team.name V2

team_03

React-Bootstrap

Deliverable 4

Members:
Alexei Coreiba
Gaganpreet Kaberwal
Harsh Patel
Kohilan Mohanarajan
Kanstantsin Ilioukevitch

Supervisor:
Prantar Bhowmik
Anya Tafliovich

April 8, 2020

Contents

1 Selected Issue: Issue #4911

As described in Deliverable 3, we decided to implement the necessary functionality in order to close issue #4911. This issue is tied to the `<ListGroup>` component. The issue is trying to clear up a shortsighted implementation of nested `<ListGroup.Item>`s. When there is a `<ListGroup.Item>` that is nested within a `<ListGroup.Item>` that is marked as `active`, the nested `<ListGroup.Item>` would rendered with white text instead of being marked as `active` as it's parent. Figure 1 below shows this behaviour. The expected behaviour is shown in Figure 2 below.

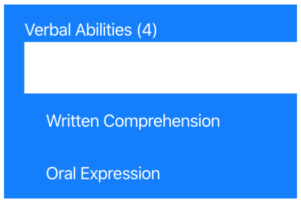


Figure 1: Demonstration of current behaviour as described by issue #4911

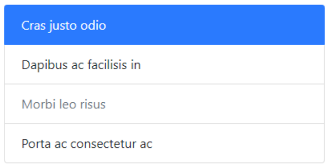


Figure 2: Demonstration of expected behaviour as described by issue #4911

1.1 Issue

The issue is that `ListGroup` and `ListGroupItem` have an `active` property which is only explicitly propagated or cascaded. Meaning if a parent `ListGroupItem` has the `active` prop set to true, only it will be `active` and not its children. The feature we are adding to the `ListGroup` and `ListGroupItem` components is the propagation or cascading of the `active`. This will help the user as they will no longer have to explicitly state `active` for every single sub component of type `ListGroup` or `ListGroupItem` and can apply that property with just one change. By using `cascadeaction` all the sub component will automatically become `active`. To then not have a `active` subcomponent the `disabled` property to then remove that `active`.

1.2 Solution

To solve this we need to pass the prop `active` down to children that are either `ListGroup` or `ListGroupItem` and no other component type. This involves changing props of the children of our current node. Props are immutable in react so work around must be thought of to implement this.

1.3 Implementation

In react we can clone child nodes using `React.cloneElement` and `React.Children` to properly iterate over and create clones with the right props set. By adding `cascadeActive` prop to both the `ListGroup` and `ListGroupItem` we can check if the children of the current node need to be set to active or not. If this is set to `true` we set all the children that are of type `ListGroup` and `ListGroupItem` to `active=true` and `cascadeactive=true`. This continues recursively.

2 User Guide

The original user guide for the `<ListGroup>` component can be found in the React-Bootstrap documentation. The only difference that we would change is supporting a nested `<ListGroup.Item>` functionality. As was already described above, it is possible to have nested `<ListGroup.Item>` components, and our solution ensures that if a parent is set to `active`, any child will subsequently be set to `active`. To do this, the user must add the `active` attribute to the parent component, which automatically carries over to the child component, provided that the child component isn't already set to `disabled`.

