TEAM.NAME V2

TEAM_03

REACT-BOOTSTRAP

Deliverable 3

Members:
Alexei COREIBA
Gaganpreet KABERWAL
Harsh PATEL
Kohilan MOHANARAJAN
Kanstantsin ILIOUKEVITCH

Supervisor:
Prantar BHOWMIK
Anya TAFLIOVICH

March 23, 2020

Contents

1 Example Issues

Before we begin to explore different features, it is important to note that the only issue that is marked as hard within `react-bootstrap`'s current issues list is to add a portal for documentation of any new releases. You can find all issues marked as hard at the following link: github-issues. As documentation is not allowed to be completed, we decided to look through the other issues in the list to pick out some that would be relatively difficult and would require a considerable amount of effort for implementation and testing. There's also only one issue that is marked as medium, which is also related directly to documentation. This can be seen here: github-issues. As described in Deliverable 2, most of the issues that are found in the issues list are not tagged and that is probably why there is only one hard and one medium issue that is tagged. The following issues are ones that we've found that would need a more considerable amount of work than the issues that were selected in Deliverable 2, thus classifying them as 'hard' issues. We are hoping that these issues are not caused by other dependencies that React-Bootstrap relies on, as we would not be able to fix them in that case.

1.1 Issue #4911

1.1.1 Description

One component of React-Bootstrap is something called a `<ListGroup>`, which is essentially a container for a list. A sub-element of the `<ListGroup>` is called a `<ListGroup.Item>`. It is possible to have nested `<ListGroup>`s by having a `<ListGroup>` within a `<ListGroup.Item>`. This functionality is important to understand for this issue. Another thing to note is that a `<ListGroup.Item>` can be marked as either `active` or `disabled` which either highlights the list entry - to show current active selection - or tints it to show it's unavailability respectfully.

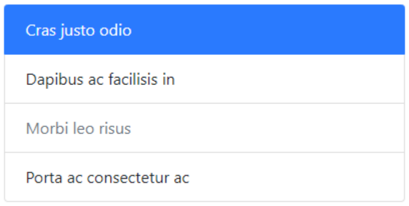


Figure 1: Expected behaviour of having a `<ListGroup.Item>` being set to `active`, `disabled`, or neither

Now the bug is described as follows: When a `<ListGroup.Item>` is set to disabled, and it contains a nested `<ListGroup>` element then it doesn't render correctly. The submitter of the issue attached two images showing this behaviour, these images can be seen down below.

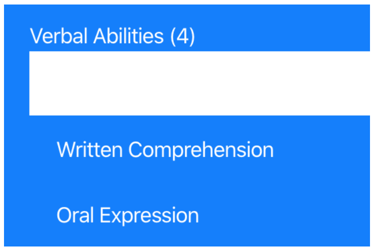


Figure 2: `<ListGroup.Item>` set to `disabled` and thus the inner `<ListGroup>` not rendered

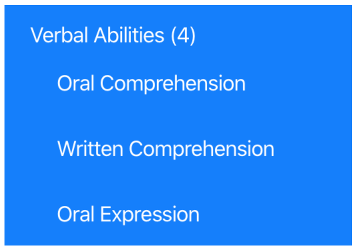


Figure 3: `<ListGroup.Item>` set to `active` and thus the inner `<ListGroup>` is rendered

1.1.2 Interaction Details

The following figure shows a sequence diagram for all the interaction between the different components being used when a ListGroup component is created. The sole purpose of `NexContext.js`, `SelectableCOntext.js`, and `TabContext.js` is to create their respective contexts being the underlying items need to be created. In addition, SelectableContext also provides an event key for AbstractNav, AbstractNavItem and ListGroupItem. ThemeProvider is also used by ListGroup to create a bootstrap prefix which does so using its own ThemeContext. This is not included in the diagram since we will not require any changes in that part of the code to perform our bug fix. All these parent files are imported by `ListGroup.js` and `ListGroupItem.js` and were thus required to be a part of the diagram as we may need to make some changes.

