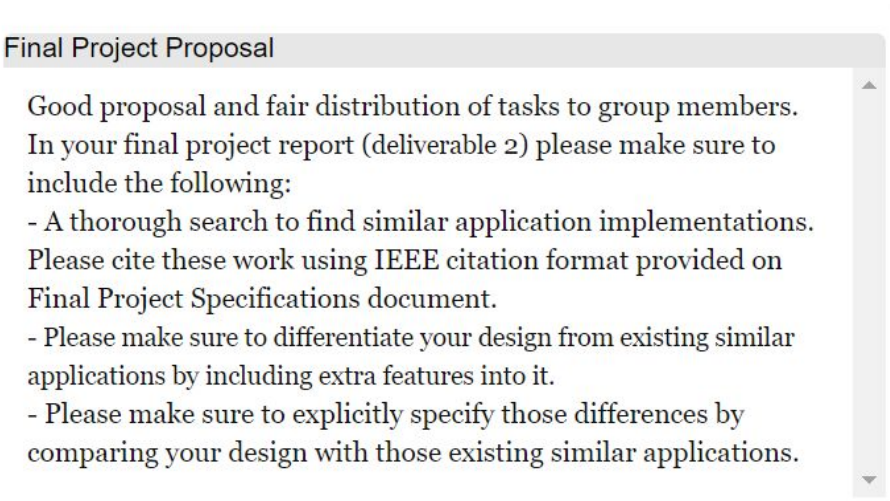
CS3354 Software Engineering
Final Project Deliverable 1

Bookkeeper
E-Reader Application

Group 9 Members:
Luca Dejesu, Miranda Flemming, Shreya Gazawada, Andy Shao, Vaibhav Sharma, Danh Vo, Jonathan Wachholz

1. [5 POINTS] Please attach here the Final Project draft description (that contains the instructor feedback). It is ok to include a picture of the original document. Address the feedback provided for your proposal by listing what you did / plan to do to comply with those proposed changes and or requests for additions to your project.

Feedback:



To address the feedback we received, we did an investigation into other currently available e-reader applications. We looked at applications such as Calibre, which is popular on desktop and laptop systems, iBooks, which is popular on iOS, as well as some Android e-reader applications such as Moon+ Reader. Since bookshelf apps are well-trodden territory in terms of software development, we thought that it would be important to include a standout feature. [1]

In particular, thanks to a growing focus on building hardware to support AI on mobile devices, such as the AI chip in the latest iphones, we felt that including a feature that benefits from natural language processing would be performant and a welcome addition for users who would rather listen to a book while occupying themselves with another activity requiring their attention. Our app will have the ability to essentially turn any text book into an audiobook by using NLP to transcribe text to audio. NLP has come a long way since the days of robotic voices, and we feel that the technology has matured enough by now to be welcomed by users who may only have a text version of a book, but want to listen to it instead of reading. [2] As stated previously, the main use case for this feature would be to provide an option that would allow users to listen to a book that they do not have an audiobook file for.

As we searched for similar applications, we found that most of them, although featureful in almost every other aspect, did not have support for text-to-audio transcription. As we searched for other implementations, as well as other information in the process of completing this deliverable, we made certain to create IEEE citations for all of our sources, in addressing the other major point of feedback we received. As asked, we will include all of these citations in our second deliverable, and we have integrated the audiobook feature into our current design. We have currently settled on a solid core set of base functionality for our application after going through the requirements gathering process, have evenly divided the tasks for the second deliverable, and have decided on using the MVC design pattern with an Agile approach. Thanks to our component-based architecture and Agile design process, we are confident that if between now and the second deliverable, new requirements emerge, that we will be able to include them in our app while incurring a minimal cost.

2. [10 POINTS] Setting up a Github repository. Please use your utdallas email accounts only for each group member.
1.1. Each team member should create a GitHub account if you don't already have one.
1.2. Create a GitHub repository named 3354-teamName. (whatever your team name will be).
1.3. Add all team members, and the TA as collaborators. Here is the TA info: TA GitHub id: bmaweu TA email: barbara.maweu@utdallas.edu
1.4. Make the first commit to the repository (i.e., a README file with [team name] as its content).
1.5. Make another commit including a pdf/txt/doc file named "project_scope". If you choose a predefined topic (one of the 4 topics described in the "Project Topic Ideas" section of this document), the contents of the file should be identical to the corresponding project in this section. If you choose other topics, the contents should follow a similar structure.
1.6. Keep all your project related files in your repository as we will check them. Include the URL of your team project repository into your project deliverable 1 report.

Important Note:
• Tasks 1.3 - 1.5 should be performed by different team members. We will check the commit history for these activities.
• Do not include credentials (e.g., UTD ID) in the repository.
• Only commits performed before the deadline will be considered. Do not forget to push your changes after you have done the work!

Link to GitHub Repo: https://github.com/danhvo11012/3354-Group9

3. [5 POINTS] Delegation of tasks: Who is doing what. If no contribution, please specify as it will help us grade each group member fairly.

1. Addressing Feedback (Vaibhav)
2. Software Process ModelSetting Up GitHub Repository (Danh, Shreya, Miranda)
    a. adding all collaborators (Danh)
    b. creating README.md (Shreya)
    c. creating a project scope file (Miranda)
3. Delegation of Tasks (Everyone)
4. Software Process model (Luca)
5. Software functional requirements (Vaibhav, Miranda)
6. A Use Case Diagram. (Jonathan)
7. Sequence Diagrams
    a. Search books by text query (Miranda)
    b. View Book (Shreya)
    c. Managing Books
        i. Load books from system into database (Luca)
        ii. Delete books (Luca)
    d. Manage Categories
        i. add (Shreya)
        ii. delete (Shreya)
8. Class Diagram. (Danh, Andy)
9. Choosing and Applying one appropriate Architectural Design for our project. (Vaibhav, Shreya, Andy)

4. [5 POINTS] Which software process model is employed in the project and why. (Ch 2)

Software Process Model: Agile Development

Why an Agile Development Model?
First and foremost, we should take a look at our software and the main purpose of it. We are developing an e-reader application, where users can keep their books and access them for reading and managing. An agile process model has many strengths, one of these being an emphasis on simplicity combined with quick and iterative development. For our purposes, the Bookkeeper application will have a quick initial development. The main functionality resides in just a few features: managing books in the form of adding or removing, viewing a book, and finding text within the book.