3 Testing

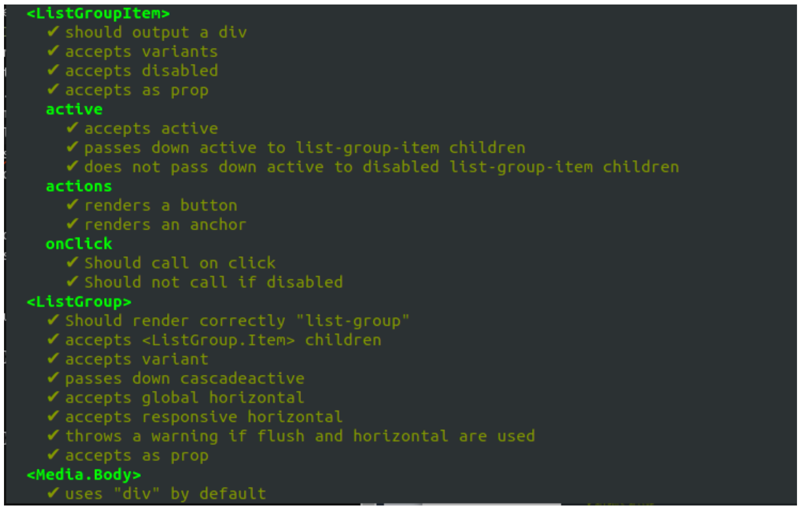
3.1 Unit Testing

As described in Deliverable 3, the project uses Enzyme.js, a Javascript library used for React unit testing, to unit test all functionalities of their components. All changes that are made must pass any existing unit tests, as well as, any new ones that you add to test your change. A disclaimer that all following links will be of our forked react-bootstrap repository, or our original team-3 github repository. The existing unit tests for the `<ListGroup>` component can be found in the ListGroupItemSpec test file, and the ListGroupSpec test file. These test files can be found in the test directory.

There are two differences between these two files. The `ListGroupSpec` file deals with testing the `<ListGroup>` functionalities; whereas the `ListGroupItemSpec` file deals with testing the `<ListGroup.Item>` functionalities. Since this issue is testing whether the `active` tag will propagate down into the `<ListGroup.Item>` child, this means we must do any unit testing in the `ListGroupItemSpec` file.

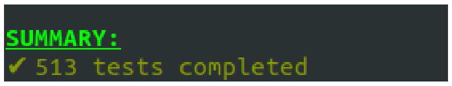
The unit testing for this issue is very simple, all that is required is to simulate the described nested behaviour - a nested `<ListGroup.Item>` within a `<ListGroup.Item>` that is set to active - followed by checking whether the current simulation has two `<ListGroup.Item>` div objects that are set to active. If this is is not the case, then the unit test has failed. The unit test can be found here. Restructuring of the current unit tests were required to combine a previously completed unit test into one section with the newly created unit test as both are related to the `active` functionality of a `<ListGroup.Item>`. An interesting edge case that was thought of is if any nested `<ListGroup.Item>` components are set to disabled, then they should remain to be left as `disabled` instead of being switched to `active`. A unit test for this edge case was created to ensure that this would not occur. The implementation of this unit test can be found here. Lastly, upon doing our final pull request into the React-Bootstrap project, their checks told us to increase code coverage for the ListGroup.js file, and thus we created the following unit test to ensure our solution is better covered under unit testing. After this unit test was introduced, all the checks required to be considered for a pull request passed.

After the implementation was complete, the tests ran without failing, implying that our solution was successful. Found underneath are screenshots of the successful unit test run.



```
<ListGroupItem>
  ✔ should output a div
  ✔ accepts variants
  ✔ accepts disabled
  ✔ accepts as prop
  active
    ✔ accepts active
    ✔ passes down active to list-group-item children
    ✔ does not pass down active to disabled list-group-item children
  actions
    ✔ renders a button
    ✔ renders an anchor
  onClick
    ✔ Should call on click
    ✔ Should not call if disabled
<ListGroup>
  ✔ Should render correctly "list-group"
  ✔ accepts <ListGroup.Item> children
  ✔ accepts variant
  ✔ passes down cascadeactive
  ✔ accepts global horizontal
  ✔ accepts responsive horizontal
  ✔ throws a warning if flush and horizontal are used
  ✔ accepts as prop
<Media.Body>
  ✔ uses "div" by default
```

Figure 3: Unit testing of the ListGroup.Item component



```
SUMMARY:
✔ 513 tests completed
```

Figure 4: Unit testing of all components complete

3.2 Acceptance Testing

Since this is project is a frontend Javascript framework, our acceptance testing is very simple. We would have to simulate the exact behaviour that this issue is trying to bring attention to. If this no longer occurs, then we would know that our solution has passed the acceptance testing. Our acceptance test can be found in the original team-03 github repository within the test-app folder. It is a React-Bootstrap frontend document that implements the exact behaviour as described in the original issue. The App.js file, the source file for the behaviour, can be found here.

