CSCD01: Deliverable 3

Team 32: CCF

Angela Zhu
Dasha Besshaposhnikova
Michelle Pasquill
Mohammed Osumah

Table of Contents

Potential features

Feature #1: Accessibility

[link issue and description]

Currently the pdf viewer has implemented keyboard shortcuts for navigating most of the toolbar. It is however missing shortcuts for some classes. There is the main toolbar that always appears. Because it is always shown, it does not allow arrow key navigation as that is reserved for navigating the pdf. It allows you to tab through all elements in the toolbar. On top of this are three additional classes: the secondary toolbar, the findbar, and the sidebar. These appear to have been implemented separately and thus the shortcut implementation is inconsistent.

The secondary toolbar can only be opened by tabbing and hitting enter. Unlike the other two, there is no keyboard shortcut. Once it is opened, it can be closed using the esc key or by tabbing and hitting enter again. The functionality for this is set in the keydown function in app.js which binds an eventListener to the window. Once the toolbar is opened, you can only tab through it and cannot navigate using arrow keys.

The findbar can be opened by tabbing + enter and also with the shortcut ctrl+f. Once it is opened, it can be closed with the esc key, but you cannot close it again with ctrl+f. It can also be closed with tab+enter. Ctrl+f is implemented in app.js just like the secondary toolbar. However, the esc key as well as enter (for opening) are implemented in pdf_find_bar.js instead and directly adds an eventListener to the ‘bar’ element in the HTML document. Once the find bar is opened, you can only tab through it and cannot use arrow key navigation.

The side bar can be opened by tabbing + enter and also with the shortcut F4. Once it is opened, it can be closed again with the same shortcut F4. Esc will not close the sidebar. You can also tab+enter to close it. The sidebar can be navigated with tabbing but not arrow keys. This functionality is set in app.js the same as the secondary toolbar.

Feature #2: Don't hide toolbar buttons in drop-down menu unnecessarily

Link: https://github.com/mozilla/pdf.js/issues/9200

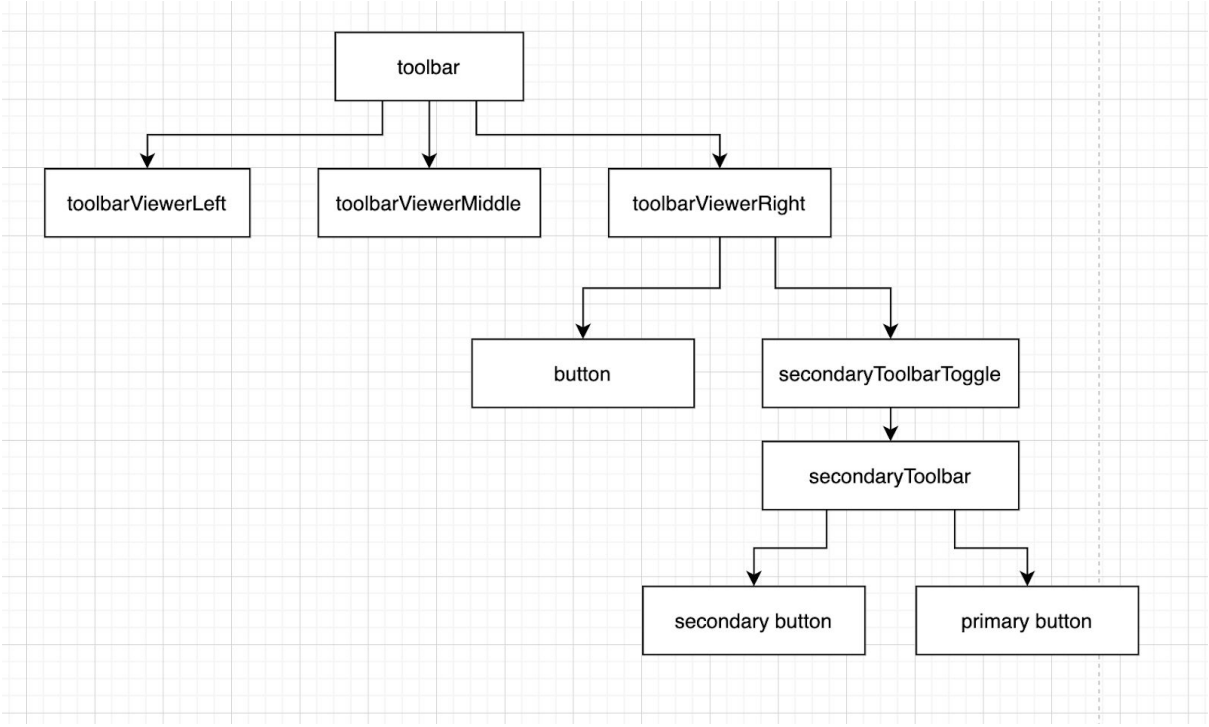
The PDF viewer has a dropdown menu that can be accessed by clicking the arrow button in the top right corner of the toolbar. This feature request is suggesting to move the options from the dropdown menu to the toolbar itself, as long as there is enough space on the screen. In other words, the user who suggested the feature only wants to see the dropdown if there is not enough space for all the actions in the toolbar.

Here is a screenshot of what the current toolbar with the open dropdown menu looks like:

In the code, specifically in viewer.html, the toolbar is divided into 3 sections: toolbarViewerLeft, toolbarViewerMiddle, and toolbarViewerRight. The dropdown menu content appears when the secondaryToolbarToggle button (in toolbarViewerRight) is clicked. The dropdown menu is in a div called secondaryToolbar. Currently, the layout of the toolbar is loaded statically, with the configuration of features and toggle buttons defined in viewer.js.