Simplicity
With simple functionality, an Agile development makes sense to start with. Agile aims to reduce the wasting of time, and our application would like to achieve that as well. Going further, our application has room to grow in the future that is all dependent on customer feedback. There are many e-reader applications out there. If we are unable to rapidly adapt and update our software according to the customer, we will fall behind and fail to prove that our application is the one to choose. Agile development methods expect system requirements to change, are built incrementally for this purpose, and have a backbone of customer involvement. All of these things make it look like the proper choice for developing Bookkeeper.

Customer Focus
As mentioned previously, there are many e-reader applications that have similar functionality to Bookkeeper. The difference in the future success of the application is dependent on the ability, and emphasis, to adapt this software based on the feedback our customer provides. With this in mind, we are ensuring ourselves that changes to the system will be made in the future. The main strength of the incremental aspect of Agile comes in here, as developing a new iteration will be less complex and require much less overhead. Not only will agile help the system be avoiding complexity, but it will also ensure our focus is on the customer feedback as for where we develop the system next.

Downsides of the Agile Approach
A key theme choosing the agile approach is maintaining simplicity, but what if our e-reader needed more functionality up front in order to support a wider variety of future developments and stand out? The first part of developing this application is to efficiently implement a system that cuts out any unneeded complexity. However, there is uncertainty in what a customer could desire in the future. A very simple system from the start could potentially make more complex

increments in the future a more challenging task, as core functionality may not support the complexity.

Alternative Models

Models such as the waterfall model with testing and validation after each step are not options for Bookkeeper. This is because they are heavily focused on detail after each step, and our plan for Bookkeeper does not require that level of validation. That model would suit something where security of the software is critical, and every step needs great attention to detail and time. However, our team had considered the general incremental process model, because the staggered linear sequences support some of the linear nature of our application. For example, we have a few ‘managing books’ functions (such as add a book, remove a book, and store a book) that would all follow a similar process. However, an agile approach is superior for us, because general incremental development has an emphasis on iterative testing as well, and we do not require that. Our initial development will not require much more than simple testing and instead should focus on an efficient and quick process.

5. [15 POINTS] Software Requirements including
5.a.) [5 POINTS] Functional requirements. To simplify your design, please keep your functional requirements in the range minimum 5 (five) to maximum 7 (seven). (Ch 4)

Requirement 1: The user must be able to load books from their system into the database.
Function: Load books from the system into the database

Description: Allows users to load books (as .txt, .pdf, etc. documents) from their system into the bookkeeper database

Input(s): the location of the desired book file (the exact location in the system or the relative path to the book file) and the bookkeeper database

Source(s): the book will be provided by the user's system, the specific file will be selected by the user, and the database will be provided by bookkeeper

Output(s): a new book in the user's library/bookkeeper's database

Destination: the bookkeeper database

Action: The user will provide the bookkeeper application with the desired book's file. If the book is of an acceptable file type, the file location is valid, and the file is accessible to the bookkeeper application, then the file will be uploaded to the bookkeeper database. Otherwise, the file will not be uploaded to the database. If the book is uploaded, then the user will be able to access the book and change or read it as they desire.

Requirements: The file must be of an acceptable file type. The file must be accessible to the bookkeeper application.

Preconditions: The user must have the bookkeeper software installed on their system. The desired file must already be downloaded on the user's system.

Postconditions: The new book is now a member of the database.

Side-effects: The new book may be a duplicate, which can cause confusion when working with other copies of the book in the database.

Requirement 2: The user must be able to delete books from the database.
Function: Delete books from the database

Description: Allows the user to delete books from the database.

Input(s): The book(file) that the user wants to delete from the database. The bookkeeper database.

Source(s): The book to be deleted will be selected by the user. The bookkeeper database will be provided by the bookkeeper software.

Output(s): An updated version of the bookkeeper database without the selected book.

Destination: The bookkeeper database.

Action: The user selects the book that they would like to remove from their library. If the book is a member of the database, then the book will be removed from the database. Otherwise, the book will not be removed.

Requirements: The selected book must already be present in the user's library (i.e. it has to be in the database already).

Preconditions: The selected book must be a member of the database.

Postconditions: The selected book is no longer a member of the database.

Side-effects: The removed book will no longer be accessible through bookkeeper. Lists or categories that the book was a part of may face issues if they are not updated to match the new contents of the database.

Requirement 3: The user must be able to add categories to books in the database.
Function: Add category to books

Description: Allows the user to add categories (tags) to books in the database.

Input(s): The selected book/file in the database. The category that the user would like to add to the book. The books that the user can select from (the books in the database).

Source(s): The bookkeeper database. The specific books and categories will be selected by the user.

Output(s): The selected book will have a new category added to it.

Destination: The bookkeeper database.

Action: The user selects a book in the database that they would like to add a category to. The user then selects a category that the book will be added to. We are assuming that the book will be a member of the database and the category is already defined. If the user has selected a category that the book is not already a member of, then the category will be added to the selected book. Otherwise, the category will not be added to the book.

Requirements: The book cannot already be a member of the category. The book must already be in the database. The category must already exist in the database too.

Preconditions: The book must already be a member of the database. The category must already exist and the category cannot contain the selected book.

Postconditions: The chosen category now contains a new book.

Side-effects: If a book's categories are changed such that all of its categories are now the same as another book already in the database, could cause a problem with duplication, making searching for a particular book more difficult. If multiple books are in many categories, then it also may become harder to search for particular books unless multiple categories are in the search criteria.

Requirement 4: The user must be able to delete categories from books in the database.
Function: Remove category from books

Description: Allows the user to remove categories from books in the database.

Input(s): The selected book in the database. The selected category that the book is a part of.

Source(s): The bookkeeper database. The books and categories will be selected by the user.

Output(s): The book will no longer be a member of the selected category. The book and category will still be members of the database.

Destination: The bookkeeper database.

Action: The user selects a book in the database and a category that they would like to delete the book from. If the book is already a member of the category, then the book will be removed from the selected category. Otherwise, nothing will happen.

Requirements: The book and category must already be members of the database.

Preconditions: The selected book must be a member of the selected category already.

Postconditions: If the function was successful, the selected book is no longer a member of the selected category. Otherwise, nothing was changed.

