

CSCD01: Deliverable 4

Team 32: CCF

Angela Zhu

Dasha Besshaposhnikova

Michelle Pasquill

Mohammed Osumah

Table of Contents

Implemented Feature	3
Code Design	4
Acceptance Tests	6
Software Development Process	8

Implemented Feature

The feature we implemented for this deliverable was based around making PDF.js easier to navigate for those with accessibility issues. Our focus was to change keyboard functionality to follow WAI-ARIA guidelines, and we based it off of the following suggestion:

WAI-ARIA enhancement for toolbars: <https://github.com/mozilla/pdf.js/issues/9556>

We modified all of the navigation bars on the page to follow the WAI-ARIA suggested method called roving tabindex (described here:

<https://www.w3.org/TR/wai-aria-practices-1.1/#toolbar>).

This is also the recommended method from Mozilla, which PDF.js is a part of (described here:

https://developer.mozilla.org/en-US/docs/Web/Accessibility/Keyboard-navigable_JavaScript_widgets).

In addition to adding the suggested arrow key navigation using roving tabindex, we also added a few additional features to help with accessibility. We added keyboard shortcuts for the tools on the toolbar, as well as tooltips for each button stating the keyboard shortcut. We added blue borders to the in-focus elements to make it more obvious which element is currently highlighted, since before it was just a slightly darker grey which is a terrible contrast for people with colour contrast issues. Additionally, we fixed some problems with the keyboard navigation which made certain elements inaccessible for people who rely on screen readers. This includes allowing check boxes to be accessed with the keyboard, and also allowing the user to focus on the document properties pop-up window so that screen readers can actually read it.

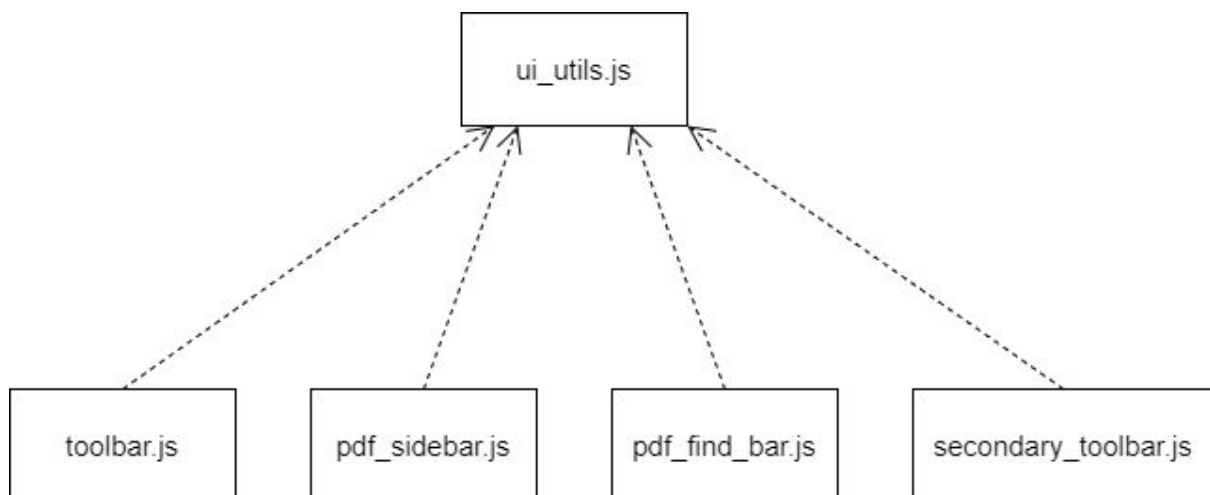
The user guide for the feature can be found under our Deliverable 4 folder on GitHub.

Code Design

In order to implement arrow key functionality following the roving tab-index method, all tab-indexes in “viewer.html” were modified and each toolbar class was given a new property which stores the buttons in order of how they should be focusable and a new private method named “_bindKeyListeners()”.

The property, “focusOrder” stores a list of the buttons in the toolbar in the order that the arrow keys should focus through them. In order to do this, we needed to make sure that all the toolbar’s buttons were passed into the constructor for the object (since a few buttons were not included in construction of the toolbar object). Additionally, in the case where the focus order could change, we needed to add a property that would dynamically change with it with the help of a new method.

Each toolbar class was given a private method named “_bindKeyListeners()”. This method is called when a new toolbar is initiated with the constructor method. The method adds keyboard event listeners to each focusable element in the toolbar, depending on how that toolbar should react on different key presses. It handles the arrow key events as well as other events that may be unique to each toolbar. The arrow key events rely on two new functions which were added in “ui_utils.js”, which are “setFocusPrevious(item, toolbar)” and “setFocusNext(item, toolbar)”. These two functions are given the current in-focus item and the toolbar it is in, and will decide which element should be in-focus next based on the toolbar’s “focusOrder” property. Since the functionality for this is the same for each toolbar, we put it in the file “ui_utils.js” which defines any generic ui functions, and import the functions into each class.



Acceptance Tests

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.
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.
3. Given that the user has tabbed to a horizontal toolbar, when they use the right arrow key, then the focus will move to the next element in the toolbar.
4. Given that the user has tabbed to a horizontal 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.
5. Given that the user has tabbed to a horizontal toolbar, when they use the left arrow key, then the focus will move to the previous element in the toolbar.
6. Given that the user has tabbed to a horizontal 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.
7. 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.
8. 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.
9. Given that a vertical toolbar is open, when the up or down arrow key is pressed, then the focus will cycle through the elements inside.
10. Given that the secondary toolbar is open, when the Escape key is pressed, then the secondary toolbar is closed and focus moves back to the button in the main toolbar which opened it.
11. 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.

12. 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.
13. Given that the mouse cursor is over a button in a toolbar, then a tooltip will display which states the function of the button and any keyboard short-cut it may have.
14. Given that a button has a keyboard short-cut, when the user uses the short-cut then the click event of the button should activate.
15. Given that the user is using a screen reader and tabs through the page then focus should eventually move to the PDF content so they can read the content of the page.
16. Given that a user has opened the document properties window, then focus should be moved to that window so that a screen reader can read it.
17. Given that the document properties window is open, then the user should be able to tab to the close button to close the window or use the Escape key to close the window.
18. Given that the document properties window is open, then no other elements of the page should be accessible via keyboard activity until it has been closed.


Software Development Process

For this deliverable, we continued following the Kanban software development process. Just as before, we used our Trello board with modifications to ensure that we are following the Kanban methodology correctly.

We started by having a meeting and creating the tasks that needed to be done for this deliverable. These tasks were listed in our backlog.

Once we had our major tasks listed out, we split up the larger tasks into smaller, more manageable tasks so that multiple people could work on them at once, and so that we could communicate more easily about what needed to be done.

An example showing the importance and usefulness of a code review:

 Activity Hide Details


MP

Write a comment...

MP

Michelle Pasquill a few seconds ago

Moved back to to-do since screen readers can't read the document properties box. Focus should be moved to the box first rather than the close button.

 - [Edit](#) - [Delete](#)

MP

Michelle Pasquill moved this card from Code Review to To Do
a minute ago

A

angelamz moved this card from Doing to Code Review
13 hours ago