMN>1</COLUMN>
<COLUMN>3</COLUMN>
</ROW>
<ROW>
<COLUMN>2</COLUMN>
<COLUMN>4</COLUMN>
</ROW>
</TABLE>
</ENTRY>
</LAB>
```

Appendix E: Web Interface Source Code index.cgi:

```
#!/usr/bin/perl -wT

use CGI;
use strict;
use warnings;
```



```

my $cgi_query = new CGI;

# Retrieve the accessing user's WPI affiliation and username from
# the environment variables set by CAS upon authentication.
my $affil = $ENV{'HTTP_CAS_EDUPERSONPRIMARYAFFILIATION'};
my $user = $ENV{'HTTP_CAS_USER'};

# The set of valid WPI usernames that can access the web interface.
# Access can be restricted to a group of usernames by adding the usernames
# to this array.
my @valid_users = ();

# Function to check if the user trying to
# access the resource is a valid user as designated
# by the valid users array.
sub validUser {
    my $user = $_[0];
    foreach my $valid_user (@valid_users) {
        if($user eq $valid_user) {
            return 1;
        }
    }
    return 0;
}

# The set of valid WPI user affiliations that can access the web interface.
my @valid_affils = ();

# Function to check if the user trying to
# access the resource is of a valid affiliation as designated
# by the valid affiliations array.
sub validAffil {
    my $affil = $_[0];
    foreach my $valid_affil (@valid_affils) {
        if($affil eq $valid_affil) {
            return 1;
        }
    }
    return 0;
}

# If more restricted access to the web interface resource is required,
# the above functions and data structures can be utilized to conditionally
# determine whether to generate the web page or to redirect the user to
# and error page.
# To redirect use the following code:
# print $cgi_query->redirect('https://users.wpi.edu/~biolabapp/forbidden.html');

# Print the necessary content header.
# NOTE: this should only be printed if the web page is to be generated successfully.
# If, instead, the user is not authorized and will be redirected, then this header should not be printed.

```

```

print $cgi_query->header();

# Print the index web page for display
print
"<!DOCTYPE html>
<html>
<head>
<meta charset=\"UTF-8\">
<title>Biolab Web Interface</title>
<link rel=\"stylesheet\" type=\"text/css\" href=\"style/index.css\">
</head>
<body>
<div id=\"topbar\"><h1>Biolab Application Web Interface</h1></div>
<div id=\"formBox\">
<div id=\"description\">
<h2>Instructions</h2>
<ul>
<li>Please only use this interface to upload lab XML files generated by the lab builder application.</li>
<li>The maximum filesize is 5MB.</li></ul>
<p>To upload a lab file to the server:</p>
<ul><li>Select the file using the option below and then press the upload button.</li></ul>
</div>
<form action=\"cgi-bin/file_upload.cgi\" method=\"post\" enctype=\"multipart/form-data\">
<input type=\"file\" name=\"file\"/>
<input type=\"submit\" value=\"Upload\"/>
</form>
</div>
<div id=\"manager\"><a id=\"manage-link\" href=\"cgi-bin/manager.cgi\">Manage Labs</a></div>
<div id=\"logo\"><img src=\"style/logo.png\" alt=\"wpi_logo\" id=\"wpi\"></div>
<div id=\"footer\">
<a id=\"email\" href=\"mailto:biolabapp@wpi.edu\">Send Us Feedback</a></div>
</body>
</html>";

```

file_upload.cgi:

```

#!/usr/bin/perl -wT

# File: file_upload.cgi
# This file handles the uploading of lab XML files
# to the server, including multiple sets of error
# checking and invalid file checks.
# Authors: Nate Miller - nwmiller@wpi.edu, John Synnott - john@wpi.edu

use strict;
use CGI;
use CGI::Carp qw (fatalsToBrowser);
use File::Basename;
use warnings;

# Include scripts this script depends on for subroutines

```

```

do("/home/biolabapp/public_html/cgi-bin/extract.cgi");
do("/home/biolabapp/public_html/cgi-bin/lablist.cgi");

$CGI::POST_MAX = 1024 * 5000;           # max upload file size
my $safe_chars = "a-zA-Z0-9_-";        # valid filename characters
my $directory = "/home/biolabapp/public_html/labs"; # upload directory

my $cgi_query = new CGI;                # new CGI query
print $cgi_query->header();              # print the CGI content header

my $filename = $cgi_query->param('file'); # get the filename from the form
my ($ext) = $filename =~ /\.[^.]+$/;    # get the file extension

# Check if the filename does not exist, e.g. some error occurred
if (!$filename) {
    # GENERATE INFO PAGE AND PASS ERROR MESSAGE
    my $err_msg1 = "Error uploading file. Please try again.";
    genUploadInfoPage($err_msg1);        # generate the info page and display error message
    exit; # exit script execution
}

# Check if the file extension is .xml and if not
# cancel upload and display error to user.
elsif($ext ne ".xml") {
    # GENERATE INFO PAGE AND PASS ERROR MESSAGE
    my $err_msg2 = "Error uploading file: File is not an XML file.";
    genUploadInfoPage($err_msg2);        # generate the info page and display error message
    exit;
}

# Parse the filename and pull out the filename, path, and extension
my ($name, $path, $extension) = fileparse ($filename, '\..*');
$filename = $name.$extension; # set the filename to be the name plus and extension

# Sanitize the filename before usage
$filename =~ tr/ /_/;                    # replace all spaces with underscores
$filename =~ s/[^$safe_chars]/g;

if ($filename =~ /^[${safe_chars}+$/ {
    $filename = $1;
}
else {
    die "Filename contains invalid characters"; # display error
}

my $upload_filehandle = $cgi_query->upload('file'); # get the file handle

# Open the file handle and upload the file to the upload directory
open (UPLOADFILE, ">$directory/$filename") or die "$!";
binmode(UPLOADFILE); # read/write in binary or text mode
while (<$upload_filehandle>) {

```

```

        print UPLOADFILE;
    }

# Clean up and close the file handle(s)
close($upload_filehandle);
close(UPLOADFILE);

# Check the filesize of the file uploaded
my $filesize = -s $directory."/".$filename;
if($filesize <= 0) {
    my $err_msg3 = "Error uploading file: File is empty.";
    genUploadInfoPage($err_msg3);
    deleteFile($filename);
    exit;
}
# Otherwise, file upload is valid and proceed with lab lister file generation
else {
    # Generate the XML lab listing file
    my $labs_file = "labs.xml";
    my $labs_list = "/home/biolabapp/public_html/".$labs_file;
    my $uploaded_lab = "/home/biolabapp/public_html/labs/".$filename;

    # Extract the lab's title
    my $lab_title = titleFromXML($uploaded_lab);
    # If the lab's title cannot be extracted, delete the file and display error
    if(!$lab_title) {
        my $err_msg4 = "Unable to extract lab title. XML file is improperly formatted.";
        genUploadInfoPage($err_msg4);
        deleteFile($filename);
        exit;
    }

    # Regenerate the lab lister file after new file upload
    # to ensure the file is update to date
    regenLabLister();

    # File upload is successful if we get to this point
    my $success_msg = "File successfully uploaded!";      # Success message
    genUploadInfoPage($success_msg);                    # Generate page and display msg
}

