

# ***Bookbag: A Web Application for Collaborative Authoring and Distribution of Course Textbooks***

Jake Levin

Partners: Joe Salter, Chris Giglio

Adviser: Robert Fish

January 10, 2017

## **Abstract**

*As it stands, the industry surrounding the creation and distribution of college textbooks is burdensome on both students and professor. For the student, the cost of textbooks is enormous. Additionally most textbooks are rarely used in full and most courses rely on a large amount of external material from the primary course textbook. This material is often times distributed in an organized fashion and is difficult as a student to manage. This problem is in part due to the fact that it is difficult for a professor to find any one single textbook that will fit the course they want to teach in full. As a result if a professor wants a textbook personalized towards their course they are required to write the entire book themselves. If a professor were indeed to do so and then at any point desired to update their course material, they would then also be required to release an entirely new addition of their course textbook. Bookbag aims to solve these issues by providing a platform for professors to collaboratively author version-controlled chapters of information which they can then bundle together into dynamic and personalized courses. These chapters and courses are then distributed to student users. By rethinking course materials as single chapter units of information, Bookbag encourages professors to release information content quickly and with ease. All content on Bookbag will be made available to students for a subscription fee far lower than the average cost of a semesters worth of textbooks, and professors will be compensated relative to the number of students who use the chapter content they publish through our platform. This report will discuss the creation of Bookbag's user interface, focusing in particular on the design and implementation of a split user interface for students and professors to provide all the essential features that make Bookbag unique, and to allow for the best possible user experience for both students and professors.*

## **1. Introduction**

### **1.1. Motivation and Goal**

The motivation for Bookbag initially came from a brainstorming session that took place one evening at the beginning of this semester between myself and my partners, Chris Giglio and Joe Salter. This project was undertaken as a part of the Independent Work Seminar 08, and therefore our first task given to us by Professor Fish was to identify an existing problem for which we might be able to

create a piece of software to help solve. During that brainstorming session we decided that the problem we wanted to try and tackle is one particular relevant to our lives as students, and that is the exorbitant sum we are required to spend every semester on a large number of textbooks that we often find we hardly use. We also realized that to best solve this problem, not only would we need to address the issue of expensive textbooks, but also the issue of professors using textbooks for their courses that were often times not dynamic nor personalized enough to encompass the entirety of the course they desired to teach, and thus from which only certain sections and particular chapters were assigned. From a professor's perspective there is no simple solution to this problem, as the work and time required to write and publish a textbook is very high, and so most professors do not write a specific textbook for every course that they teach. Additionally, updating a textbook to reflect any changes a professor might want to make in the content covered in their course or any recent changes in the field would require a new edition of the entire textbook be published.

Thus, we have spent this past semester building a platform we call Bookbag, which we think can help solve both facets of the problem identified above. The goal of Bookbag is to: 1. reduce the burdensome cost of textbooks for students worldwide through a subscription based service that provides chapters of information organized by professors into courses that students can then subscribe to and use for their coursework, and in doing so, 2. provide a simple and elegant platform for professors to create dynamic and personalized courses without relying on any one single textbook, while 3. using this break from the classic textbook model to allow professors to write their work in smaller chapter-based units that can be written in a collaborative and version controlled environment, so as to also ease the burden on professors for the creation and distribution of their work.

## **1.2. Problem Background**

## **2. Related Work**

## **3. Approach**

What makes Bookbag unique and what will allow it to reshape the current textbook industry is the way in which it condenses the creation and distribution of course materials into one platform. Bookbag effectively cuts out the publisher as a middleman and delivers content straight from the professor to the student. Bookbag also aims to ease the process of content creation for professors by facilitating collaborative authorship, and by lowering the threshold for published content from a full textbook or lengthy academic paper, to a single chapter of information. In order to design and implement a platform that provides for all of these essential features, Bookbag's user interface has been split into two separate interfaces for the consumers of content, *Students*, and the producers of content, *Professors*.

### **3.1. Student User Interface**

Student users are provided with an interface designed for content consumption and personalized content management. The role of a student is to curate a personalized library of course materials, which they do primarily by "subscribing" to courses which have been assembled and made publicly available by professors. Students have been given unlimited access to all content published by professors on Bookbag, and have been given an interface to search through these chapters and courses. Additionally, students have been given the ability to create personalized "Folders" of chapters. This is so that students are not locked into subscribing solely to courses as they have been assembled by professors. As students, we find that often times some chapters resonate more strongly than others on any given topic, and we wanted to make sure that students would be able to personalize the way in which their content libraries are structured within Bookbag. We also hope that this will encourage students to use Bookbag for research and other academic interests outside of their courses.

The student users core functionality is much simpler than the professors and centered predominantly around their personal library, and therefore the approach used in designing the student interface itself reflected this simplicity. The student never navigates away from their main library page, except when searching for new content to subscribe to. All of a student's course subscriptions and folders are dynamically displayed and listed for them in a sidebar. When a student selects one of these courses or folders from their library sidebar menu, its chapter contents are displayed for them in the central view component of the library page. Any chapter that a student encounters, either while searching or browsing through the courses they have subscribed to, can be added to any of their folders. Students have unlimited access to all of the content in their library, and can download any of their chapters at any time as a PDF in browser.

### **3.2. Professor User Interface**

Professor users are provided with an interface designed for content creation and distribution. The primary role of a professor is to create "Chapters" of information which they can then bundle together into courses. The key idea here is that Bookbag has professors focus on producing chapters of information, rather than entire books. Using chapters as the essential unit of information has a number of advantages over the classic text-*book* model:

1. Professors can publish content far more easily and at a much faster rate.
2. Professors can collaborate on isolated units of information when desired and revenue can be easily split accordingly.
3. Necessary revisions to content can be easily isolated and updated.
4. Courses can be constructed dynamically and personalized exactly to fit a courses needs.
5. Professors can be given access to all other public chapters of information to use in their research and in constructing courses.