Again, we are looking for a similar behaviour as seen in Figure 2 to ensure that our solution has passed the acceptance test. Otherwise, if the behaviour continues to be similar to the behaviour seen in Figure 1, then we know that our solution has not passed the acceptance test.

After the implementation was complete, the acceptance test passed our visual check, meaning that our solution was successful. Found underneath are screenshots of the successful acceptance test run with no console errors. The warnings that can be seen are related to other components outside of this issue.

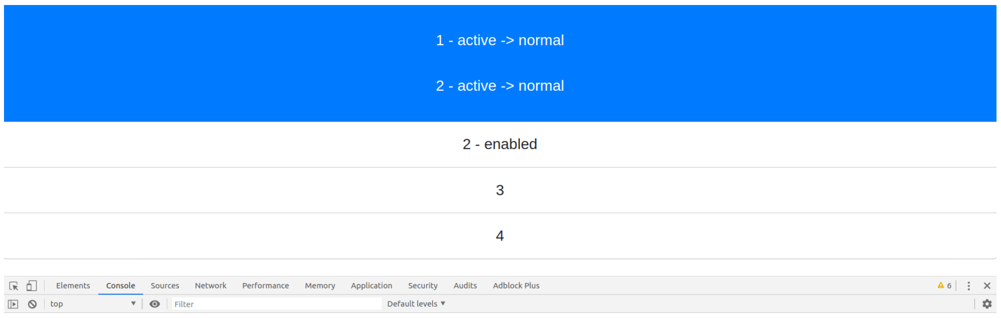
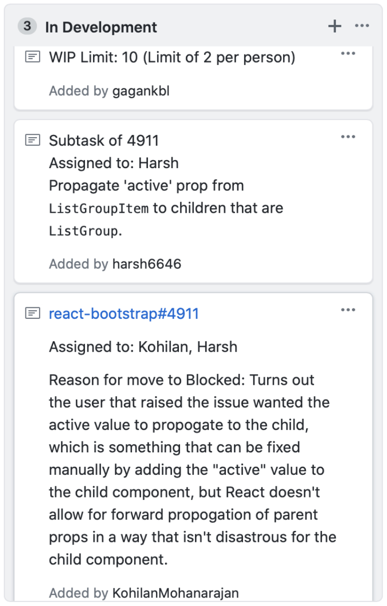


Figure 5: Testing of the ListGroup.Item component

4 Software Development Process

As we planned before, we utilized both Kanban and XP in our approach for this deliverable. By combining the best of both worlds, we progressed further and faster than if we have done otherwise.

For the kanban methodology, we used the github project board (as highlighted in an earlier phase) to visualize the workflow. WIP limits were set to limit over-assigning tasks under a specific lane. Each lane had specific policies set that must be adhered to when moving any issue into that lane. Most importantly, feedback loops were implemented using a "Code Review" section on the board which incorporated peer programming/reviewing. The issue that was being worked on this time was something that required more dev effort than previous cases, so to make the process efficient, the issue was split into sub-tasks.


3 In Development

WIP Limit: 10 (Limit of 2 per person)

Added by gagankbl

Subtask of 4911
Assigned to: Harsh
Propagate 'active' prop from `ListGroupItem` to children that are `ListGroup`.