# Generate the upload information page
# Display any messages passed to the subroutine,
# such as error or success messages.
sub genUploadInfoPage {
    # Grab the info message passed to the subroutine.
    # This message will be displayed to the user on the page,
    # e.g. for errors or success info.
    my $message = $_[0];

    # Generate the HTML and head sections

```

```

print
"<!DOCTYPE html>
<html>
<head>
<meta charset=\"UTF-8\">
<title>Lab File Uploader</title>
<link rel=\"stylesheet\" type=\"text/css\" href=\"../style/upload.css\">
</head>";

# Generate the first half of the body
print
"<body>
<div id=\"topbar\"><h1>Lab File Upload</h1></div>
<div id=\"infoBox\">
<div id=\"information\">
<h2>Upload Information</h2>";

print "<h3>$message</h3>"; # Display the message to the user

# If the upload was successful, display the filename and upload directory
if($message eq "File successfully uploaded!") {
print
"<p>Filename: $filename</p>
<p>File uploaded to: $directory/$filename</p>";
}

# Generate the rest of the page
print
"</div>
</div>
<div id=\"back\"><a id=\"back-link\" href=\"../\">Back</a></div>
<div id=\"footer\">
<a id=\"email\" href=\"mailto:biolabapp@wpi.edu\">Send Us Feedback</a></div>
</body>
</html>";
}

# Delete the specified file.
# This subroutine is used if the file is uploaded to the server,
# but is invalid (e.g. file is empty or is of proper/expected XML format).
sub deleteFile {
    my $file = $_[0]; # get the file to delete
    my $file_to_delete = $directory."/".$file;
    unlink $file_to_delete or die "Unable to delete file: $!"; # delete the file
}

```

extract.cgi:

```

#!/usr/bin/perl -wT

# File: extract.cgi

```

```

# This file's subroutine extracts the title from
# a lab XML file for usage in the lab lister file.
# Author: John Synnott - john@wpi.edu

use strict;
use CGI;
use CGI::Carp qw (fatalsToBrowser);
use File::Basename;
use warnings;

# Function to pull the lab title out of a lab XML file
sub titleFromXML {
    my $labFile = $_[0]; # get the first argument to the function.

    # Open the file and place its contents into a single variable.
    open(LFH, "<",$labFile) or die "unable to open $labFile\n";
    my $allLines;
    while (<LFH>) {
        my $currLine = $_;
        $allLines = $allLines.$currLine;
    }

    # Break the file into smaller blocks, isolating the title of the lab.
    my @firstBreak = split('title=', $allLines);
    my @secondBreak = split("", @firstBreak[1]);
    my $title = $secondBreak[0];

    if(!$title) {
        my $error = "Unable to extract lab title. XML file is improperly formatted.";
        return;
    }
    else {
        return $title;
    }
}

```

manager.cgi:

```

#!/usr/bin/perl -wT

# File: lablister.cgi
# This file generates the lab management page on
# the biolab application web interface.
# Author: Nate Miller - nwmiller@wpi.edu

use CGI;
use strict;
use warnings;

# Print the content header
my $cgi_query = new CGI;

```

```

print $cgi_query->header();

# Generate the web page content piece by piece
print "<!DOCTYPE html><html>
    <head>
    <meta charset=\"UTF-8\">
    <title>Lab File Manager</title>
    <link rel=\"stylesheet\" type=\"text/css\" href=\"../style/manager.css\">
    </head>"; # Begin HTML

print "<body>"; # Begin HTML body
print "<div id=\"topbar\"><h1>Lab File Manager</h1></div><table>";
print "<tr><th>Filename</th><th>Action</th></tr>";

my $labs_dir = "/home/biolabapp/public_html/labs";
chdir($labs_dir) or die "Error changing directories.";

# Construct the table to list the lab files on the server
foreach my $lab (<*.xml>) {
    print "<tr><td>$lab</td><td><form action=\"delete.cgi\" method=\"post\">
    <input type=\"submit\" value=\"Delete\"></input>
    <input type=\"hidden\" name=\"Deleted\" value=\"$lab\"></input>
    </form></td></tr>";
}

print "</table><div id=\"back\"><a id=\"back-link\" href=\"../\">Back</a></div>";
print "<div id=\"footer\"><a id=\"email\" href=\"mailto:biolabapp@wpi.edu\">Send Us
Feedback</a></div>";
print "</body>"; # End HTML body
print "</html>"; # End HTML

```

lablister.cgi:

```

#!/usr/bin/perl -wT

# File: lablister.cgi
# This file's subroutine regenerates the lab lister
# XML file used by the Biolab Application.
# Author: Nate Miller - nwmiller@wpi.edu

use CGI;
use strict;
use warnings;
use CGI::Carp qw (fatalsToBrowser);
use File::Basename;

# Include the lab title extraction script for usage
do("/home/biolabapp/public_html/cgi-bin/extract.cgi");

my $labs_file = "labs.xml"; # lab lister file

```

```

my $labs_list = "/home/biolabapp/public_html/" . $labs_file;      # lister file path

# Regenerates the lab lister XML file
sub regenLabLister {
    # Open the file for writing (this overwrites existing file contents)
    open(LABLIST, ">$labs_list") or die "Unable to open file for writing.";
    print LABLIST "<BIOLAB>\n";          # write start file tag

    # Change to the labs directory for processing
    my $labs_dir = "/home/biolabapp/public_html/labs/";
    chdir($labs_dir) or die "Error changing directories.";

    # Find each XML file in the labs directory and process it
    foreach my $lab (<*.xml>) {
        # Extract the lab's title
        my $lab_title = titleFromXML($labs_dir.$lab);
        my $lab_url = "http://users.wpi.edu/~biolabapp/labs/" . $lab;
        # Add the labfile entry to the lab lister file
        print LABLIST "<LABFILE title='$lab_title'>$lab_url</LABFILE>\n";
    }

    print LABLIST "</BIOLAB>"; # write the end file tag
    close(LABLIST);          # close the file handle
}

```

delete.cgi:

```

#!/usr/bin/perl -wT

# File: delete.cgi
# This file deletes lab XML files that are currently
# stored on the server when initiated from the manage labs page.
# Authors: Nate Miller - nwmiller@wpi.edu

use CGI;
use strict;
use warnings;

# Include the lab lister script for usage in lab lister file regen
do("/home/biolabapp/public_html/cgi-bin/lablisterscript.cgi");

# Print the content header info
my $cgi_query = new CGI;
print $cgi_query->header();

# Define the valid characters for filenames
my $safe_chars = "a-zA-Z0-9_-.";

my $filename = $cgi_query->param('Deleted');

# Sanitize the filename before usage for deletion

```



```

$filename =~ tr/ /_/;          # replace all spaces with underscores
$filename =~ s/[^$safe_chars]//g;

if ($filename =~ /^[${safe_chars}+]$/) {
    $filename = $1;
}
else {
    die "Filename contains invalid characters";    # display error
}

my $file = $filename;    # store sanitized filename in new variable
my $file_to_delete = "/home/biolabapp/public_html/labs/" . $file; # append filename to full file path

# Delete the file specified by the full file path
if(unlink $file_to_delete) {
    my $mess1 = "Successfully deleted $file.";
    regenLabLister();
    genDeleteInfoPage($mess1);
} else {
    my $mess2 = "Unable to delete $file: $!";
    genDeleteInfoPage($mess2);
}