When a professor creates a chapter it is initialized as private and without any contributors. A professor can then add any other professor users on Bookbag as contributors, and when they are ready, publish the working version of the chapter so that it can be accessed by student users and

they can begin to receive revenue for their work based on the number of subscribers the chapter receives. Until it has been made public, only the owner and contributors of a chapter can view its contents. Additionally, whenever a professor updates the working chapter files a new "version" of the chapter is created. This way a professor has access to every version of the chapter, and can revert and update the chapter at any point in its revision tree.

The other primary function of professors is to assemble collections of chapters into "Courses". Professors can create courses from their personal chapters, as well as any other publicly available chapter in Bookbag's library. This is another key component of Bookbag as it allows professors to construct their courses from a diverse collection of content, and eliminates the often times unorganized distribution of content to students from external sources. Similar to the chapter functionality, when a professor is ready they can publish their course so that it is made publicly available to students to subscribe to. The course creation interface was designed so that professors could assemble courses with an easy search and click style, similar to how a user can assemble playlists in iTunes or Spotify.

### **3.3. Abstracting Over Git For Collaboration and Version Control**

The majority of the chapter authorship functionality described above is accomplished by using an abstraction over Git created by Joe Salter. While Git is an incredibly powerful tool, it is not necessarily the most intuitive software to use without a computer science background. Therefore, the professors user interface was designed specifically so that no knowledge of Git or version-control is necessary for a professor to use Bookbag's work environment. All versions of a chapter are kept in a list for the professor, and when a professor selects a chapter version all of that versions information is dynamically displayed. Similar to a library book, when a professor wants to upload new files for a chapter they must first "Check Out" the chapter. When a chapter is checked out all other contributors can still view and download the chapter contents, but only the professor who has checked out the chapter can make edits and upload new files to the chapter. In this manner we have ensured that at no point will any professors have to deal with the possibility of a complicated merge

conflict. The maximum duration of a chapter's checkout period is set by the owner of the chapter, and once it has expired the chapter becomes available once again to all other contributors to check out and upload their edits.

## **4. Implementation**

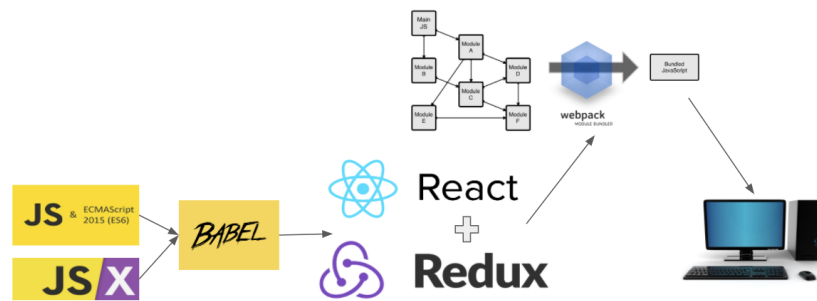
Creating Bookbag's user interface required a great deal of technological and design related decisions. As the application is currently deployed, the user interface consists of thousands of lines of code in over 50 different files. Naturally, as this project grew and took form the technologies and design used constantly evolved. The following section discusses in detail the process involved in designing and implementing Bookbag's user interface.

### **4.1. Development Stack**

Bookbag's user interface was developed using React and Redux JavaScript libraries. The majority of it was written in ES2015 and JSX, with Babel being used to translate and compile these higher level JavaScript languages into simple JavaScript to make sure everything runs smoothly in browser. Webpack was used to bundle all of the additional node and react modules used and their dependencies so they could be displayed in browser. CSS was used for styling, and a number of React-Bootstrap components were used for things such as lists, forms, modals, and popovers. A full list of the modules used in developing Bookbag's user interface can be found in the Appendix of this report. A diagram of this development stack can be seen in Figure 1. This development stack was inspired by Cory House's react-slingshot boilerplate which is available for public use on GitHub.[?] In particular this boilerplate was used for the initial Webpack and Babel configurations.

### **4.2. An Introduction to React and Redux, Stateful Component-Based Design**

React is a component-based framework for building user interfaces. Practically speaking this means that the user interface is broken up into a number of JavaScript files that each contain a piece of the pages being rendered within the browser. There is a great deal of configuration and setup involved in developing within React that I will not describe in any detail, but what is important to know



**Figure 1: Diagram of user interface development stack.**

is that when a user visits one of the pages within Bookbag they are interacting with a series of components that each contain the data relevant to the specific function of that component. For example, a button is a component that has all of the information and data needed for it to perform whatever function that button is supposed to trigger on click. These components are broken up into "smart" components called containers and "dumb" components. The "smart" containers are the containers for a given page that a user interacts with. These containers contain the state of all the data being used for a given page, and disperse this data to all of the dumb components within the page as needed. This data can be data coming from the server, or from the user as they interact with the user interface. This is where Redux comes in. Redux helps store and manage this state so that it can be passed within and between components on a page. As the state of the data within a page changes, which might happen for example when a student user subscribes to a course and it needs to now be displayed within the users course library, the state of the container page is updated and React automatically re renders any of the components within the container that receive this data. Redux also allows the dispatching of "actions", which allow the user interface to take the data stored in the state of the page and use it when communicating with the server. While this paper will not go into any more detail on React, Redux <sup>1</sup>, or what is happening on the back end of Bookbag, <sup>2</sup> a low-level understanding of this hierarchical container-component structure and the notion of

<sup>1</sup>More information on React and Redux can be found at <https://facebook.github.io/react/> and <http://redux.js.org/> respectively

<sup>2</sup>See Joe Salter's and Chris Giglio's report for more information on the server, database, and Git abstraction of Bookbag.