Side-effects: If the function was successful, then the chosen book is no longer a member of the selected category. Thus the application may face issues when searching for books of that specific category if the category and book's information are not updated.

Requirement 5: The user must be able to search their books via text query.
Function: Search book by text query

Description: Allows the user to search through books in the database based on text queries.

Input(s): The text that the user would like to search. The books in the database.

Source(s): The user will supply the text and the books will come from the database.

Output(s): The function will return search results from the books in the database. If the text query is found in a book in the database then the function will return the book and the book's contents containing said materials.

Destination: The user's screen/view.

Action: The user will provide a text query to the bookkeeper application. Bookkeeper will then search for the text in the books in the database. If the text is present in a book, then the book and its contents will be returned. Otherwise, the application will not return anything.

Requirements: The user must enter a text query. The application will not search for empty strings. The application can only search through books in the database.

Preconditions: There must be at least zero books in the database. (If the database is empty, the results will automatically be empty.)

Postconditions: The database will remain the same size and contain the same books.

Side-effects: None.

Requirement 6: The user must be able to view books in the database.
Function: The user must be able to view books in the database.

Description: Allows the user to view and read books that have been uploaded to the database.

Input(s): The database containing the user's books. The user's selected book.

Source(s): The database.

Output(s): The contents of the book that the user has selected.

Destination: The user's screen/the application's interface.

Action: The user chooses a book from the ones in the database. Bookkeeper then displays the book's contents on the user's screen. The user can then go on to read the book.

Requirements: The book that the user has selected must already be a member of the database.

Preconditions: The database cannot be empty.

Postconditions: The book's contents have been displayed to the user.

Side-effects: None.

5.b.) [10 POINTS] Non-functional requirements (use all non-functional requirement types listed in Figure 4.3 - Ch 4. This means provide one nonfunctional requirement for each of the leaves of Figure 4.3. You can certainly 4 make assumptions, even make up government/country based rules, requirements to be able to provide one for each. Please explicitly specify if you are considering such assumptions.)

Usability Requirements: The application should have a simple, clean user interface. Buttons should be labeled in a self-explanatory fashion, such as for opening, deleting, closing, or bookmarking, which shall utilize the common symbols already in use for such activities. The application shall facilitate accessibility by offering the user the choice of adjusting the color of the background and book text to make it easier on their eyes. As previously mentioned, our audio feature shall make the books accessible to people with vision loss.

Performance requirements: Scrolling and/or turning pages shall seem instantaneous to the user. No user action within a book shall exceed a response time of 100 ms. The database shall add and delete books rapidly, but may take up to 30 seconds if there are thousands of titles that need to be populated. If the user points the application to a local download folder. All actions within a book should execute within 100 ms to facilitate a fluid response time not noticeable to the user.

Space requirements: The application shall be compact, not more than 100 MB including all icons, code, and user interface elements.

Dependability requirements: The application shall be highly available with minimal bugs. Since the application is not dependent on the Internet, barring a malfunctioning or incompatible device, the application shall essentially always be available for use. If a bug is encountered, the application shall notify the user, save the user's last page, and gracefully and quickly restart, within 5 seconds, going back to where the user left off.

Environmental requirements: The application shall strive to use as little power as possible in order to preserve the user's battery life. To this end, scrolling and page turning animations shall be optimized for performance and utility, without ostentatious transitions. All EPA regulations will be adhered to in the making of this application

Security requirements: The application shall not contain any exploit that would allow a user to bypass the biometric or password based security of the device it is installed on. The application shall abide by all of the security requirements for the Apple App Store and the Google Play Store.

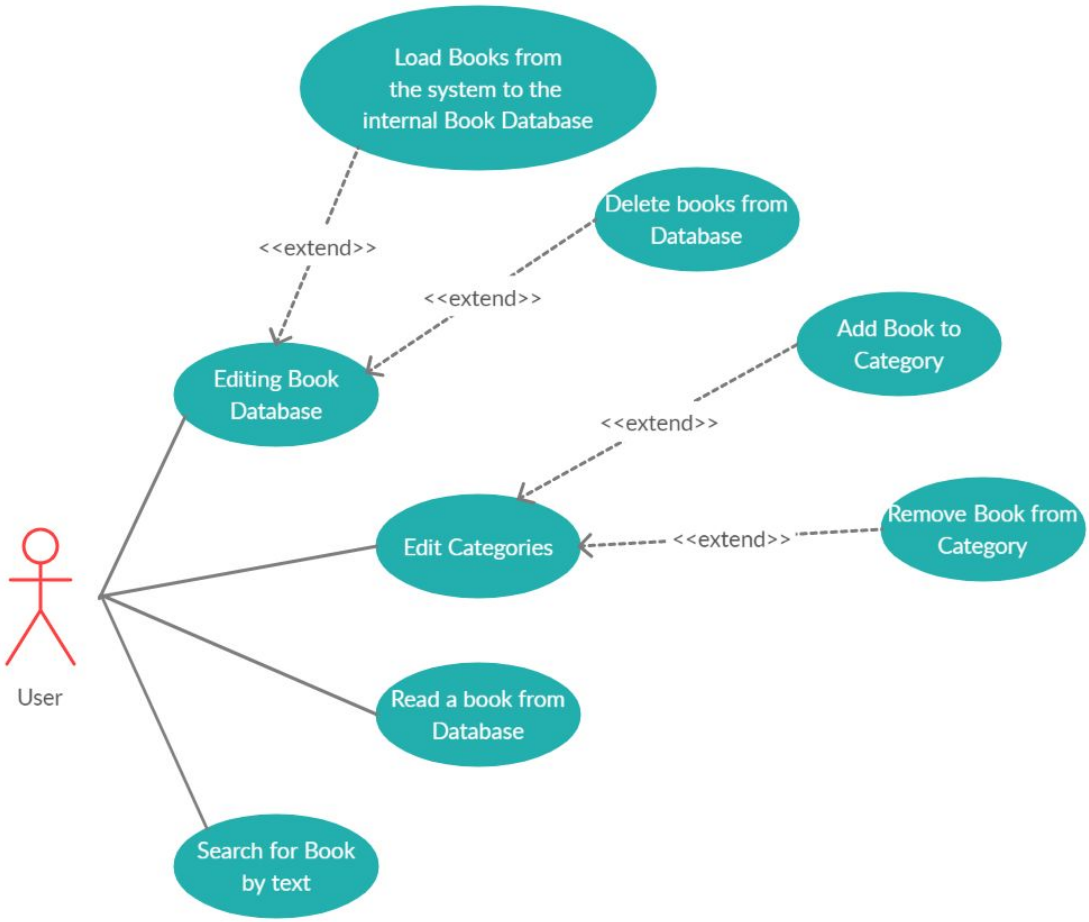
Operational requirements: The application shall be launched through a mouse button press via cursor, or via user's touch on the icon. All operations, such as bookmarking, annotating, and book management shall be possible using only a mouse cursor, only user touch input, or any combination of aforementioned devices.