# Generate the file deletion information page
# that displays info about the file's deletion.
sub genDeleteInfoPage {
    my $message = $_[0];    # Get the passed in info message

    # Generate the HTML and head sections
    print
    "<!DOCTYPE html>
    <html>
    <head>
    <meta charset='UTF-8'>
    <title>Lab File Manager</title>
    <link rel='stylesheet' type='text/css' href='../style/delete.css'>
    </head>";

    # Generate the first half of the body
    print
    "<body>
    <div id='topbar'><h1>Manager Information</h1></div>
    <div id='infoBox'>
    <div id='information'>
    <h2>Deletion Information</h2>";

    print "<h3>$message</h3>";    # Display the message to the user

    # Generate the rest of the page
    print
    "</div>";
}

```

```

    </div>
    <div id=\"back\"><a id=\"back-link\" href=\"./manager.cgi\">Back</a></div>
    <div id=\"footer\">
    <a id=\"email\" href=\"mailto:biolabapp@wpi.edu\">Send Us Feedback</a></div>
    </body>
    </html>";
}

```

Appendix F: Biolab Application Source Code

AppDelegate.h

```

//
// AppDelegate.h
// BioLab
//
// Created by Nicholas Morin on 3/17/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//

```

```
#import <UIKit/UIKit.h>
```

```
@interface AppDelegate : UIResponder <UIApplicationDelegate>
```

```
@property (strong, nonatomic) UIWindow *window;
```

```
@end
```

AppDelegate.m

```

//
// AppDelegate.m
// BioLab
//
// Created by Nicholas Morin on 3/17/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//

```

```
#import "AppDelegate.h"
```

```
@implementation AppDelegate
```

```

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Override point for customization after application launch.
    UISplitViewController *splitViewController = (UISplitViewController
*)self.window.rootViewController;
    UINavigationController *navigationController = [splitViewController.viewControllers
lastObject];
    splitViewController.delegate = (id)navigationController.topViewController;
}

```

```

    return YES;
}

- (void)applicationWillResignActive:(UIApplication *)application
{
    // Sent when the application is about to move from active to inactive state. This can occur for
    // certain types of temporary interruptions (such as an incoming phone call or SMS message) or
    // when the user quits the application and it begins the transition to the background state.
    // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES
    // frame rates. Games should use this method to pause the game.
}

- (void)applicationDidEnterBackground:(UIApplication *)application
{
    // Use this method to release shared resources, save user data, invalidate timers, and store
    // enough application state information to restore your application to its current state in case it is
    // terminated later.
    // If your application supports background execution, this method is called instead of
    // applicationWillTerminate: when the user quits.
}

- (void)applicationWillEnterForeground:(UIApplication *)application
{
    // Called as part of the transition from the background to the inactive state; here you can undo
    // many of the changes made on entering the background.
}

- (void)applicationDidBecomeActive:(UIApplication *)application
{
    // Restart any tasks that were paused (or not yet started) while the application was inactive. If
    // the application was previously in the background, optionally refresh the user interface.
}

- (void)applicationWillTerminate:(UIApplication *)application
{
    // Called when the application is about to terminate. Save data if appropriate. See also
    // applicationDidEnterBackground:.
}

@end

```

DetailViewController.h

```

//
// DetailViewController.h
// BioLab
//
// Created by Nicholas Morin on 3/17/13.

```

```

// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//

#import <UIKit/UIKit.h>

#import "MasterViewController.h"
#import "DataTableView.h"
#import "HTMLTableParser.h"
#import "TextView.h"
#import "ImageView.h"
#import "VideoView.h"

#define SPACE_BETWEEN_FEATURE 10

@interface DetailViewController : UIViewController <UISplitViewControllerDelegate,
UIAlertViewDelegate>

@property (weak, nonatomic) IBOutlet UILabel *detailDescriptionLabel;
@property (strong, nonatomic) MasterViewController * master;
@property (strong, nonatomic) NSString * labName;
@property (strong, nonatomic) Entry * currentEntry;

-(void)addMenuButton;
-(void)setupLab:(LabFile *)lab;
-(void)loadTab:(Entry *)entry;
-(BOOL)saveChangesToEntry;
@end

DetailViewController.m
//
// DetailViewController.m
// BioLab
//
// Created by Nicholas Morin on 3/17/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//
// Main View Controller for displaying tabs

#import "DetailViewController.h"

@interface DetailViewController ()
@property (strong, nonatomic) UIPopoverController *masterPopoverController;
@end

@implementation DetailViewController

@synthesize master, labName, currentEntry;

```

```

#pragma mark - Managing the detail item

// save title and labName, creates a MainMenu button
-(void)setupLab:(LabFile *)lab
{
    self.labName = lab.title;
    self.title = lab.title;
    [self addMenuButton];
}

// load a new tab with the given Entry in the view
// removes the previous entry from the view
-(void)loadTab:(Entry *)entry
{
    // update tab name
    NSMutableString * detailName = [[NSMutableString alloc] init];
    if(self.labName == nil) { // no labName since not in a lab
        [detailName appendString:@"BioLab"];
    } else if(entry.title == nil) { // not inside a section of the lab
        [detailName appendString:self.labName];
    } else {
        [detailName appendString:self.labName];
        [detailName appendString:@" : "];
        [detailName appendString:entry.title];
    }
    self.title = detailName;
    // save data of previous tab
    [self saveChangesToEntry];
    // clear the view of all subviews
    [self.view.subviews makeObjectsPerformSelector: @selector(removeFromSuperview)];
    // keep track of the current entry being displayed
    self.currentEntry = entry;
    float heightSoFar = 5; // start the entry a few pixels below top of screen
    float SCREEN_WIDTH = self.view.frame.size.width; // get the width of the screen
    UIScrollView * scrollView = [[UIScrollView alloc] initWithFrame:CGRectMake(0, 0,
self.view.frame.size.width, self.view.frame.size.height)];
    [self.view addSubview:scrollView];
    // loop through all the features in the entry and create FeatureViews for each one
    for(Feature * feature in entry.features) {
        if([feature.type isEqualToString:TEXT]) {
            // create a new textView and add it to the scroll view as non-editable text
            UITextView * textView = [[UITextView alloc] init];
            heightSoFar = [textView addTextFeature:feature toScrollView:scrollView
withHeight:heightSoFar withWidth:SCREEN_WIDTH];
        } else if ([feature.type isEqualToString:IMAGE]) {
            // create a new imageView and add it to the scroll view

```

```

    UIImageView * imageView = [[UIImageView alloc] init];
    heightSoFar = [imageView addImageFeature:feature withScrollView:scrollView
withHeight:heightSoFar withWidth:SCREEN_WIDTH];
    } else if ([feature.type isEqualToString:VIDEO]) {
        // create a new videoView and add it to the scroll view
        VideoView * vidView = [[VideoView alloc] init];
        heightSoFar = [vidView addVideoFeature:feature toScrollView:scrollView
withHeight:heightSoFar withWidth:SCREEN_WIDTH];
    } else if ([feature.type isEqualToString:INPUT]) {
        // create a new textView and add it to thte scroll view as editable text
        TextView * textView = [[TextView alloc] init];
        heightSoFar = [textView addQuestionFeature:feature toScrollView:scrollView
withHeight:heightSoFar withWidth:SCREEN_WIDTH];
    } else if ([feature.type isEqualToString:TABLE]) {
        // create a new DataTableView and add it to the scroll view
        // DataTableView sets it's own height
        DataTableView * table = [[DataTableView alloc] initWithFrame:CGRectMake(0,
heightSoFar, SCREEN_WIDTH, 0) withFeature:feature];
        [scrollView addSubview:table];
        heightSoFar += table.frame.size.height + SPACE_BETWEEN_FEATURE;
    }
}
// set the size of the size of the scroll view to the size of the data
[scrollView setContentSize:CGSizeMake(SCREEN_WIDTH, heightSoFar)];
//[scrollView
setAutoresizingMask:UIViewAutoresizingFlexibleWidth|UIViewAutoresizingFlexibleHeight];
[scrollView setScrollEnabled:YES];
}