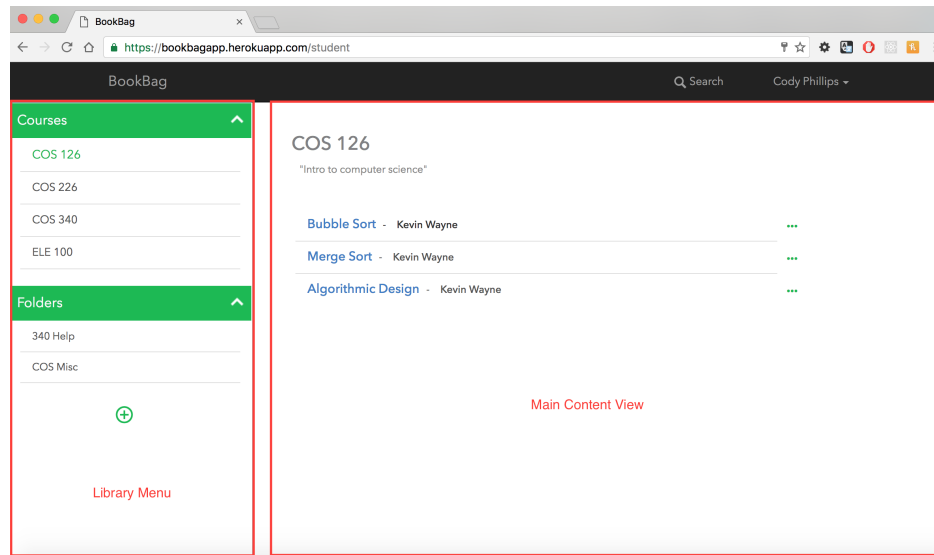
handling data as a state is important for the remainder of this section on implementation. For an amazing video-course covering the basics of developing using React and Redux, check out Cory House's "Building Applications with React and Redux in ES6", available on Pluralsight. [?]

### **4.3. Designing and Implementing the Student User Interface**

When designing the student user interface, the primary goal was to design an interface that was clean, simple, and totally intuitive. Ultimately the student user interface is just a digital library of written content with some additional personalized management features and search functionality. For inspiration I looked at the user interfaces for a number of digital library based applications, such as Evernote, Spotify, and iTunes. While very different in stylistic approaches, all of these interfaces displayed libraries centered around organized lists of content that when selected are displayed in a central view. For Bookbag's purposes, all of the content student will be interacting with are chapters that can be organized into either courses built by professors or folders built by the students themselves. Thus, it seemed most intuitive to provide the student with a library menu broken into two lists of selectable content, one for courses and one for folders. Alongside this content would be the main view display, and whenever a course or folder is selected its chapter contents would be displayed in the main view. Because the entirety of a student's actions within Bookbag revolve around managing their content library, the student interface consists of one singular central page. This page is broken up into the library menu and the main content view as can be seen in Figure 2. Note here that stylistically I opted for a lighter color scheme, as I found that it provided for a cleaner and more appealing aesthetic when dealing with text-based digital content.

**4.3.1. The Student Library Menu** The most important decision regarding designing the student library menu was whether to keep it displayed at all times. Ultimately, since student users are likely to frequently switch between courses and folders as they access the content within them, the decision was made to keep the library menu displayed at all times. If at any point the student wants to switch between which course or folder has its chapter contents displayed, they simply click on the name of the desired course or folder in their library menu. These menus are populated by requesting



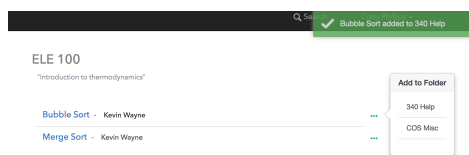


**Figure 2: Bookbag’s student interface as divided into the library menu and main content display.**

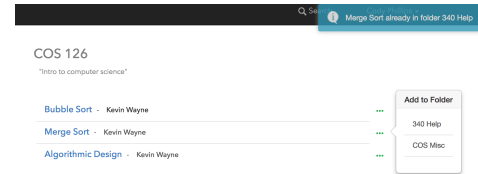
a student’s courses and folders from the server, and dynamically displaying this data through a series of list item components. The menus themselves are collapsible thanks to a collapsible react component available for public use on GitHub. [?] Note also that within the folder menu there is a plus button for the student to add a new folder. When the student clicks on this button a text field form pops up for the student to input the new folder’s name. Once submitted this form data is sent to the server, a new empty folder is created for the student user, and the state of the student’s folders data is dynamically reloaded so as to display the new folder. When a student clicks on one of the course or folder names within the library menu, an event is triggered that changes the state of which course or folder content has been selected to be displayed. Within the library menu the current course or folder being displayed is signified by its name being highlighted in green.