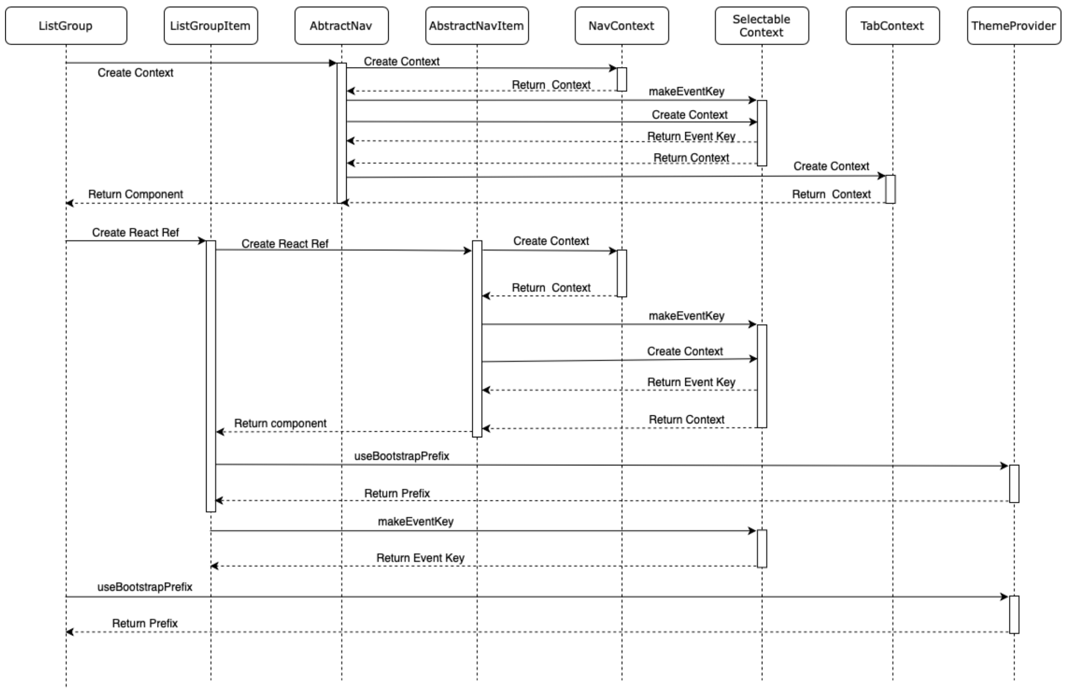


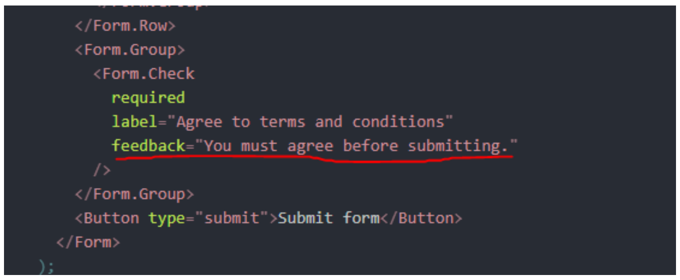
Figure 4: Sequence diagram showing interaction details

1.2 Issue #3939

1.2.1 Description

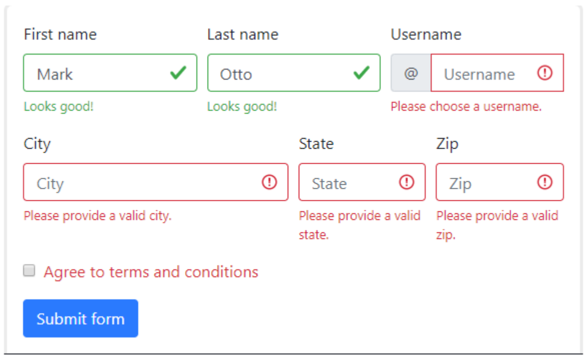
Let us introduce to you the component `<Form>`. This works in a similar fashion as traditional form but allows for Bootstrap styling. Following the description of the component on React-Bootstrap documentation, a basic form is comprised of several `<Form.Group>` components and a `<Button>` to submit the form. The `<Form.Group>` component contains any other components that would be used for input gathering. An example can be that a `<Form.Group>` component contains three other components: `<Form.Label>`, `<Form.Control>`, and `<Form.Text>`. These would set a label to the input field, define the type of the input field, and some text to be shown underneath the input field.

This open issue states that the Checkbox and Radio validation is not working correctly as demonstrated on their own website. When going to the following documentation: native HTML5 form validation at submitting the first form that you can find, we can see that the checkbox says 'Agree to terms and conditions', instead of 'You must agree before submitting' as is shown in the code right below it. This implies that the method that they demonstrate in the code below does not work, as adding a tag `feedback` within the `<Form.Control>` component. A screenshot can be seen below.



```
    </Form.Row>
    <Form.Group>
      <Form.Check
        required
        label="Agree to terms and conditions"
        feedback="You must agree before submitting."
      />
    </Form.Group>
    <Button type="submit">Submit form</Button>
  </Form>
);
```

Figure 5: As can be seen, the checkbox does not state required feedback as set in the screenshot above


First name

Mark ✓

Looks good!