// Loop through entry saving data changes inside of currentEntry
- (BOOL)saveChangesToEntry
{
    BOOL success = true;
    for(int i = 0; i < currentEntry.features.count; i++) {
        // assumes view has a scrollview and all the entry data is stored inside of the scrollview
        Feature * feature = [currentEntry getFeatureAtIndex:i];
        if([feature.type isEqualToString:INPUT]) {
            success &= [((TextView *)(((UIScrollView *)[self.view.subviews
objectAtIndex:0]).subviews objectAtIndex:i)) saveChanges];
        } else if([feature.type isEqualToString:TABLE]) {
            success &= [((DataTableView *)(((UIScrollView *)[self.view.subviews
objectAtIndex:0]).subviews objectAtIndex:i)) saveChanges];
        }
    }
    return success;
}
}

```

```

- (void)viewDidLoad
{
    [super viewDidLoad];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

// creates a menu button in top right of DetailViewController that
// calls verifyReturnToMenu when pressed
- (void)addMenuButton
{
    UIBarButtonItem * menuButton = [[UIBarButtonItem alloc] initWithTitle:@"Main Menu"
style:UIBarButtonItemStyleBordered target:self action:@selector(verifyReturnToMenu)];
    [self.navigationItem setRightBarButtonItem:menuButton animated:YES];
}

// creates an alertView that asks if user wants to continue to menu to prevent data loss
- (void)verifyReturnToMenu
{
    UIAlertView * alertView = [[UIAlertView alloc] initWithTitle:@"Are You Sure?"
message:@"All data will be lost for current lab." delegate:self cancelButtonTitle:@"Cancel"
otherButtonTitles:@"Continue To Menu", nil];
    [alertView show];
}

// called when alertView button is pressed
- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex
{
    if(buttonIndex == 1) {
        [master resetApplication];
    }
}

#pragma mark - Split view

- (void)splitViewController:(UISplitViewController *)splitController
willHideViewController:(UIViewController *)viewController
withBarButtonItem:(UIBarButtonItem *)barButtonItem
forPopoverController:(UIPopoverController *)popoverController
{
    barButtonItem.title = @"Menu"; // Set menu button label
    [self.navigationItem setLeftBarButtonItem:barButtonItem animated:YES];
}

```

```

    self.masterPopoverController = popoverController;
}

- (void)splitViewController:(UISplitViewController *)splitController
willShowViewController:(UIViewController *)viewController
invalidatingBarButtonItem:(UIBarButtonItem *)barButtonItem
{
    // Called when the view is shown again in the split view, invalidating the button and popover
    controller.
    [self.navigationItem setLeftBarButtonItem:nil animated:YES];
    self.masterPopoverController = nil;
}

@end

```

MasterViewController.h

```

//
// MasterViewController.h
// BioLab
//
// Created by Nicholas Morin on 3/17/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//

```

```

#import <UIKit/UIKit.h>
#import <MessageUI/MFMailComposeViewController.h>
#import "LabFile.h"
#import "XMLParser.h"

```

```

#define BIOLABHOME @"http://users.wpi.edu/~biolabapp/labs.xml"

```

```

@class DetailViewController;

```

```

@interface MasterViewController : UITableViewController
<MFMailComposeViewControllerDelegate>

```

```

@property (strong, nonatomic) DetailViewController *detailViewController;
@property (strong, nonatomic) NSArray * tabList;
@property (strong, nonatomic) LabFile * labFile;
@property (strong, nonatomic) TitlePage * titlePage;
@property (nonatomic) BOOL inFile;
@property (strong, nonatomic) Entry * mainMenuEntry;

```

```

- (void)resetApplication;
@end

```

MasterViewController.m


```

//
// MasterViewController.m
// BioLab
//
// Created by Nicholas Morin on 3/17/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//
// View Controller for controlling table of contents and telling DetailViewController
// what to display

#import "MasterViewController.h"
#import "DetailViewController.h"

@interface MasterViewController () {
    NSMutableArray *_objects;
}
@end

@implementation MasterViewController

@synthesize tabList, labFile, titlePage, inFile, mainMenuEntry;

- (void)awakeFromNib
{
    self.clearsSelectionOnWillAppear = NO;
    self.contentSizeForViewInPopover = CGSizeMake(320.0, 600.0);
    [super awakeFromNib];
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    [self setupLabFile:BIOLABHOME];
    // get a list of all the lab names to display
    NSMutableArray * temp = [[NSMutableArray alloc] init];
    for(int i = 0; i < titlePage.labList.count; i ++) {
        [temp addObject:[titlePage getFeatureAtIndex:i].type];
    }
    // set the labList
    tabList = [[NSArray alloc] initWithArray:temp];

    self.detailViewController = (DetailViewController
*)[[self.splitViewController.viewControllers lastObject] topViewController];
    self.detailViewController.master = self;
    [self.tableView reloadData];
    [self.detailViewController.navigationItem setRightBarButtonItem:nil animated:YES];
    inFile = NO;
}

```

```

mainMenuEntry = [[Entry alloc] init];

// create a feature for the title page of the app
Feature * f = [[Feature alloc] init];
f.type = TEXT;
[f.data appendString:@"Welcome to Bio Lab!\nSelect a lab from the menu to get started" ];
self.detailViewController.labName = nil;
[mainMenuEntry.features addObject:f];
[self.detailViewController loadTab:mainMenuEntry];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (void)insertNewObject:(id)sender
{
    if (!_objects) {
        _objects = [[NSMutableArray alloc] init];
    }
    [_objects addObject:[NSDate date] atIndex:0];
    NSIndexPath *indexPath = [NSIndexPath indexPathForRow:0 inSection:0];
    [self.tableView insertRowsAtIndexPaths:@[indexPath]
    withRowAnimation:UITableViewRowAnimationAutomatic];
}

- (void)setupLabFile:(NSString *)name
{
    // files from a server
    NSURL * path = [NSURL URLWithString:name];
    NSData * data = [NSData dataWithContentsOfURL:path];
    NSXMLParser * xmlParser = [[NSXMLParser alloc] initWithData:data];
    XMLParser * parser = [[XMLParser alloc] init];
    [xmlParser setDelegate:parser];
    BOOL worked = [xmlParser parse];

    // if parsing was successful, save labfile and title page data
    if(worked) {
        if(parser.labFile != nil) {
            self.labFile = parser.labFile;
        }
        if(parser.titlePage != nil) {
            self.titlePage = parser.titlePage;
        }
    }
}

```