**4.3.2. The Student Library Course and Folder Contents View** The main view component of the student user interface is the selected course/folder content display. This component receives the state of which course or folder is selected to be displayed and all of its content. It displays the name of the selected course/folder, the description of the course if selected, and all of its chapters as a series of list item components. When a student clicks on the chapter an event is triggered that opens a new tab in the users browser with the corresponding chapter PDF. Additionally the student can click on the name of the chapter to download the PDF. Alongside each chapter list item component

is a button that triggers a pop-up on click prompting the user to add this chapter to one of their folders if they so desire. This pop-up contains a dynamically displayed list of all of the users folders, which it receives from the main page container's folders data state. When the student clicks on one of the listed folders an action is dispatched that tells the server to add the selected chapter to the selected folder. If the server responds successfully the folder data state for the user is reloaded, and the chapter will now be displayed within the folder contents when selected. The user also receives a confirmation (or chapter already contained in folder) message as shown in Figure 3 and Figure 4. These confirmation messages have been generated using a module named toastr, publicly available on GitHub. [?] Note also in Figure 3 and Figure 4 the pop-up prompt for adding a chapter to a folder.



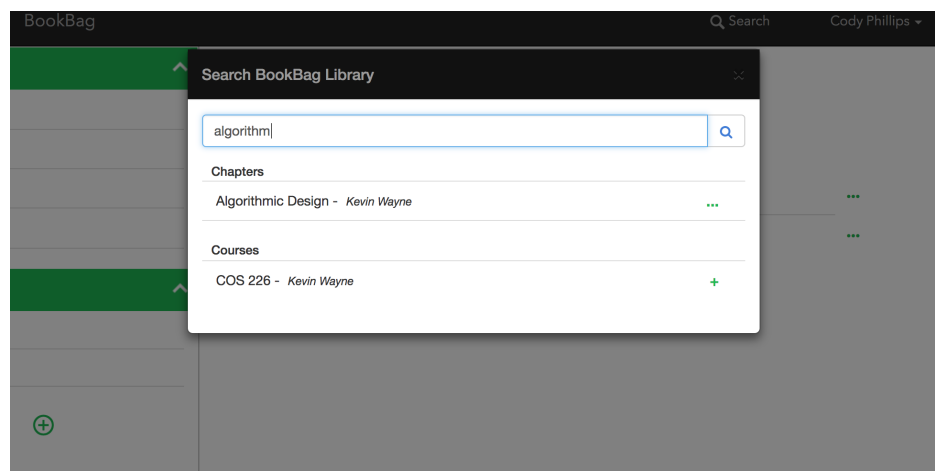
**Figure 3: Successful add chapter to folder confirmation message.**



**Figure 4: Chapter already contained in folder alert message.**

**4.3.3. Searching for Content** The final primary functionality provided for students is the ability to search through Bookbag's content and subscribe to courses, as well as add the chapters they encounter in their search to their folders. Searching for content occurs entirely within a react-bootstrap modal component overlay that is triggered by clicking on "Search" in the interface header menu. The search feature was designed to be contained within a modal so as to significantly separate the search contents display from the main library contents display, while at the same time keeping the student's actions centralized around the same main library page. When a student enters a search query into the search form and submits it, the query is dispatched and sent to the server, and the server responds with all the matching results based on names, descriptions, and keywords. These results are then stored within a search results state which is dynamically displayed to the user. These results are displayed as two lists, one for the courses found that match the query, and one for the

chapters found as seen in Figure 5. When a student clicks on the green plus button next to a course an action is dispatched that sends a call to the server to subscribe the student to the selected course, and the student's course data state is reloaded so as to dynamically display this new course in their library menu. When a student clicks on the button next to a chapter, a pop-up component is used to allow the student to add the selected chapter to one of their folders. Note here the beauty of React; the pop-up component for adding a chapter to a folder can be reused across a number of different event triggers. All that changes between these use cases is the state of the data being accessed to reference the selected chapter and folder.



**Figure 5: Search modal and its results.**

#### **4.4. Designing and Implementing the Professor User Interface**

While the student user interface was rather simple in its design and implementation, Bookbag offers far more complicated features, and therefore a far more intricate user interface split up between a few different pages, for professors to author and distribute course and chapter content. When a professor logs in they are routed initially to their "Workbench". From the workbench a professor can see the courses and chapters they are currently working on and have already published. When one of these courses or chapters is clicked on the professor is routed to the selected course or chapter page. The course page provides an interface for the professor to add/remove chapters in a course, update the settings for a course, and if the course is still in progress to publish the course so that it will become publicly available to student users. The chapter page is a bit more intricate as it

provides an interface for the Git abstraction we developed. This includes features for a professor to view all versions of a chapter, download the content files of the selected version of the chapter, checkout/upload files to the chapter if it has not already been checked out, as well as add/remove contributors and publish the chapter if the user is the owner of the chapter. Further details regarding the design and implementation of the professor user interface are discussed in the following sections of this report.

**4.4.1. Professor Workbench** The workbench acts as a professors home page. From here a professor can access all of the courses and chapters they have created and contribute to. In designing this workbench the primary challenge was how best to split up a professors courses and chapters, and how to differentiate between content that is in progress and content that has been published. Ultimately I decided to keep courses and chapters separated through a react-bootstrap tab component that switches between courses and chapters on click. Depending on which tab has been selected, the main content view displays either the professors courses or chapters, and these are then separated into two lists for those that are in progress and those that are published. This layout can be seen in Figure 6. When a professor clicks on one of their chapters or courses they are routed to the corresponding course or chapter page. Additionally a professor can create a new chapter or course by clicking on the green plus sign and selecting either "Create New Chapter" or "Create New Course".