In order to implement this feature, we would need to modify viewer.js and viewer.html so that the toolbar buttons are loaded dynamically. We would need to ensure that the layout is responsive, meaning that buttons move to the dropdown menu when the screen width is decreased and move back to the toolbar when the screen width is increased. This is already partially done with some of the icons in the toolbar, so we would need to extend this functionality. We would need to add buttons to the toolbar, and keep their secondary counterparts in the dropdown menu (as is done with the current buttons), then make sure that the correct version is shown depending on the screen size, so either there will be a button in the toolbar or an option in the dropdown menu for a particular function.

The following UML diagram shows a hierarchy of HTML elements in viewer.html. Specifically, it shows the elements involved in this feature request.


toolbar
toolbarViewerLeft
toolbarViewerMiddle
toolbarViewerRight
button
secondaryToolbarToggle
secondaryToolbar
secondary button
primary button

Chosen feature and justification

We have chosen to focus on accessibility as our feature for this project. As described earlier, this feature will include fixes to the following two issues listed on PDF.js:

WAI-ARIA enhancement for toolbars: https://github.com/mozilla/pdf.js/issues/9556
Accessible text flow: https://github.com/mozilla/pdf.js/issues/9554

We made the decision to combine these two into the feature we will be implementing because we feel that accessibility should be a priority in any web based product. The current application can be difficult to navigate for someone who relies on a screen reader. We made a much smaller accessibility fix in the previous deliverable to allow for a screen reader user to skip to the navigation bar or the main PDF. In making and testing this, we came to realize that navigating the page using the keyboard can be quite tedious and somewhat confusing. There are not enough ARIA labels in PDF.js, so it can be unclear what state certain buttons on the toolbar are in. We intend to improve the toolbar's accessibility and add more keyboard shortcuts following the suggestions made in the issue *WAI-ARIA enhancement for toolbars*. Additionally, the way the PDF is parsed and rendered leads to unnatural speech for the screen reader. We hope to fix this problem, or at the very least improve it, so that reading a PDF is not too difficult for screen readers.

Many of our group members have done work with web accessibility in the past, so we felt that these two issues were a feature we would be able to implement. Given our skills, we are confident in our ability to improve the toolbars within the time we have. The *Accessible text flow* problem will be a bit more difficult for us since we are not as familiar with the parsing and rendering of PDF files, and it will require a change in the core functionality of PDF.js. We believe we will be able to improve upon the current text flow within the time limit. By combining these two issues into one feature, we believe the feature is sufficiently difficult, and we know we will be able to deliver some functionality.

Implementation plan

[insert some introduction sentence]

The function webViewerKeyDown in app.js handles all eventListeners for the window for key inputs. Here we will add a case for a new shortcut that will toggle the secondary toolbar - it will check if the toolbar is open and then open or close accordingly. We will also modify the case for ctrl+f so that the find bar can be open and closed with the shortcut.

In pdf_find_bar.js, in the function that adds eventListeners to the ‘bar’ element, we will also add input for arrow keys. This will allow one to go through the find bar with the left and right arrow keys. It will only apply when the find bar is open. Will will do the same in secondary_toolbar.js - add eventListeners to allow input for arrow keys when open.

Acceptance tests

1. Toolbar tests

1.1. Given that the user is on the web page, when they tab or use a landmark to reach the toolbar, then the toolbar should be highlighted and focus moved to the first element in the toolbar.

1.2. Given that the user is tabbed to a toolbar, when focus is put on an element in the toolbar, then the in-focus element should very clearly be highlighted.

1.3. Given that the user has tabbed to a toolbar, when they use the right arrow key, then the focus will move to the next element in the toolbar.

1.4. Given that the user has tabbed to a toolbar and the focus is on the last element in the toolbar, when they use the right arrow key, then the focus will move back to the first element in the toolbar.

1.5. Given that the user has tabbed to a toolbar, when they use the left arrow key, then the focus will move to the previous element in the toolbar.

1.6. Given that the user has tabbed to a toolbar and the focus is on the first element in the toolbar, when they use the left arrow key, then the focus will move to the last element in the toolbar.

1.7. Given that the user has tabbed to a toolbar and has clicked either the left or right arrow key, when the next element is disabled, then the focus will still move to that element and a screen reader should announce that it is disabled.

1.8. Given that focus is on a spinner element in the toolbar, when the user uses the up arrow or down arrow, then the spinner value will cycle to the previous or next option respectively.

1.9. Given that focus is on an element in the toolbar which opens a sub-toolbar, when the Enter key is clicked, then the sub-toolbar will open and have focus moved to the first element in the toolbar.

1.10. Given that a sub-toolbar is open, when the up or down arrow key is pressed, then the focus will cycle through the elements inside.

1.11. Given that a sub-toolbar is open, when the Escape key is pressed, then the sub-toolbar is closed and focus moves back to the button in the main toolbar which opened it.

1.12. Given that focus is not on a toolbar, when the user presses the left or right arrow key, then the PDF will switch to the previous or next page respectively.

1.13. Given that focus is not on a toolbar, when the user presses the up or down arrow key, then the PDF will scroll up or down respectively.

2. Text flow tests

2.1. Given a PDF with a bullet point list, when reading it with a screen reader, there should be an obvious pause between each bullet point.

2.2. Given a PDF with line breaks, when reading it with a screen reader, there should be an obvious pause after a line break.

2.3. Given a PDF with a line that has different styling mid-line (such as a line with a bolded word), when reading it with a screen reader, there should be a small pause between each spoken word.

2.4. Given a PDF where a line wraps mid-word, when reading it with a screen reader, the wrapped word should sound like one word.