```

// if parse was unsuccessful, create a popup alert letting the user know about an error
else {
    // create an error popup if unable to load
    UIAlertView * aView = [[UIAlertView alloc] initWithTitle:@"Unable to Load Data From
Server" message:@"Please try again later." delegate:nil cancelButtonTitle:@"OK"
otherButtonTitles:nil];
    [aView show];
}
}

```

```

#pragma mark - Table View

```

```

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}

```

```

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    return tabList.count;
}

```

```

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"Cell"
forIndexPath:indexPath];

    cell.textLabel.text = [tabList objectAtIndex:indexPath.row];
    return cell;
}

```

```

// Reset the application (reload the mainMenu)
- (void)resetApplication
{
    [self.detailViewController.navigationItem setRightBarButtonItem:nil animated:YES];
    [self.detailViewController loadTab:mainMenuEntry];
    [self viewDidLoad];
}

```

```

// what to do when a tab is selected
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
*)indexPath
{
    // if not inside of a file (at the home screen)
    if(!inFile) {

```

```

NSString *name = [tabList objectAtIndex:indexPath.row];
[self setupLabFile:[titlePage getDataWithName:name]];
[self.detailViewController setupLab:labFile];
NSMutableArray * temp = [[NSMutableArray alloc] init];
for(int i = 0; i < labFile.entries.count; i++) {
    [temp addObject:((Entry *)[labFile.entries objectAtIndex:i]).title];
}
tabList = [[NSArray alloc] initWithArray:temp];
[self.tableView reloadData];
inFile = YES;
Entry * e = [[Entry alloc] init];
Feature * f = [[Feature alloc] init];
f.type = TEXT;
[f.data appendString:name];
[e.features addObject:f];
[self.detailViewController loadTab:e];
} else { // inside of a lab now so show body of lab
    Entry * entry = [labFile getEntryAtIndex:indexPath.row];
    // display tab
    [self.detailViewController loadTab:entry];
    // if export tab, create email
    if([entry.title isEqualToString:EXPORT]) {
        [self export];
    }
}
}
}

// what to do after the export email has been sent/canceled
- (void)mailComposeController:(MFMailComposeViewController *) controller
didFinishWithResult:(MFMailComposeResult) result error:(NSError *)error
{
    //Dismiss the MailController.
    [self dismissViewControllerAnimated:YES completion:NULL];
}

// create an email with questions and tables as attachments to the email
- (void)export
{
    if([MFMailComposeViewController canSendMail]) {
        MFMailComposeViewController *mailViewController =
[[MFMailComposeViewController alloc] init];
        mailViewController.mailComposeDelegate = self;

        // string for title of questions.txt
        NSMutableString * questionsTitle = [[NSMutableString alloc] initWithString:labFile.title];
        // string with all questions data

```

```

NSMutableString * questions = [[NSMutableString alloc] init];
// loop through all the entries in the lab looking to pull out tables and questions
for(Entry * e in labFile.entries ) {
    // loop through each feature in the entry looking for tables and questions
    for(Feature * f in e.features) {
        if([f.type isEqualToString:INPUT]) {
            // add the data to the questions file
            [questions appendString:f.data];
            [questions appendString:@"\n\n"];
        } else if ([f.type isEqualToString:TABLE]) {
            // turn the table into valid csv format
            NSMutableString * tableData = [[NSMutableString alloc] init];
            for(NSArray * row in f.tableData) {
                for(NSString * data in row) {
                    // replace break tags with spaces and @BLANKS with empty strings
                    [tableData appendString:[data
stringByReplacingOccurrencesOfString:@"<br>" withString:@" "]
stringByReplacingOccurrencesOfString:@"@BLANK" withString:@""];
                    // append commas between data
                    [tableData appendString:@","];
                }
                // remove the last comma in the list
                [tableData deleteCharactersInRange:NSMakeRange([tableData length]-1, 1)];
                [tableData appendString:@"\n"];
            }
            // add an attachment for the table
            NSMutableString * tableTitle = [[NSMutableString alloc] initWithString:f.data];
            [tableTitle appendString:@".csv"];
            [mailViewController addAttachmentData:[tableData
dataUsingEncoding:NSUTF8StringEncoding] mimeType:@"text/csv" fileName:tableTitle];
        }
    }
    // add an attachment for the questions
    [questionsTitle appendString:@"-Questions.txt"];
    [mailViewController addAttachmentData:[questions
dataUsingEncoding:NSUTF8StringEncoding] mimeType:@"text/plain"
fileName:questionsTitle];
    // set subject and body of the message
    NSMutableString * subject = [[NSMutableString alloc] initWithString:@"Lab Data : "];
    [subject appendString:labFile.title];
    [mailViewController setSubject:subject];
    [mailViewController setMessageBody:subject isHTML:NO];

    // present the email to the viewer
    [self presentViewController:mailViewController animated:YES completion:NULL];
} else {

```

```

        // create an error popup if unable to send email
        UIAlertView * aView = [[UIAlertView alloc] initWithTitle:@"Unable to send email."
message:@"Please try again later." delegate:nil cancelButtonTitle:@"OK"
otherButtonTitles:nil];
        [aView show];
    }
}

@end

```

DataTableView.h

```

//
// DataTableView.h
// BioLab
//
// Created by Nicholas Morin on 4/16/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "Feature.h"
#import "HTMLTableParser.h"

@interface DataTableView : UIWebView

@property (strong, nonatomic)Feature * displayFeature; // feature to display

- (id)initWithFrame:(CGRect)frame withFeature:(Feature *)feature;
- (BOOL)saveChanges;

@end

```

DataTableView.m

```

//
// DataTableView.m
// BioLab
//
// Created by Nicholas Morin on 4/16/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//
// Class for displaying html tables inside of a webview

#import "DataTableView.h"

@implementation DataTableView

@synthesize displayFeature;

```

```

// height of given frame is ignored
// (sets it's own height based on given feature)
- (id)initWithFrame:(CGRect)frame withFeature:(Feature *)feature
{
    self = [super initWithFrame:frame];
    if (self) {
        displayFeature = feature;
        [self drawTable];
    }
    return self;
}

// create an editable HTML table inside of the webview (self)
- (void)drawTable
{
    int numRows = displayFeature.tableData.count;
    int numCols = ((NSArray *)[displayFeature.tableData objectAtIndex:0]).count;
    int titleHeight = 20;
    int rowHeight = 50;

    // determine the height of the table from amount of data needed
    self.frame = CGRectMake(self.frame.origin.x, self.frame.origin.y, self.frame.size.width,
titleHeight*2 + numRows * rowHeight);

    // setup style for table (100% width of screen with fixed cell width)
    // height based on size determined above
    NSMutableString * table = [[NSMutableString alloc] init];
    [table appendString:[NSString stringWithFormat:@"<table style=\"table-layout:fixed;\"
border=\"1\" width=\"100%%\" height=\"%i\">", ((int)self.frame.size.height - titleHeight*2)]];

    // add the title of the table in a caption above the table
    [table appendString:[NSString stringWithFormat:@"<caption style=\"font-
size:%ipx\">% @</caption>", titleHeight, displayFeature.data]];

    // for each row, open a new row tag
    for (int i = 0; i < numRows; i++) {
        [table appendString:@"<tr>"];

        // for each column in a row, open a new data tag
        for(int j = 0; j < numCols;j++) {
            NSString * val = [displayFeature dataInColumn:j andRow:i];
            // make text centered and editable
            [table appendString:@"<td style=\"text-align:center;\"contenteditable>"];