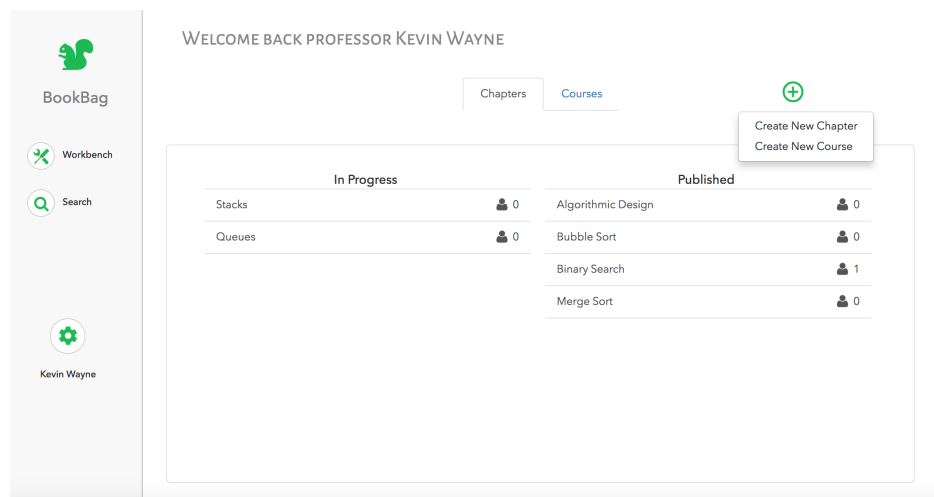
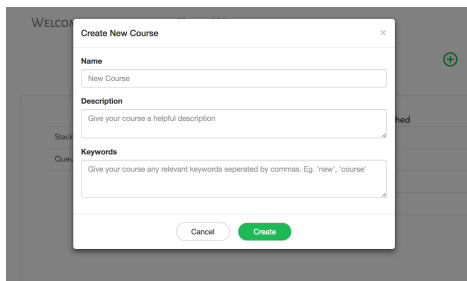
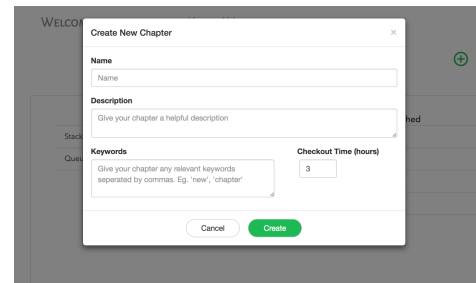


Figure 6: Professor workbench page.

**4.4.2. Course and Chapter Creation** When a professor requests to create a new course or chapter, a react-bootstrap modal drops down containing a form prompting the professor to fill out the desired course/chapter fields. For a course these fields are: name, description, and keywords. For a chapter the fields are very similar, but the professor is also asked to set a checkout duration time. This specifies how long a contributor can keep a chapter checked out as they upload new chapter files. Once submitted this form data is sent to the server, and if successful, the state of the professor's course/chapter data is reloaded so as to dynamically display the new course/chapter. The course and chapter creation forms can be seen in Figure 7 and Figure 8.

A screenshot of a web application showing a modal titled "Create New Course". The modal has a close button (X) in the top right corner. It contains three text input fields: "Name" with the placeholder text "New Course", "Description" with the placeholder text "Give your course a helpful description", and "Keywords" with the placeholder text "Give your course any relevant keywords separated by commas. Eg. 'new', 'course'". At the bottom of the modal are two buttons: "Cancel" and "Create".

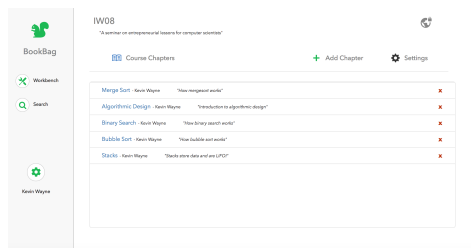
**Figure 7: Create new course form.**

A screenshot of a web application showing a modal titled "Create New Chapter". The modal has a close button (X) in the top right corner. It contains three text input fields: "Name", "Description" with the placeholder text "Give your chapter a helpful description", and "Keywords" with the placeholder text "Give your chapter any relevant keywords separated by commas. Eg. 'new', 'chapter'". To the right of the "Keywords" field is a "Checkout Time (hours)" field with a numeric input box containing the value "3". At the bottom of the modal are two buttons: "Cancel" and "Create".

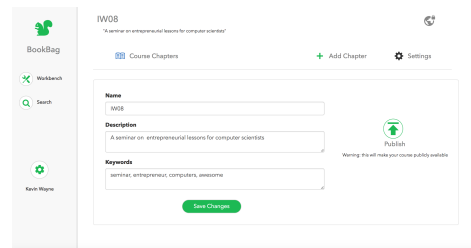
**Figure 8: Create new chapter form.**

**4.4.3. Course Page** When a professor clicks on a course from their workbench they are routed to the the course page which immediately loads all of the selected course's data and stores it in the page state. It does so by making a request to the server and passing it the selected course ID, which the page receives from a URL parameter which is set by the workbench page when the course is initially selected. The course page was designed around two key features, that of adding/removing chapters to a course, and that of updating the courses so as to change its name, description, keywords, and privacy settings as desired. The bulk of the page is centered around a main content display, which depending on the current state of the page, displays either the current course chapters or a form to edit the current course settings as seen in Figures 9 and 10. The state that determines which of these is displayed is set by default to the course chapter display, but this can be toggled by the professor by clicking on either the "Course Chapters" or "Settings" buttons located at the top of the page. Within the course chapters display the professor can see all chapters which are currently a part of the course, and can delete them as desired by clicking on the red "x" button to the right of

the chapter name. Additionally, clicking on the chapter name will open the chapter PDF in a new tab. The course settings display contains a form for updating all of the course settings, as well as button for making the course public.