Last name

Otto ✓

Looks good!

Username

@ Username

Please choose a username.

City

City

Please provide a valid city.

State

State

Please provide a valid state.

Zip

Zip

Please provide a valid zip.

Agree to terms and conditions

Figure 6: As can be seen, the checkbox does not state required feedback as set in the screenshot above

1.2.2 Interaction Details

Below is a sequence diagram outlining the Form components of the application. We are assuming that there is an issue with the Feedback interaction and that is what we would investigate further in the next deliverable.

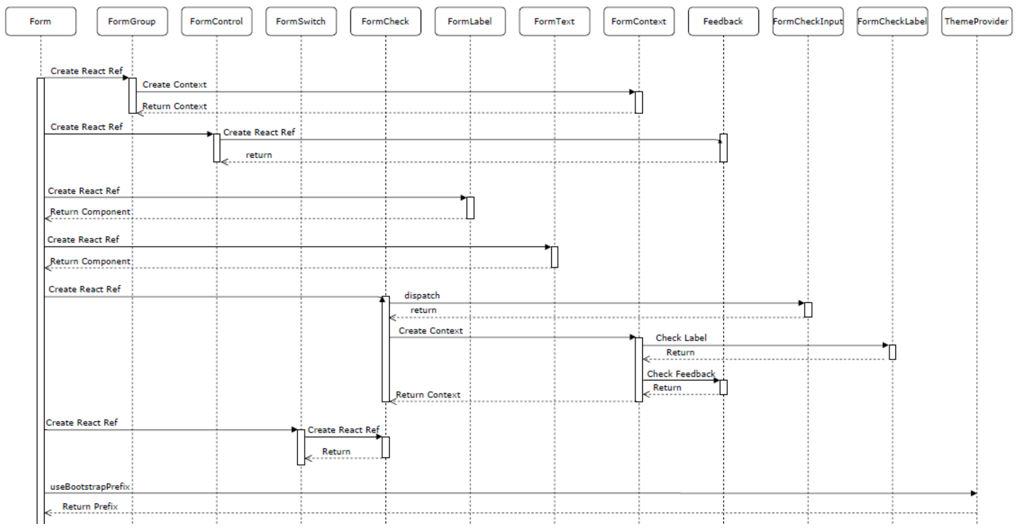


Figure 7: Sequence Diagram for showing interaction details for Form components. Larger image here.

1.3 Issue #4140

1.3.1 Description

A useful component in trying to serve extra content to a reader is to put it into a `<Accordion>` component. Essentially, you click on a button, which expands a hidden section underneath, similarly to an accordion, that can be filled with any other content that you wish. The documentation can be found here: React-Bootstrap Accordion. A functionality that is not supported, which this issue outlines, is the ability to change the styling of the `<Accordion.Toggle>` component dynamically. Meaning that when the toggle is hit to collapse it's respective element, the toggle is not able to be changed stylistically to reflect the toggle action. This issue is not a bug fix, which the other issues could be labeled as, but a feature implementation of an already existing component. An example was posted to the issue which shows the behavior that is currently implemented. Down below, you will see that the Accordion item 1 has been expanded by the Accordion Item 1 Toggle directly above it, but the icon cannot be changed as it is not dynamically updated depending on the action of expanded or not expanded.

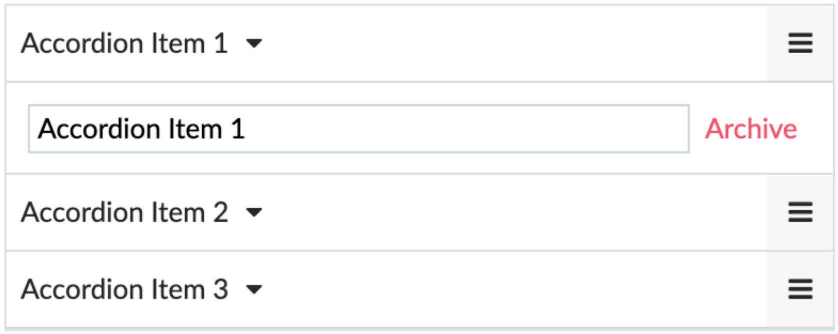


Figure 8: As can be seen, the checkbox does not state required feedback as set in the screenshot above

1.3.2 Interaction Details

The following shows a sequence diagram for the interaction between all the different components pertaining to Accordian and its parent react components. Majority of the interaction is for context creation which will need minimal changes when we are incorporating the bug fixes for the underlying issue. The three major files that will require a deeper dive will be `Accordian.js`, `AccordianToggle.js`, and `AccordianCollapse.js`. It can also be seen that AccordianCollapse uses a Collapse component, which further uses multiple other components. These were not included in the diagram as we do not need to interact with those classes.