            // if text needs to be blank, make cell empty
            if ([val isEqualToString:@"@BLANK"]) {

```

```

        [table appendString:@""];
    } else {
        [table appendString:val];
    }
    [table appendString:@"</td>"];
}
[table appendString:@"</tr>"];
}
[table appendString:@"</table>"];
// load the new table in the view
[self loadHTMLString:table baseURL:nil];
}

// save changes to the feature
// returns whether or not save was successful
- (BOOL)saveChanges
{
    NSString * htmlData = [self
stringByEvaluatingJavaScriptFromString:@"document.documentElement.outerHTML"];
    // make sure break tags are closed to confine to xml formatting (since HTMLTableParser is
just an xml parser
    htmlData = [htmlData stringByReplacingOccurrencesOfString:@"<br>" withString:@"<br
/>"];

    // create a new HTMLTableParser and parse the table data
    HTMLTableParser * tableParse = [[HTMLTableParser alloc] init];
    NSData * data = [htmlData dataUsingEncoding:NSUTF8StringEncoding];
    NSXMLParser * xmlParser = [[NSXMLParser alloc] initWithData:data];
    [xmlParser setDelegate:tableParse];
    BOOL worked = [xmlParser parse];

    // if parse was successful save the parsed data
    if(worked) {
        displayFeature.tableData = tableParse.tableData;
        return YES;
    } else {
        return NO;
    }
}
@end

```

ImageView.h

```

//
// ImageView.h
// BioLab
//
// Created by Nicholas Morin on 4/21/13.

```



```

// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "Feature.h"
#import "DetailViewController.h"

@interface ImageView : UIImageView

@property (strong, nonatomic) Feature * displayFeature;

- (float)addImageFeature:(Feature *)feature withScrollView:(UIScrollView *)scrollView
withHeight:(float)height withWidth:(float)width;

@end

ImageView.m
//
// ImageView.m
// BioLab
//
// Created by Nicholas Morin on 4/21/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//
// Class for displaying an Image feature

#import "ImageView.h"

@implementation ImageView

@synthesize displayFeature;

- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code
    }
    return self;
}

// add a UIImageView to the given scroll view
// returns the new total height
- (float)addImageFeature:(Feature *)feature withScrollView:(UIScrollView *)scrollView
withHeight:(float)height withWidth:(float)width
{
    displayFeature = feature;
}

```

```

    NSURL * imageURL = [NSURL URLWithString:[feature.data
stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]];
    NSData * imageData = [NSData dataWithContentsOfURL:imageURL];
    UIImage * image = [UIImage imageWithData:imageData];
    self.image = image;
    [self sizeToFit];
    [scrollView addSubview:self];
    CGRect frame = self.frame;
    // if image is too wide, scale image width and height to match screen width
    if(self.frame.size.width > width) {
        self.contentMode = UIViewContentModeScaleAspectFit;
        frame.size.width = width;
        frame.size.height = self.frame.size.height * width / self.frame.size.width;
    } else {
        frame.size.width = self.frame.size.width;
        frame.size.height = self.frame.size.height;
    }
    // put picture in the center of the screen
    frame.origin.x = width/2 - frame.size.width/2;
    frame.origin.y = height;
    self.frame = frame;
    return height + self.frame.size.height + SPACE_BETWEEN_FEATURE;
}

```

@end

TextView.h

```

//
// TextView.h
// BioLab
//
// Created by Nicholas Morin on 4/21/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "Feature.h"
#import "DetailViewController.h"

@interface TextView : UITextView

@property (strong, nonatomic) Feature * displayFeature; // feature for display

- (float)addTextFeature:(Feature *)feature toScrollView:(UIScrollView *)scrollView
withHeight:(float)height withWidth:(float)width;
- (float)addQuestionFeature:(Feature *)feature toScrollView:(UIScrollView *)scrollView
withHeight:(float)height withWidth:(float)width;

```

```
-(BOOL)saveChanges;
@end
```

TextView.m

```
//
// TextView.m
// BioLab
//
// Created by Nicholas Morin on 4/21/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//
// Class for displaying display text and questions inside of a UITextView

#import "TextView.h"

@implementation TextView

@synthesize displayFeature;

// add a UITextView to the given scroll view (non editable)
// returns the new total height
- (float)addTextFeature:(Feature *)feature toScrollView:(UIScrollView *)scrollView
withHeight:(float)height withWidth:(float)width
{
    self.displayFeature = feature;
    self.text = feature.data;
    [self setFont:[UIFont systemFontOfSize:17]];
    [self setEditable:NO];
    [scrollView addSubview:self];
    self.frame = CGRectMake(0,0,width,0); // set width before setting height
    self.frame = CGRectMake(0,height,width, self.contentSize.height);
    return height + self.contentSize.height + SPACE_BETWEEN_FEATURE;
}

// add a UITextView to the given scroll view (editable)
// returns the new total height
- (float)addQuestionFeature:(Feature *)feature toScrollView:(UIScrollView *)scrollView
withHeight:(float)height withWidth:(float)width
{
    self.displayFeature = feature;
    self.text = feature.data;
    UIFont * font = [UIFont systemFontOfSize:17];
    [self setFont:font];
    [self setEditable:YES];
    self.backgroundColor = [UIColor colorWithRed:255/255.0 green:251/255.0 blue:204/255.0
alpha:1];
    [scrollView addSubview:self];
}
```

```

self.frame = CGRectMake(0,0,width,0); // set width before setting height
// add 3 lines of extra height for response entry
// (View can be scrolled through if more data is entered and textView will
// resize on reload to better fit the data and still have 3 extra lines of space)
self.frame = CGRectMake(0,height,width, self.contentSize.height + font.lineHeight * 3);
return height + self.frame.size.height + SPACE_BETWEEN_FEATURE;
}

// save changes to feature
-(BOOL)saveChanges
{
    // if the text is editable then save possible changes
    if(self.editable) {
        displayFeature.data = [[NSMutableString alloc] initWithString:self.text];
    }
    return true;
}

@end

```

VideoView.h

```

//
// VideoView.h
// BioLab
//
// Created by Nicholas Morin on 4/21/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//

```

```

#import <UIKit/UIKit.h>
#import "Feature.h"
#import "DetailViewController.h"

```

```

@interface VideoView : UIWebView

```

```

@property (strong, nonatomic) Feature * displayFeature;

```

```

- (float)addVideoFeature:(Feature *)feature toScrollView:(UIScrollView *)scrollView
withHeight:(float)height withWidth:(float)width;

```

```

@end

```

VideoView.m

```

//
// VideoView.m
// BioLab
//

```

```

// Created by Nicholas Morin on 4/21/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//
// Class for loading a video in an HTML IFrame inside of a UIWebView

#import "VideoView.h"

@implementation VideoView
@synthesize displayFeature;

- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code
    }
    return self;
}