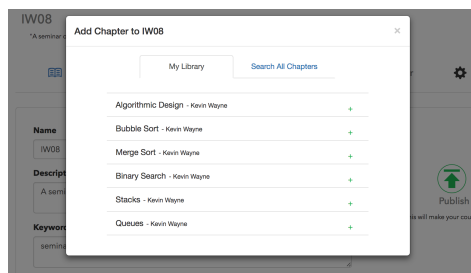


**Figure 9: Course chapter display.**

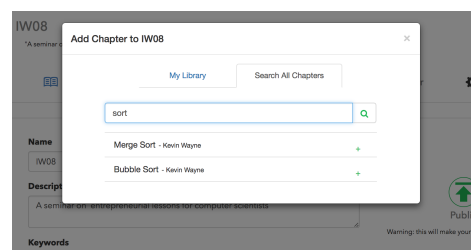


**Figure 10: Course settings display.**

A central "Add Chapter" button is also located at the top of the page. Clicking on this button triggers a react-bootstrap modal dropdown which displays either a list of all the chapters that a professor owns and contributes to, or a search form similar to that of the search modal in the student interface as seen in Figures 11 and 12. Which of these methods is displayed is determined by toggling between two tabs located at the top of the add chapter modal.



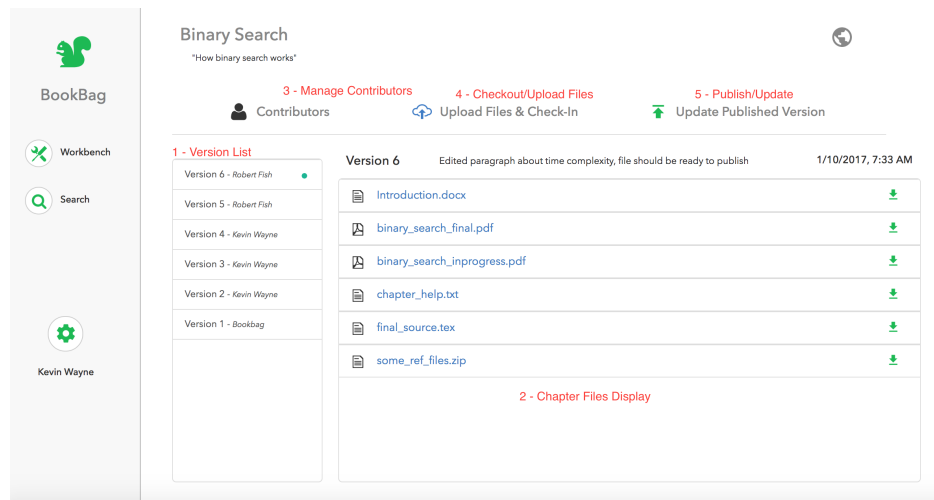
**Figure 11: Add chapter by library.**



**Figure 12: Add chapter by search.**

**4.4.4. Chapter Page** When a professor clicks on a chapter from their workbench they are routed to the the chapter page which, just like the course page, immediately loads all of the necessary chapter data and saves it in the page state. The chapter page was by far the most difficult design and implementation challenge I faced while building Bookbag's user interface. Ultimately, the goal of the chapter page was to provide a complete user interface for our abstraction over Git without requiring any user knowledge regarding collaborative, version-controlled work environments. This required handling an immense amount of data, dispatching a large number of actions to load and

store this data on the state, and linking together a bunch of intricately related components. Half this report could be spent discussing the implementation of the chapter page alone, however we will attempt to touch only briefly on each of the key components in this page. See Figure 13 for a visual reference of each component being discussed.



**Figure 13: Chapter page, an interface over Git abstraction.**

**1. Version List:** The version list is located in the far left side of the main chapter page display, and lists all of the versions that have been uploaded for the selected chapter and the contributor that was responsible for uploading that version. Clicking on one of these versions will update the state that stores all of the data related to the current version files to be displayed in the central chapter files display view. The currently selected version that is being displayed is indicated by a green dot next to the version name. By giving professors single-click access to all of the past versions of a chapter, we aim to make it easy to keep track of all contributor updates for a given chapter and to revert files if necessary. All of this functionality calls on the API we wrote for abstracting Git.

**2. Chapter Files Display:** This is the main view display that lists all of the chapter files for the selected chapter version. This component was modeled off of GitHub's interface for repository files display. When a professor clicks on one of the listed file names or the download icon to the right of one of the files, a copy of the file is downloaded and displayed in a new browser tab. This functionality is made available to all contributors, for all versions of a chapter, at any point in time.

In this manner we hope to make it easy for professors to collaboratively author chapters, as this gives them total transparency regarding all versions of a chapter, and eliminates the possibility of ever having to worry about working off of outdated files. Additionally, whenever a contributor uploads a new version of a chapter they are required to type in a short descriptive message of the update made, similar to that of a GitHub's commit message. This feature was one that I have found very useful in the past when working with Github, and so I wanted to make sure to implement it within Bookbag as well. This message can be seen displayed above the chapter files list, alongside the date and time of the upload of the chapter version being displayed.

**3. Manage Contributors:** When clicked, this icon button displays a list of all of the contributors currently working on the chapter. If the current professor is also the owner of the chapter (i.e. they were the one to initially create it), they also have the ability to add and remove contributors from this list. To add a contributor, the owner of the chapter simply searches for other professors on Bookbag by name. This search functions through a simple search form and display, similar to the search form used to add a published chapter to a course. To remove a contributor, the chapter owner simply clicks on a red x located next to one of the listed contributor's name.