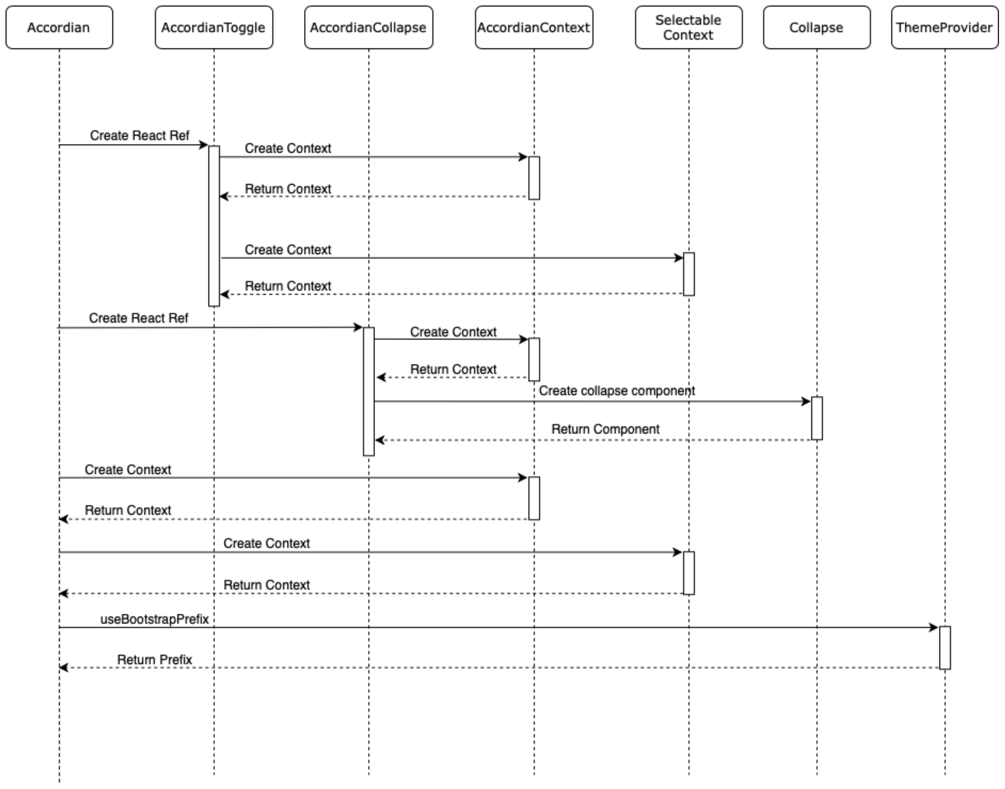


Figure 9: Sequence Diagram for showing interaction details for Accordian components

2 Selected Issue: Issue 4911

We have decided to select Issue 4911.

2.1 Reason for Selection

We opted to choose this as our main issue for the next deliverable, as we felt that it had the least potential for blockers. As we encountered a significant blocker last deliverable, we were more cautious with what we chose for this iteration, and ultimately landed on fixing the `ListGroup` issue.

2.2 Testing

We learned from our previous deliverable that several components of the `react-bootstrap` app use Enzyme, and will continue to work with it for this upcoming deliverable.

We will be able to analyze the current test files and learn from them in order to implement our own testing. Since we have already successfully accomplished adding test cases to existing functions, we should have no problem coming up with our own tests for our purposes.

We will have 1 or 2 people start working on the tests right away. This way we have both the development and testing working side by side to cover more ground and bounce ideas off of each other. Since we do not have much time and the the system is complicated, it would be beneficial to have a variety of perspectives on the functions and problems. By having development work side by side with testing would benefit us greatly.

Once all of the tests have been created, those developers will join the main development group to assists with any final features and bugs that need to be completed.