// add a UIWebView to the given scroll view and load a youtube video in the view
// returns the new total height
- (float)addVideoFeature:(Feature *)feature toScrollView:(UIScrollView *)scrollView
withHeight:(float)height withWidth:(float)width
{
    displayFeature = feature;
    int videoWidth = 570; // hardcoded. should be dynamically set for extensability
    int videoHeight = 325; // hardcoded. should be dynamically set for extensability
    self.frame = CGRectMake(width/2 - videoWidth/2, height, videoWidth, videoHeight);
    // HTML to embed YouTube video
    NSString * iFrame = @"<iframe width=\"560\" height=\"315\" src=\"% @\"
frameborder=\"0\" allowfullscreen></iframe>"; // height and width hardcoded to the minimum
embedded video dimensions. should be dynamically set for extensability
    // add url to iFrame
    NSString * videoURL = [NSString stringWithFormat:iFrame, feature.data];
    // Load the html into the webview
    [self loadHTMLString:videoURL baseURL:nil];
    [scrollView addSubview:self];
    return height + self.frame.size.height + SPACE_BETWEEN_FEATURE;
}

@end
Entry.h
//
// Entry.h
// BioLab
//
// Created by Nicholas Morin on 3/24/13.

```

```

// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "Feature.h"

@interface Entry : NSObject

@property (strong, nonatomic) NSString * title; // title of entry
@property (strong, nonatomic) NSMutableArray * features; // list of features in entry

-(Feature *)getFeatureAtIndex:(NSInteger)index;

@end

Entry.m
//
// Entry.m
// BioLab
//
// Created by Nicholas Morin on 3/24/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//
// Class for holding all information associated with one tab in a lab

#import "Entry.h"

@implementation Entry

@synthesize title, features;

-(id)init
{
    self = [super init];
    features = [[NSMutableArray alloc] init];
    return self;
}

// get feature at given index
-(Feature *)getFeatureAtIndex:(NSInteger)index
{
    return (Feature *)[features objectAtIndex:index];
}

@end

Feature.h
//

```

```

// Feature.h
// BioLab
//
// Created by Nicholas Morin on 3/24/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface Feature : NSObject

@property (strong, nonatomic) NSString * type; // type of feature (ie text/table/input/...)
@property (strong, nonatomic) NSMutableString * data; // data for all types except table (name
of table if type is table)
@property (strong, nonatomic) NSMutableArray * tableData; // 2d array for storing tables (only
used if type is table)

// retrieve data in given row and column
-(NSString *)dataInColumn:(int)col andRow:(int)row;

@end

Feature.m
//
// Feature.m
// BioLab
//
// Created by Nicholas Morin on 3/24/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//
// Class used to store information associated with one feature inside of an entry (ie a text entry /
table / text input / image / video/ ...)

#import "Feature.h"

@implementation Feature

@synthesize data, type, tableData;

- (id)init
{
    self = [super init];
    self.data = [[NSMutableString alloc] init];
    return self;
}

// get data from given col and index

```

```
// returns nil if no table data exists
-(NSString *)dataInColumn:(int)col andRow:(int)row
{
    NSString * val;
    if(tableData == nil) {
        val = nil;
    } else {
        val = [[(NSArray *)[self.tableData objectAtIndex:row]] objectAtIndex:col];
    }
    return val;
}
}
```

@end

LabFile.h

```
//
// LabFile.h
// BioLab
//
// Created by Nicholas Morin on 3/24/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//
```

```
#import <Foundation/Foundation.h>
#import "Entry.h"
```

```
#define EXPORT @"Export Data"
```

```
@interface LabFile : NSObject
```

```
@property (strong, nonatomic) NSString * title; // title of the lab
@property (strong, nonatomic) NSMutableArray * entries; // list of entries
```

```
-(Entry *)getEntryAtIndex:(NSInteger)index;
-(void)addExportTab;
```

@end

LabFile.m

```
//
// LabFile.m
// BioLab
//
// Created by Nicholas Morin on 3/24/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//
// Class for holding all data associated with a lab
```



```

#import "LabFile.h"

@implementation LabFile

@synthesize title, entries;

-(id)init
{
    self = [super init];
    entries = [[NSMutableArray alloc] init];
    return self;
}

// get entry at given index
-(Entry *)getEntryAtIndex:(NSInteger)index
{
    return (Entry *) [entries objectAtIndex:index];
}

// add an export tab entry to entries
-(void)addExportTab
{
    Entry * exportTab = [[Entry alloc] init];
    exportTab.title = EXPORT;
    [entries addObject:exportTab];
}

@end

```

TitlePage.h

```

//
// TitlePage.h
// BioLab
//
// Created by Nicholas Morin on 3/24/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//

```

```

#import <Foundation/Foundation.h>
#import "Feature.h"

```

```

@interface TitlePage : NSObject

```

```

@property (strong, nonatomic) NSMutableArray * labList; // list of labs in title page

```

```

-(Feature *)getFeatureAtIndex:(NSInteger)index;

```

```
-(NSString *)getDataWithName:(NSString *)name;
```

```
@end
```

TitlePage.m

```
//
```

```
// TitlePage.m
```

```
// BioLab
```

```
//
```

```
// Created by Nicholas Morin on 3/24/13.
```

```
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
```

```
//
```

```
// Class used to store the information of the BioLab title page
```

```
#import "TitlePage.h"
```

```
@implementation TitlePage
```

```
@synthesize labList;
```

```
-(id)init
```

```
{
```

```
    self = [super init];
```

```
    labList = [[NSMutableArray alloc] init];
```

```
    return self;
```

```
}
```

```
// get feature at given index
```

```
-(Feature *)getFeatureAtIndex:(NSInteger)index
```

```
{
```

```
    return (Feature *)[labList objectAtIndex:index];
```

```
}
```

```
// get the data (URL) associated with the given lab name
```

```
-(NSString *)getDataWithName:(NSString *)name
```

```
{
```

```
    NSString * data;
```

```
    for(Feature * f in labList) {
```

```
        if([name isEqualToString:f.type]) {
```

```
            data = f.data;
```

```
            break;
```

```
        }
```

```
}
```

```
    return data;
```

```
}
```

```

@end

HTMLTableParser.h
//
// HTMLTableParser.h
// BioLab
//
// Created by Nicholas Morin on 4/18/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//

#import <Foundation/Foundation.h>

#define TR @"tr"
#define TD @"td"
#define BR @"br"
#define HTMLTABLE @"table"

@interface HTMLTableParser : NSObject <NSXMLParserDelegate>

@property (strong, nonatomic) NSString * recentTag; // stores most recent open tag found
@property (strong, nonatomic) NSMutableArray * tableData; // used for creating new table data
from html data being parsed
@property int rowNum, colNum; // stores current row and column number

@end

HTMLTableParser.m
//
// HTMLTableParser.m
// BioLab
//
// Created by Nicholas Morin on 4/18/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//
// Class for parsing html table data for DataTableView

#import "HTMLTableParser.h"

@implementation HTMLTableParser

@synthesize recentTag, tableData, rowNum, colNum;

// called when open tag is found
-(void) parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
attributes:(NSDictionary *)attributeDict

```

```

{
    // ignore line break tags since internal to data from other labs
    if(![elementName isEqualToString:BR]) {
        recentTag = elementName;
    }
    if([elementName isEqualToString:HTMLTABLE]) {
        tableData = [[NSMutableArray alloc] init];
        rowNum = 0;
    } else if([elementName isEqualToString:TR]) { // ROW tag
        colNum = 0;
        [tableData addObject:[NSMutableArray alloc] init];
    } else if([elementName isEqualToString:TD]) { // COLUMN tag
        [[tableData objectAtIndex:rowNum] addObject:@""];
    }
}