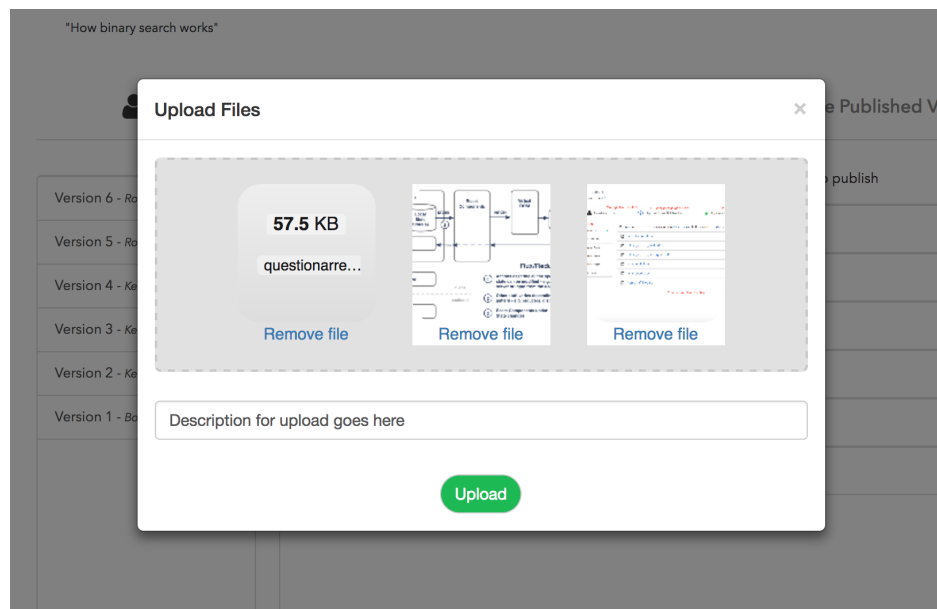
**4. Checkout Chapter Upload Files:** From a technical standpoint, checking out a chapter and uploading files was one of the most difficult features of Bookbag's user interface to implement. To do so requires keeping track of the chapter owner, if the chapter is checked out and if so to whom the chapter is checked out too, which version of the chapter is currently being accessed, and if this chapter version being displayed is the most recent version. This not only required a great deal of state handling, but a lot of constant communication with the server. However, this checkout style design was essential in abstracting over Git in a manner that ensured that no merge conflicts could ever take place. If no two contributors can ever checkout the chapter at any one point in time, then no two professors can commit any changes to a given chapter repository at the same time. Thus, we can safely provide collaborative, version-controlled authorship to professors without them ever needing to know a thing about the underlying technologies.

Additionally, being able to upload files required an entire user interface in and of itself. Ultimately I



used a react-dropzone-component module which is publicly available on GitHub [], to help create an all-in-one drag and drop and click to upload file component. I then needed to create a state to reference and keep track of the files as they were being added and removed to the upload queue of the dropzone-component, as well as the desired commit message for the file uploads. This information can then all be sent to the server when the professor triggers the final upload submission. The upload files interface can be seen in Figure 14

**5. Publish/Update Chapter:** Lastly, an interface was required for professors to publish their chapters and to update the public version of already published chapters. Ultimately, while we want professors to be able to upload as many files of whatever file type they want while authoring their chapters, the final "chapter" that is published needs to be a single PDF file so that it will be easily accessible by students. Therefore the Publish/Update Chapter button was designed so that when clicked, it would display a list of all of the PDF files within the chapter. Whichever one of these files is selected is set as the PDF file associated with the published chapter, and it is this file that is then downloaded by student subscribers when they access the chapter within the student interface.



**Figure 14: The upload files interface.**

## 5. Business Model

## 6. Evaluation

It can be difficult to evaluate the user interface of a web application such as Bookbag, as much of the user interface functionality relies on the server and database of the application. To help mitigate this problem, throughout the development of Bookbag's user interface I made sure to initially implement each component by using "dummy data" - data that was hard coded in and not reliant upon the server. As a result, I am confident that assuming the server and database behave as expected, Bookbag's user interface effectively provides all of the features discussed in Section 4 of this report. Therefore, we can now partially evaluate Bookbag's user interface on whether or not it could be considered a successful minimum viable product. Additionally, I gave Bookbag to a number of my peers along with a user experience questionnaire in order to receive some quantifiable evaluative data.

### 6.1. A Successful Minimum Viable Product

As it stands, is Bookbag a successful minimum viable product? The short answer is yes, given that some additional security features were implemented.<sup>3</sup> When we started building Bookbag, our goal was to create an application that gave professors the ability to collaboratively author chapters in a version-controlled environment, that they could then bundle into courses and distribute to student users. A semester later we have produced a product that provides all of the essential features necessary for this desired functionality. Admittedly, we have yet to implement any sort of payment mechanism and so professors have no way of receiving compensation for their work, however for our initial release we would want to beta-test our product for free anyways before attempting to monetize it. Thus, I think it is quite reasonable to say that we can evaluate Bookbag as having (just about) successfully reached the point of MVP.

---

<sup>3</sup>The reality today is that many web applications are initially released with horrendous security flaws far more egregious than those of Bookbag. That being said, on principle we would like to refrain from releasing a product with known security flaws to the public.

## 6.2. User Feedback

In order to receive some more quantifiable data to evaluate Bookbag's user interface, I reached out to a number of my peers and asked them to use Bookbag for 15 minutes and then fill out a short user experience questionnaire. This questionnaire can be found in the Appendix at ????.

## 7. Conclusion

Currently, we have a fully functioning web application deployed and running on Heroku.<sup>4</sup> Student and professor users can sign up for Bookbag and use all of the features discussed in Section 4. While we have successfully accomplished the key goals we set when we initially began to work on Bookbag, there are of course some limitations and a great deal of work to be done in the future.