Development requirements: Development shall proceed only under strict adherence to all labor laws of the United States. The application shall be constructed using the MVC model and an incremental, Agile approach in order to make efficient use of development time. The application shall be written in Java to meet performance and portability requirements. All developers shall make commits to the Bookshelf app on the project's Github. All commits shall be tested and reviewed before merge into master branch.

Accounting requirements: The application shall simultaneously track where a user left off in a book for each book that the user is reading. The application shall also track the number of pages the user has read, the time they have spent reading, and the number of times a title has been re-read.

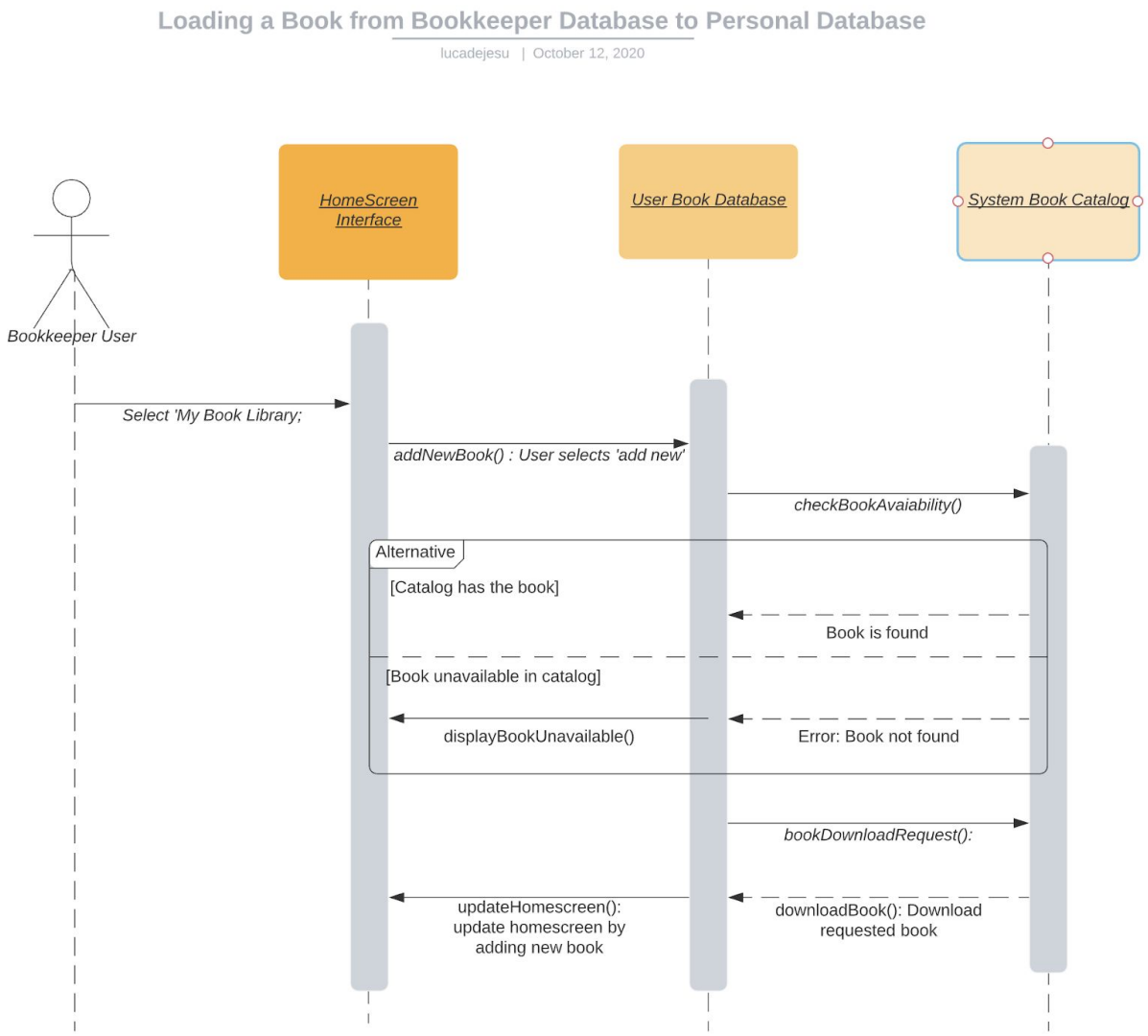
Safety/security requirements: The application shall not ask for nor store any personally identifying information of the user. The application will abide by all relevant OSHA requirements. The application shall not have any control over unrelated device functions, such as the flashlight, camera, or radios. The volume of the application shall not exceed the maximum device volume set by the user.

6. [15 POINTS] Use case diagram – Provide a use case diagram (similar to Figure 5.5) for your project. Please note than there can be more than one use case diagrams as your project might be very comprehensive. (Ch 5 and Ch 7)