2.3 Contingency Plan

The biggest uncertainty we have with our chosen issue is the level of difficulty. We've examined the issue and, while we can see that there aren't any visible possible blockers, we struggle with estimating the difficulty of the task at hand. In the occasion that the task itself ends up being easier than expected, we've set forth a contingency plan to do another issue to balance the workload with what's expected of us. For this, we have chosen Issue #3939, as it is the most feasible given the short time frame we have for Deliverable 4.

3 System Architecture

To start off, it is important to remind the reader that we switched projects midway through the last deliverable. This means that this would be our first time evaluating the UML for this specific project, but first we must create a full encompassing UML for this project.

3.1 UML choices and explanation

We decided to include all files and packages from the src folder in our UML. Our reasoning for this was, as mentioned above, this was the first time we were looking at this project's structure and organization due to us switching our open source project. By mapping out the entire layout we were more easily able to identity patterns such as wrappers/adapters, decorators, and composites. We decided to omit any npm libraries or distributions like react as they as this project is an extension of these libraries and it they will be present and used in all files. In the structure we realized that there were a few classes like Themeprovider, ElementChildren, SelectableContext, etc that were used imported by many of the react-bootstrap components. We decided to include this because they provide insights on the use abilities of the component that uses it and shows how the different components are similar/related.

React-Bootstrap is comprised of 27 different components, with the support of 3 utilities. This is almost on par with Bootstrap, and it is safe to say that any components and utilities that are not yet present will not be added in the future - although this may not be the case. Most components are independent of each other, but there are a few exceptions of certain components that build upon other's functionalities. What this means is that, for the most part, most components within React-Bootstrap do not have any dependencies on other components. There are a lot of internal dependencies between sub-components that make up the generalized higher level components. Such an example being the `<Form>` component having dependencies on `<FormText>`, `<FormCheck>`, etc.. whilst still being referred to on the higher level as the generalized `<Forms>` component. There are 109 source files that are used within this project. Each file can be broken down to the following

2 groups. The first group being individual sub-component files, as described above, and the second group being Wrappers - or Adapters - of basic React functionalities to be used by React-Bootstrap components.

3.2 UML

The many following figures show the full scale React-Bootstrap project. It is important to explain what you will be looking at prior to showing them. A React-Bootstrap component can be seen encompassed in a component box with the name of the component present in the box in blue. The boxes within each component are the sub-component files that make up the overarching component. The arrows that you see are all 'use' arrows. Meaning that the sub-component references the other sub-component within it's code. Now there are boxes that represent the wrapped React files that are not within any component. These files are used by many different components. As this would clutter the UML, we've created many instances of these files throughout the UML, and have them colour coded, to make it as clean as we can.