### 7.1. Limitations

At the end of the day, in its current state Bookbag is truly a bare-bones minimum viable product. Even within the features we have successfully implemented, there are a number of limitations. From the student perspective, a student user can currently only subscribe to courses or add a chapter to one of their folders. There is no option to simply subscribe to a chapter. Additionally, folders only take names and no other fields (such as a description). Because the student library was the first component of the user interface to be implemented, it also references some of its course/chapter/folder data by name rather than ID which is poor coding practice as it can lead to corner case collisions (this is because unique IDs were not initially implemented in the database architecture). There have also been some styling issues noticed in the student interface at higher resolutions than the laptop from which I have been developing on which, when isolated, would require some simple CSS adjustments. From a usability point of view there is also the giant limitation of not having a large database of content for students to subscribe to. As we have not yet given Bookbag to professors to start uploading content, there is very little that a student can practically do at this point.

---

<sup>4</sup>Bookbag can currently be accessed at <https://bookbagapp.herokuapp.com/>

On the professor side, functionality wise the biggest limitation is that I did not have the time to include in the professor interface all of the features that students have for curating personal libraries. This was intended to be implemented in addition to the content creation workbench to help incentivize professors to use Bookbag, and also to help facilitate any research professors may need to do in order to produce quality chapter content. The other big limitation is how to deal with content copyright infringement and illegal distribution. As it is currently implemented, students can download the PDF files of any published chapters without any sort of restrictions on access and redistribution. While this may seem like a critical limitation, in truth this issue is actually rather easy to fix with a bit of funding. INSERT INFO ON PDF COPYRIGHT COMPANY HERE.

As far as more general limitations go, we currently have not implemented a reliable method to verify whether or not a professor who signs up for Bookbag is actually a professor. This causes a bit of a security issue as we have no way of ensuring what sort of content is now published.<sup>5</sup> Lastly, we have few security features in place to deal with various attacks such as SQL injection and cross site scripting.

## **7.2. Future Work**

Possibly the most exciting part of this project is the incredible potential for future work on Bookbag. There are so many features that, given time, would transform Bookbag from its current proof-of-concept MVP state, into a marketable and most likely profitable product. Once a solid database of content has been published on Bookbag, the first feature that needs to be implemented is an intelligent Browse feature for students with a recommendation system based on their current course and chapter subscriptions. This, in parallel with a fine-tuned search function, would make the students ability to access content far more powerful. Additionally, we would like to implement an idea for the student interface called "Course Notes". Course notes would allow students to upload their own PDF/text files and then pin these to courses alongside chapters, effectively simulating a students ability to take notes within a copy of their textbook. We would also like to implement

---

<sup>5</sup>One would hope that in theory a legitimate professor would not publish any sort of malicious data files as they would risk their academic career in doing so

a notifications system so students would be alerted when a published change has been made to a course or chapter they are subscribed to. This notifications system could also be used to allow professors to directly assign courses or chapters to their students. This would then open up the option for a whole "class" interface, which would consist of a professor(s) being given a direct channel of communication to students. At some point we would also need to create an account settings interface, as well as updated user profile information. As mentioned in limitations, professors would also ideally be given all of the features described for students as well, and certain security features such as professor verification would need to be added. While we are very proud of what we have accomplished with Bookbag this semester, we are even more excited by the prospect of what is yet to come!

Rate the following on a scale of 1 – 5 (5 is best, 1 is worst):

1. Interface was intuitive and required minimal explanation
2. Once figured out, interface was easy to use
3. All key features for creating chapters courses, as well as browsing and subscribing to content were present and obvious
4. Interface layout was clean and uncluttered
5. Interface had no apparent bugs
6. Interface was quick and responsive
7. Interface was aesthetically appealing

General questions:

8. Would I use Bookbag for my courses?
  - 8.1. If so, how much would I be willing to spend on Bookbag a semester?
11. Was the interface at any point a deterrent to my use of Bookbag?
12. Are there any features not present I would like to see in Bookbag?
13. Was anything confusing and not obvious about how to operate?

Any suggestions?

If you are using L<sup>A</sup>T<sub>E</sub>X [1] to typeset your paper, then we strongly suggest that you start from the template available at <http://iw.cs.princeton.edu> – this document was prepared with that template.

### 7.3. Citations and Footnotes

There are various reasons to cite prior work and include it as references in your bibliography. For example, If you are improving upon prior work, you should include a full citation for the work in the bibliography [2, 3]. You can also cite information that is used as background or explanation[4]. In addition to citing scholarly papers or books, you can also create bibtex entries for webpages or other sources. Many online databases allow you to download a premade bibtex entry for each paper you access. You can simply copy-paste these into your references.bib file.

Sometimes you want to footnote something, such as a web site.<sup>6</sup> Note that the footnote number comes after the punctuation.

Figure ?? and Figure ??. (Note that the the “F” in Figure is capitalized.

## References

- [1] L. Lamport, *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System*, 2nd ed. Reading, Massachusetts: Addison-Wesley, 1994.
- [2] F. Lastname1 and F. Lastname2, “A very nice paper to cite,” in *International Symposium on Computer Architecture*, 2000.
- [3] F. Lastname1 and F. Lastname2, “Another very nice paper to cite,” in *International Symposium on Computer Architecture*, 2001.
- [4] B. Salzberg and T. Murphy, “Latex: When Word fails you,” in *Proceedings of the 33rd Annual ACM SIGUCCS Conference on User Services*, ser. SIGUCCS ’05. New York, NY, USA: ACM, 2005, pp. 241–243.

---

<sup>6</sup><http://www.cs.princeton.edu>