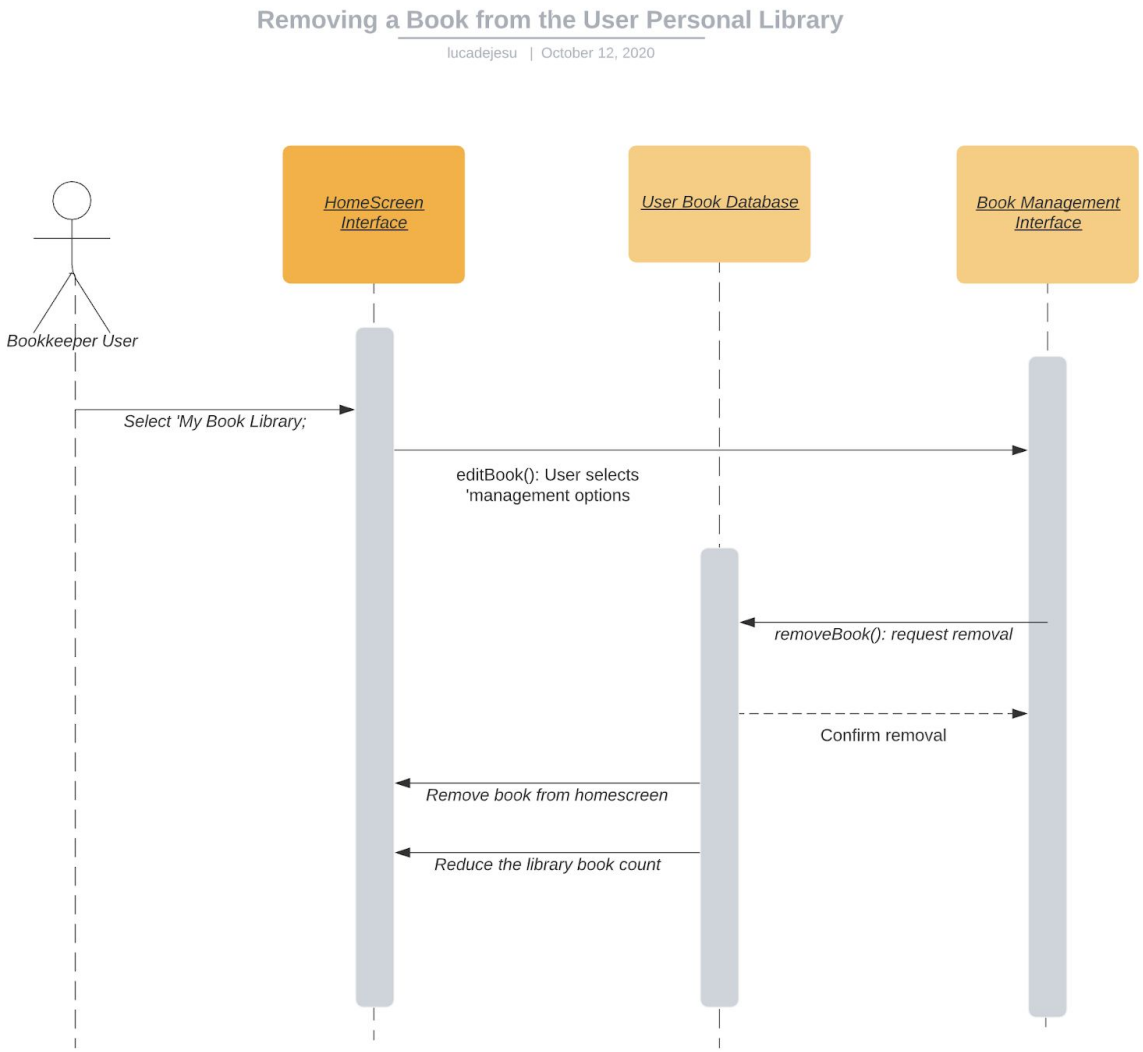


7. [15 POINTS] Sequence diagram – Provide sequence diagrams (similar to Figure 5.6 and Figure 5.7) for each use case of your project. Please note that there should be an individual sequence diagram for each use case of your project. (Ch 5 and Ch 7)

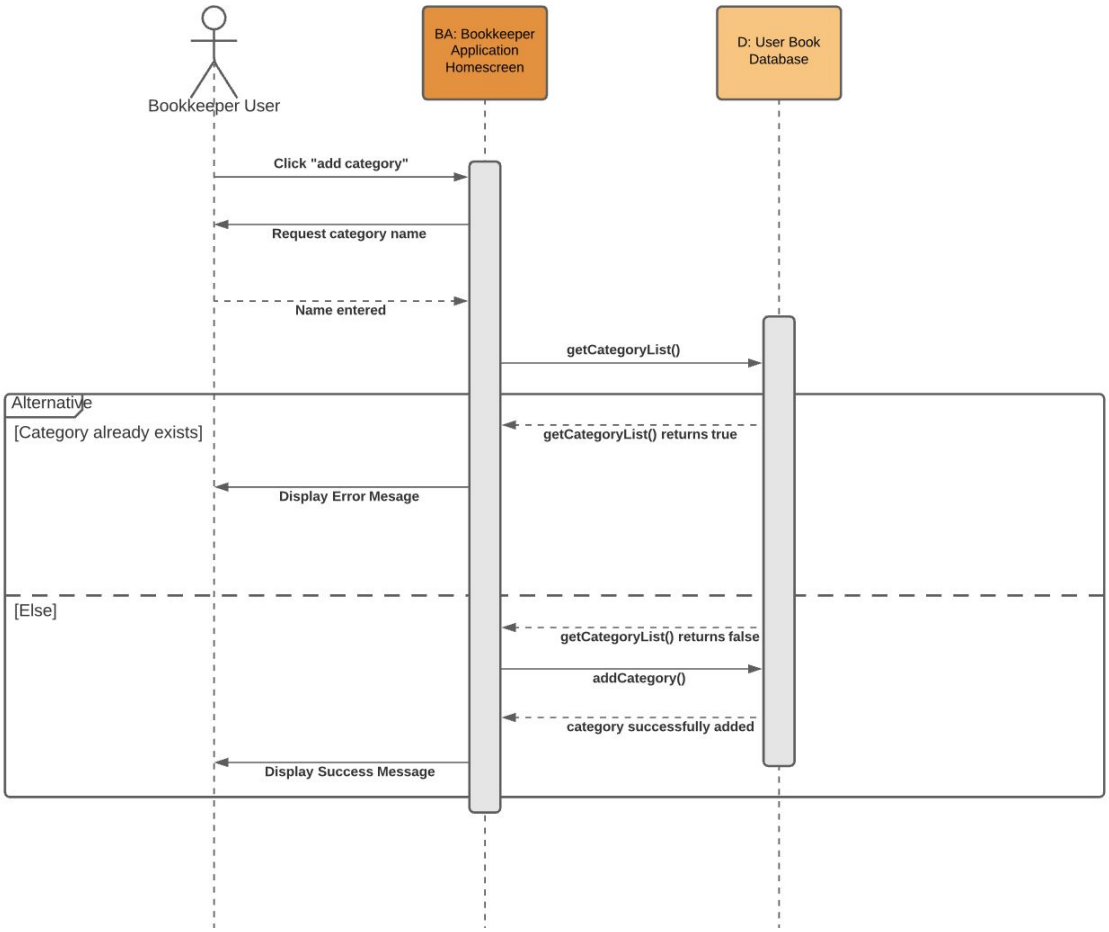
Sequence Diagram: Loading a book from the catalog to the user's library