Added by harsh6646

react-bootstrap#4911

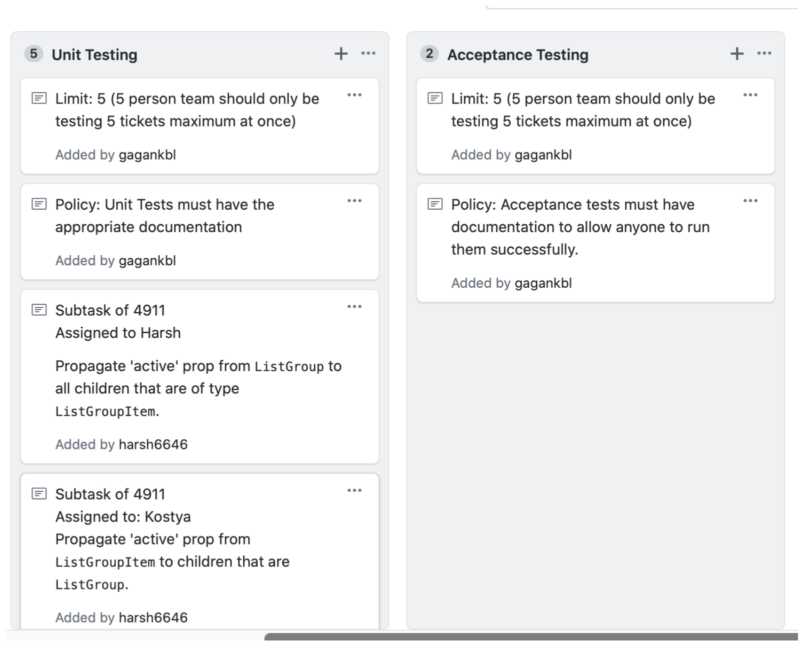
Assigned to: Kohilan, Harsh

Reason for move to Blocked: Turns out the user that raised the issue wanted the active value to propogate to the child, which is something that can be fixed manually by adding the "active" value to the child component, but React doesn't allow for forward propogation of parent props in a way that isn't disastrous for the child component.

Added by KohilanMohanarajan

Figure 6: Splitting the main issue into sub-tasks

Something different this time was that we incorporated unit testing in terms of Enzyme tests which was explained up. Previously acceptance testing was only done, so this time a new lane needed to be added to the kanban board to help visualize the issue movements into its appropriate sections. Similar WIP limit and similar policy was applied to this lane when compared to the acceptance testing lane.


5 Unit Testing

Limit: 5 (5 person team should only be testing 5 tickets maximum at once)

Added by gagankbl

Policy: Unit Tests must have the appropriate documentation

Added by gagankbl

Subtask of 4911
Assigned to Harsh

Propagate 'active' prop from `ListGroup` to all children that are of type `ListGroupItem`.

Added by harsh6646

Subtask of 4911
Assigned to: Kostya
Propagate 'active' prop from `ListGroupItem` to children that are `ListGroup`.

Added by harsh6646

2 Acceptance Testing

Limit: 5 (5 person team should only be testing 5 tickets maximum at once)

Added by gagankbl

Policy: Acceptance tests must have documentation to allow anyone to run them successfully.

Added by gagankbl

Figure 7: Addition of new 'Unit Testing' lane for testing

We had multiple meetings throughout this phase to discuss and finalize decisions, troubleshoot any bottlenecks we may have run into, and most importantly develop and test the issue we decided to proceed with.

- Meeting 1: Looked and pros and cons of all the possible issues we could tackle. Calculated the dev effort required and investigated all the files that might need to be changed/not changed in the react-bootstrap code base. These findings we properly documented in an earlier phase of this project.

- Meeting 2: After working on the chosen issue (#4911) for a couple of days, held a meeting to decide whether or not to switch issues as the current one was giving us a lot of trouble. We came to the conclusion that we would spend one day to further explore another issue (#3939) to see if it would be more feasible.

- Meeting 3: We had a breakthrough in our thought process on how to solve the initial issue, so we decided to stick with the first ticket (#4911).

- Meeting 4: Our final meeting was held on the last day to consolidate and finish up all of our work that we wanted to submit. We used the opportunity to summarize everything that we have done and accomplished in order to make sure that we have done everything that needed to be completed. At the end of the meeting, we submitted everything and that was the end of the project.

In this deliverable, we wanted to apply certain XP principles in order to strengthen our team and increase the quality of our product. The first and most important approach that we took was to make our process test driven. This is why our unit tests were completed before the development started. This approach allowed our developers to be fully aware and knowledgeable of the component, what exactly it is supposed to achieve, and the details that they have to pay attention to. The exact details of the tests created could be seen in the section above.

Once the creation of the tests has been completed, the development stage started. Here, we made the conscience decision to pair program vigorously due to the complexity of the issue. Pair programming improved our communication throughout the team and allowed the team to help each other in different sections of the issue. This was all achieved through discord and the screen sharing feature that it offers. It allowed us to visually see and follow the person that was coding, and provide real-time useful inputs and discussions. This was a very successful alternative and compromise considering the COVID virus and inability to meet in person.