// called when data is found between tags
-(void) parser:(NSXMLParser *)parser foundCharacters:(NSString *)string
{
    if([recentTag isEqualToString:TD]) {
        NSString * prevData = [[tableData objectAtIndex:rowNum] objectAtIndex:colNum];
        // if there was no data, then store new data
        if([prevData isEqualToString:@""]) {
            [[tableData objectAtIndex:rowNum] replaceObjectAtIndex:colNum withObject:string];
        }
        // if there was data, append a break tag and add new data to old data
        else {
            NSMutableString * newData = [[NSMutableString alloc] initWithString:prevData];
            [newData appendString:@"<br>"];
            [newData appendString:string];
            [[tableData objectAtIndex:rowNum] replaceObjectAtIndex:colNum
withObject:newData];
        }
    }
}

// called when closed tag is found
-(void) parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
{
    if([elementName isEqualToString:TR]) { // ROW tag
        rowNum++;
    } else if([elementName isEqualToString:TD]) { // COLUMN tag
        colNum++;
    }
}
@end

```

XMLParser.h

```
//  
// XMLParser.h  
// BioLab  
//  
// Created by Nicholas Morin on 3/24/13.  
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.  
//
```

```
#import <Foundation/Foundation.h>  
#import "LabFile.h"  
#import "TitlePage.h"
```

```
#define BIOLAB @"BIOLAB"  
#define LABFILE @"LABFILE"  
#define LAB @"LAB"  
#define TITLE @"title"  
#define ENTRY @"ENTRY"  
#define TEXT @"TEXT"  
#define IMAGE @"IMAGE"  
#define VIDEO @"VIDEO"  
#define INPUT @"INPUTTEXT"  
#define TABLE @"TABLE"  
#define ROW @"ROW"  
#define ROWS @"rows"  
#define COLUMN @"COLUMN"  
#define COLUMNS @"columns"
```

```
@interface XMLParser : NSObject <NSXMLParserDelegate>
```

```
@property (strong, nonatomic) LabFile * labFile; // new lab file being generated while parsing  
@property (strong, nonatomic) Entry * currentEntry; // current entry being created during  
parsing  
@property (strong, nonatomic) Feature * currentFeature; // current feature being created during  
parsing  
@property (strong, nonatomic) NSString * recentTag; // most recent open tag found  
@property (strong, nonatomic) TitlePage * titlePage; // used when creating a the title page when  
parsing labs.xml  
@property (nonatomic) int rowNum, colNum; // current row/column number when creating  
tables
```

```
@end
```

XMLParser.m

```
//  
// XMLParser.m
```

```

// BioLab
//
// Created by Nicholas Morin on 3/24/13.
// Copyright (c) 2013 Worcester Polytechnic Institute. All rights reserved.
//
// Class for parsing both menu and individual lab xml files.

#import "XMLParser.h"

@implementation XMLParser

@synthesize labFile, currentEntry, currentFeature, titlePage, recentTag, rowNum, colNum;

// found open tag (ie <Entry>...)
-(void) parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
attributes:(NSDictionary *)attributeDict {

    recentTag = elementName; // store most recent open tag

    if([elementName isEqualToString:BIOLAB]) { // BIOLAB tag
        // create a new title page
        titlePage = [[TitlePage alloc] init];
    } else if([elementName isEqualToString:LABFILE]) { // LABFILE tag
        // create a new feature for the title page
        currentFeature = [[Feature alloc] init];
        currentFeature.type = [attributeDict objectForKey:TITLE];
    } else if([elementName isEqualToString:LAB]) { // LAB tag
        // create a new lab file
        labFile = [[LabFile alloc] init];
        labFile.title = [attributeDict objectForKey:TITLE];
    } else if([elementName isEqualToString:ENTRY]) { // ENTRY tag
        // create a new entry
        currentEntry = [[Entry alloc] init];
        currentEntry.title = [attributeDict objectForKey:TITLE];
    } else if([elementName isEqualToString:TABLE]) { // TABLE tag
        // create a new feature and set it up for tables
        currentFeature = [[Feature alloc] init];
        currentFeature.type = elementName;
        currentFeature.data = [attributeDict objectForKey:TITLE];
        currentFeature.tableData = [[NSMutableArray alloc] init];
        rowNum = 0;
    } else if([elementName isEqualToString:ROW]) { // ROW tag
        colNum = 0;
        [currentFeature.tableData addObject:[NSMutableArray alloc] init];
    } else if([elementName isEqualToString:COLUMN]) { // COLUMN tag
        // nothing needs to be created for a column tag
    }
}

```

```

    } else {
        // For inside an entry
        // create a new Feature and set the type to the name of this tag
        currentFeature = [[Feature alloc] init];
        currentFeature.type = elementName;
    }
}

-(void) parser:(NSXMLParser *)parser foundCharacters:(NSString *)string {
    if([recentTag isEqualToString:TEXT]) { // TEXT tag
        [currentFeature.data appendString:string];
    } else if ([recentTag isEqualToString:IMAGE]) { // IMAGE tag
        [currentFeature.data appendString:string];
    } else if ([recentTag isEqualToString:VIDEO]) { // VIDEO tag
        [currentFeature.data appendString:string];
    } else if ([recentTag isEqualToString:INPUT]) { // INPUTTEXT tag
        [currentFeature.data appendString:string];
    } else if ([recentTag isEqualToString:LABFILE]) { // LABFILE tag
        [currentFeature.data appendString:string];
    } else if ([recentTag isEqualToString:ROW]) { // ROW tag
        // no data should be inside row tag
    } else if ([recentTag isEqualToString:COLUMN]) { // COLUMN tag
        // will be inside of a table so add a new element to the end of the current row
        if(colNum >= ((NSArray *)[currentFeature.tableData objectAtIndex:rowNum]).count) {
            [[currentFeature.tableData objectAtIndex:rowNum] addObject:string];
        }
        // otherwise append to end of current cell
        else {
            NSString * prevData = [[currentFeature.tableData objectAtIndex:rowNum]
objectAtIndex:colNum];
            NSMutableString * newData = [[NSMutableString alloc] initWithString:prevData];
            [newData appendString:@" "];
            [newData appendString:string];
            [[currentFeature.tableData objectAtIndex:rowNum] replaceObjectAtIndex:colNum
withObject:newData];
        }
    }
}

// found close tag (ie ...</Entry>)
-(void) parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName {

    recentTag = @"";

    if([elementName isEqualToString:BIOLAB]) { // BIOLAB tag
        return; // parsing is done
    }
}

```

```

} else if([elementName isEqualToString:LABFILE]) { // LABFILE tag
    [titlePage.labList addObject:currentFeature];
    currentFeature = nil;
} else if([elementName isEqualToString:LAB]) { // LAB tag
    [labFile addExportTab];
    return;
} else if([elementName isEqualToString:ENTRY]) { // ENTRY tag
    // add the new entry to the labFile
    [labFile.entries addObject:currentEntry];
    currentEntry = nil;
} else if([elementName isEqualToString:TABLE]) { // TABLE tag
    [currentEntry.features addObject:currentFeature];
    currentFeature = nil;
} else if([elementName isEqualToString:ROW]) { // ROW tag
    rowNum++;
} else if([elementName isEqualToString:COLUMN]) { // COLUMN tag
    colNum++;
} else {
    // Inside entry tag
    // add the new feature to the list of features
    [currentEntry.features addObject:currentFeature];
    currentFeature = nil;
}
}
@end

```