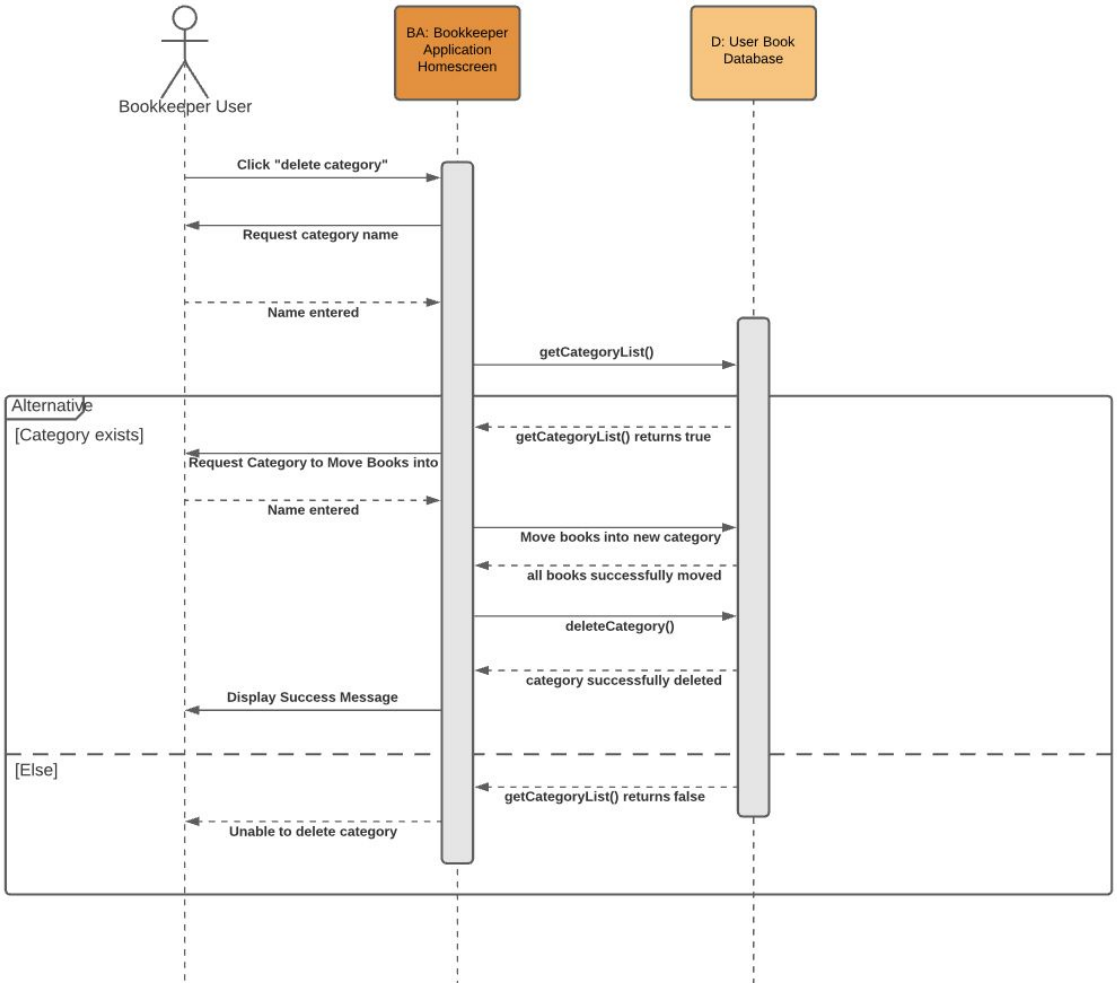
Sequence Diagram: Delete Books



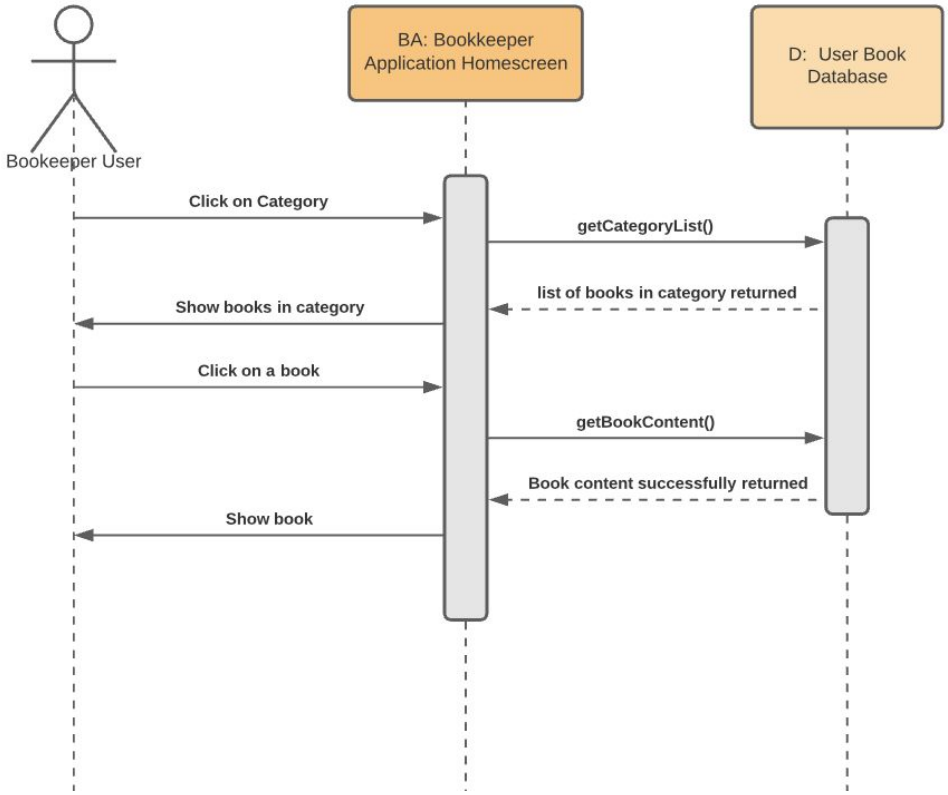
Sequence Diagram: Add Category



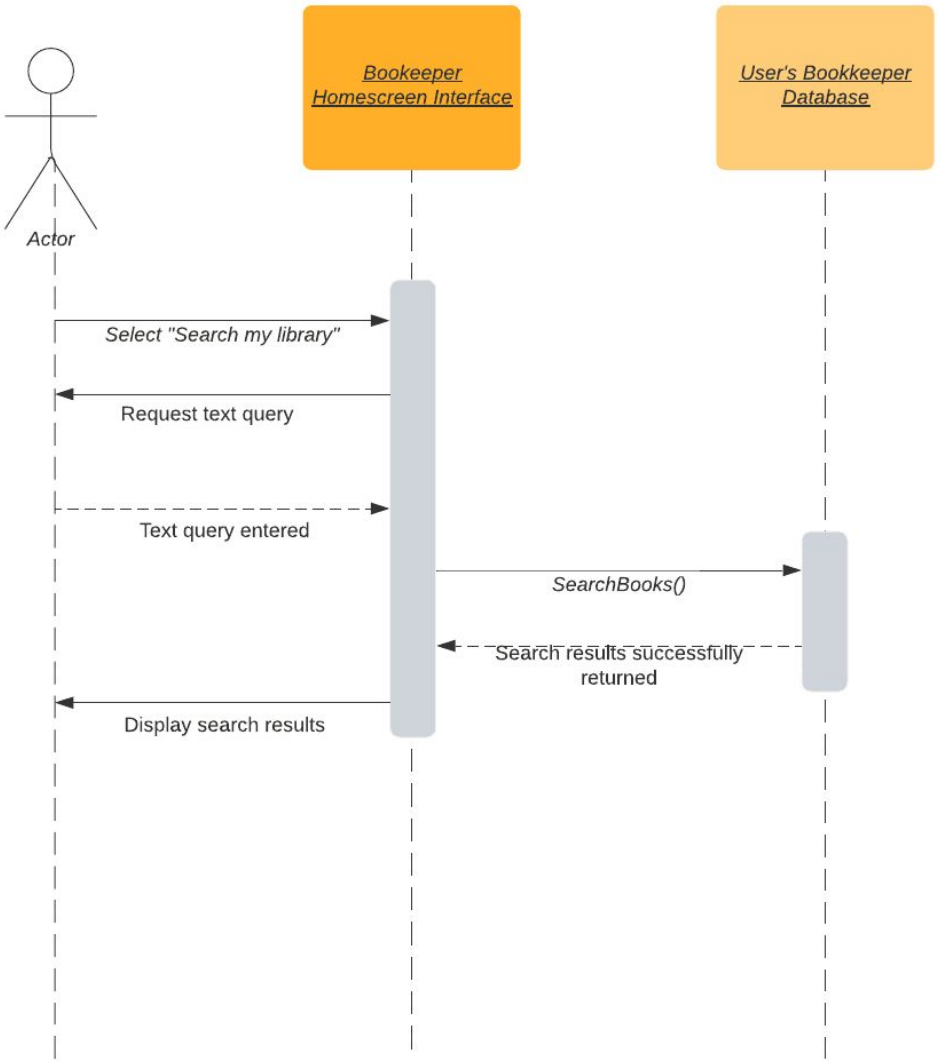
Sequence Diagram: Delete Category



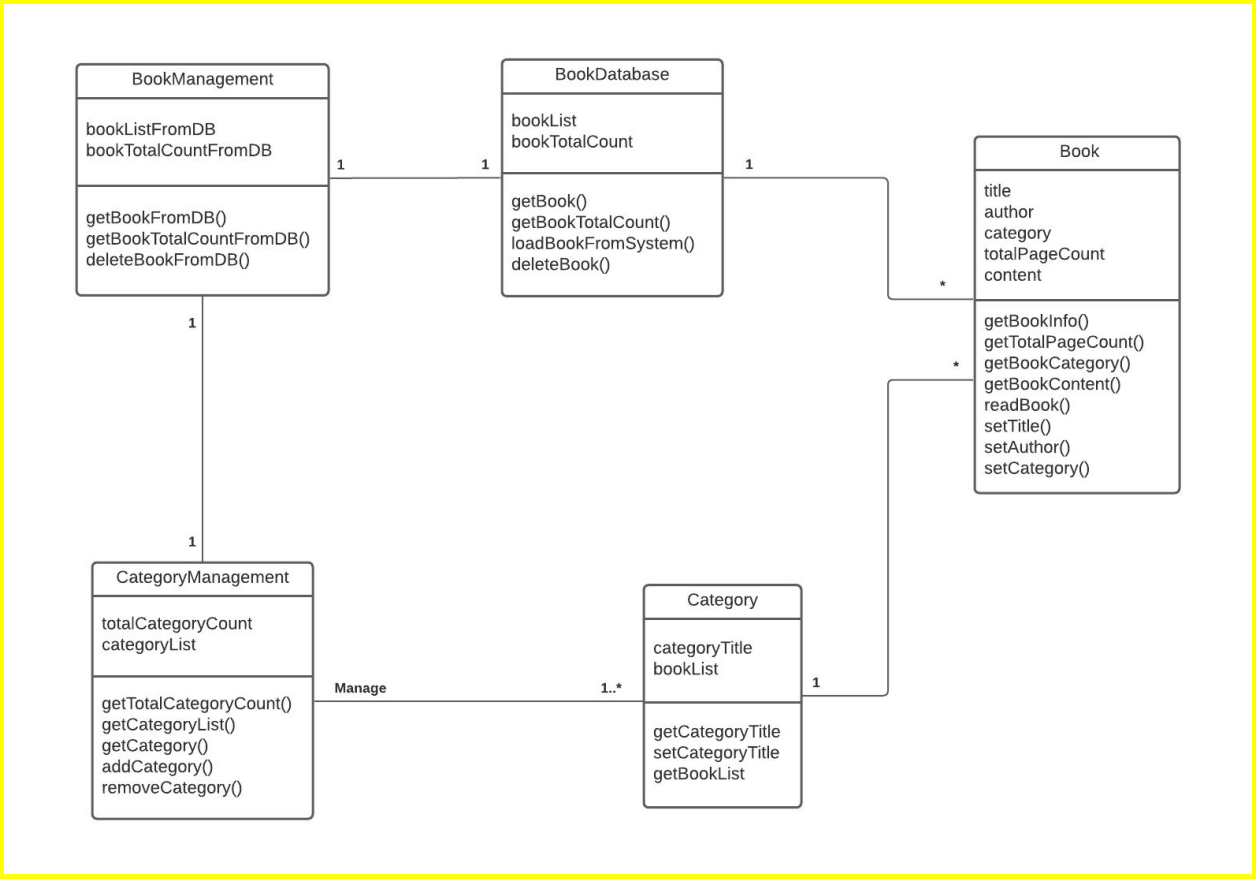
Sequence Diagram: View Book



Sequence Diagram: Search Books by text query

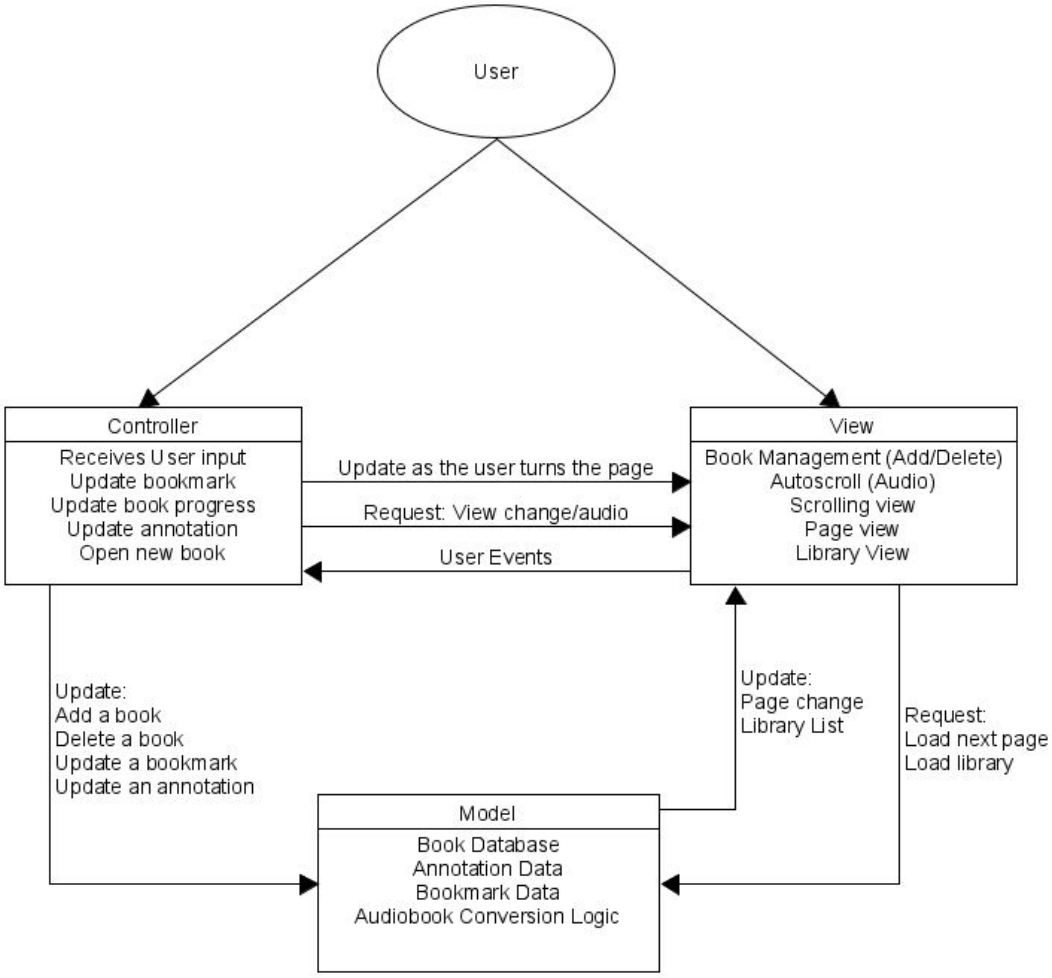


8. [15 POINTS] Class diagram – Provide a class diagram (similar to Figure 5.9) of your project. The class diagram should be unique (only one) and should include all classes of your project. Please make sure to include cardinalities, and relationship types (such as generalization and aggregation) between classes in your class diagram. Also make sure that each class has class name, attributes, and methods named (Ch 5).



9. [15 POINTS] Architectural design – Provide an architectural design of your project. Based on the characteristics of your project, choose and apply only one appropriate architectural pattern from the following list: (Ch 6 section 6.3)

Model-View-Controller (MVC) pattern (similar to Figure 6.6)



After a thorough analysis of alternatives, our group decided that The Model-View-Controller (MVC) pattern would be the best fit for Bookkeeper. MVC is commonly used for web applications, can have multiple different presentation views, and has separate components. We felt that the extensibility and flexibility offered by the MVC design's division of features into components would be particularly beneficial for an application which could easily be expanded to have Internet-based features, such as buying books from an online bookstore or retrieving books stored in an online drive.

The model component contains the book database. The database contains information on the annotations the user has made, where they left off in each book, and of course, their library of books. The

model also contains the logic for the automated conversion of text to audio for users who would rather listen to their books. This is the feature of our bookshelf app that distinguishes it from those currently available.

The view component contains book management (library), category management (such as sorting by genre), and page views. Multiple presentation views are needed to flip or scroll through books and show the user their library. The controller component handles requests and returns responses. Organizing the system into three parts, model (data), view (presentation), and controller (triggers events) allows data to be changed or updated without updating the other two components. This makes the application adaptable, manageable, and organized. While MVC has disadvantages, such as complex code for simple applications and potential uneven separation of features into different components, the benefits outweigh the drawbacks.

IEEE Citations

[1] M. Smith, “10 Best eBook Reader Apps for Android You Need to Know,” *Lifehack*, 03-Jan-2018. [Online]. Available: https://www.lifehack.org/articles/technology/10-best-ebook-reader-apps-for-android-you-need-know.html. [Accessed: 17-Oct-2020].

[2] D. Subramanian, “Easy Speech-to-Text with Python,” *Medium*, 08-Jul-2020. [Online]. Available: https://towardsdatascience.com/easy-speech-to-text-with-python-3df0d973b426. [Accessed: 17-Oct-2020].