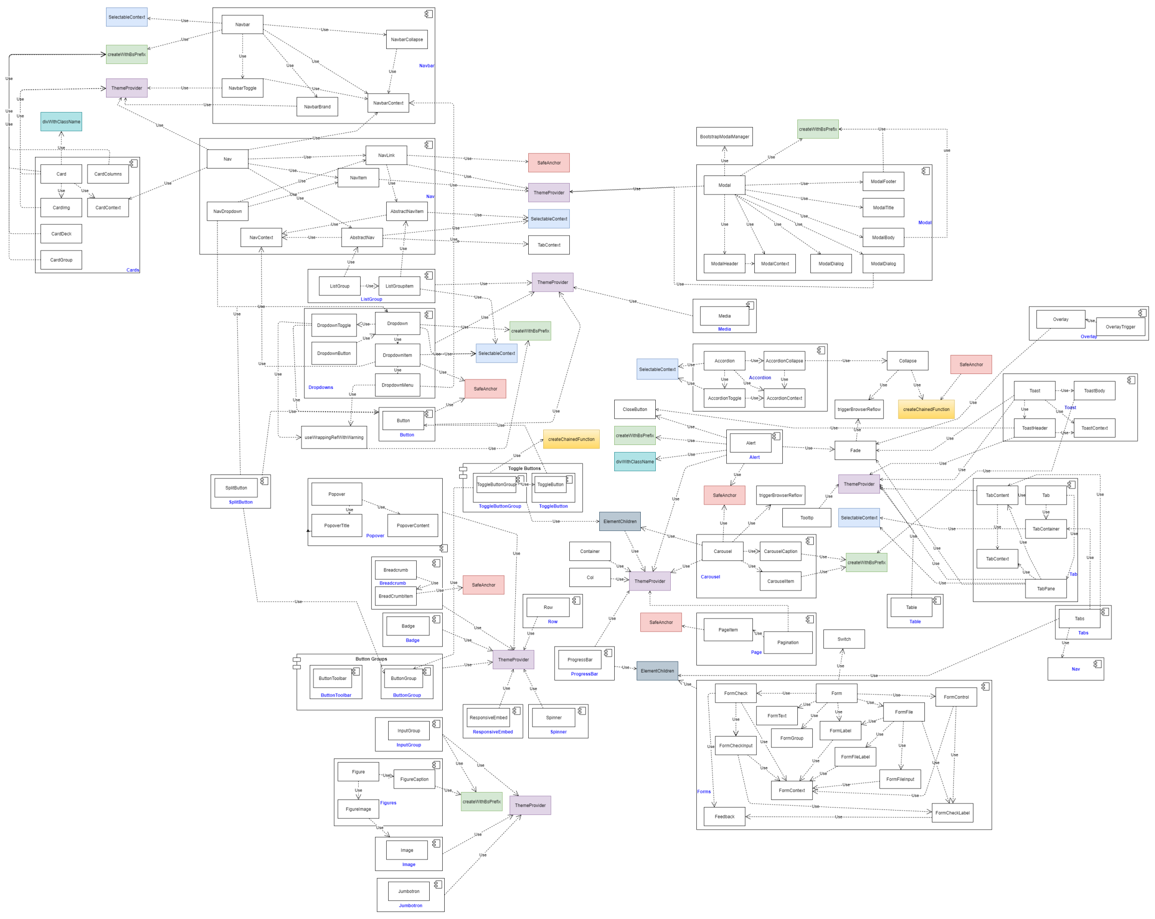


Figure 10: Full UML for React-Bootstrap

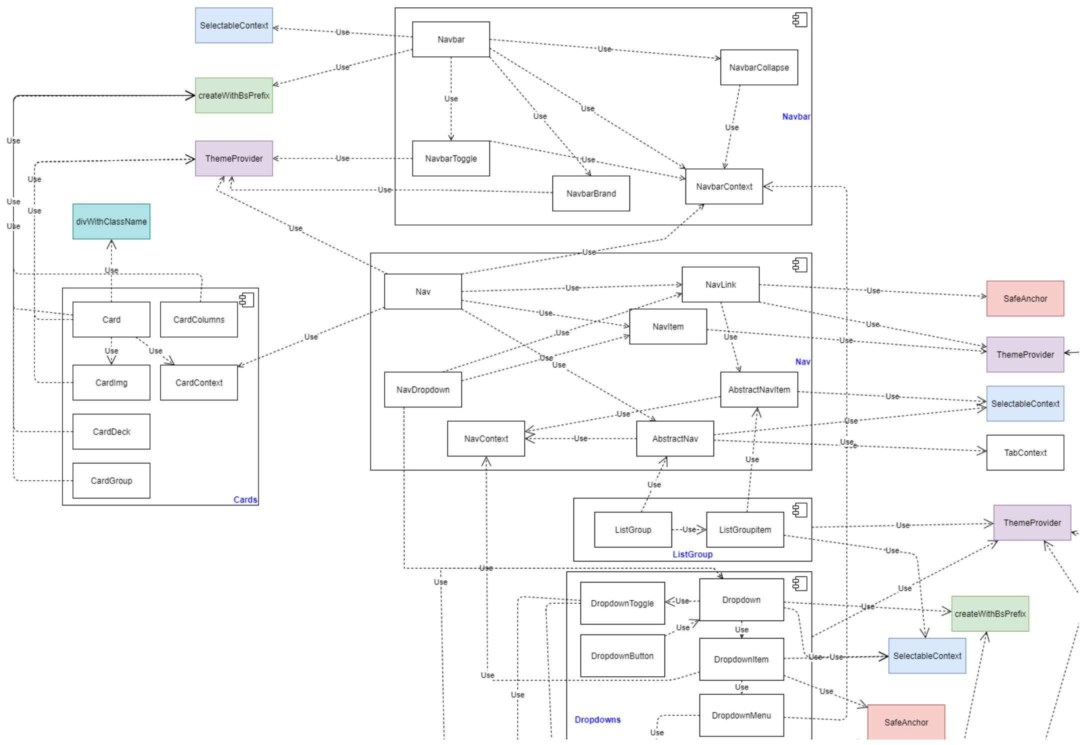


Figure 11: Top left corner of Figure 10

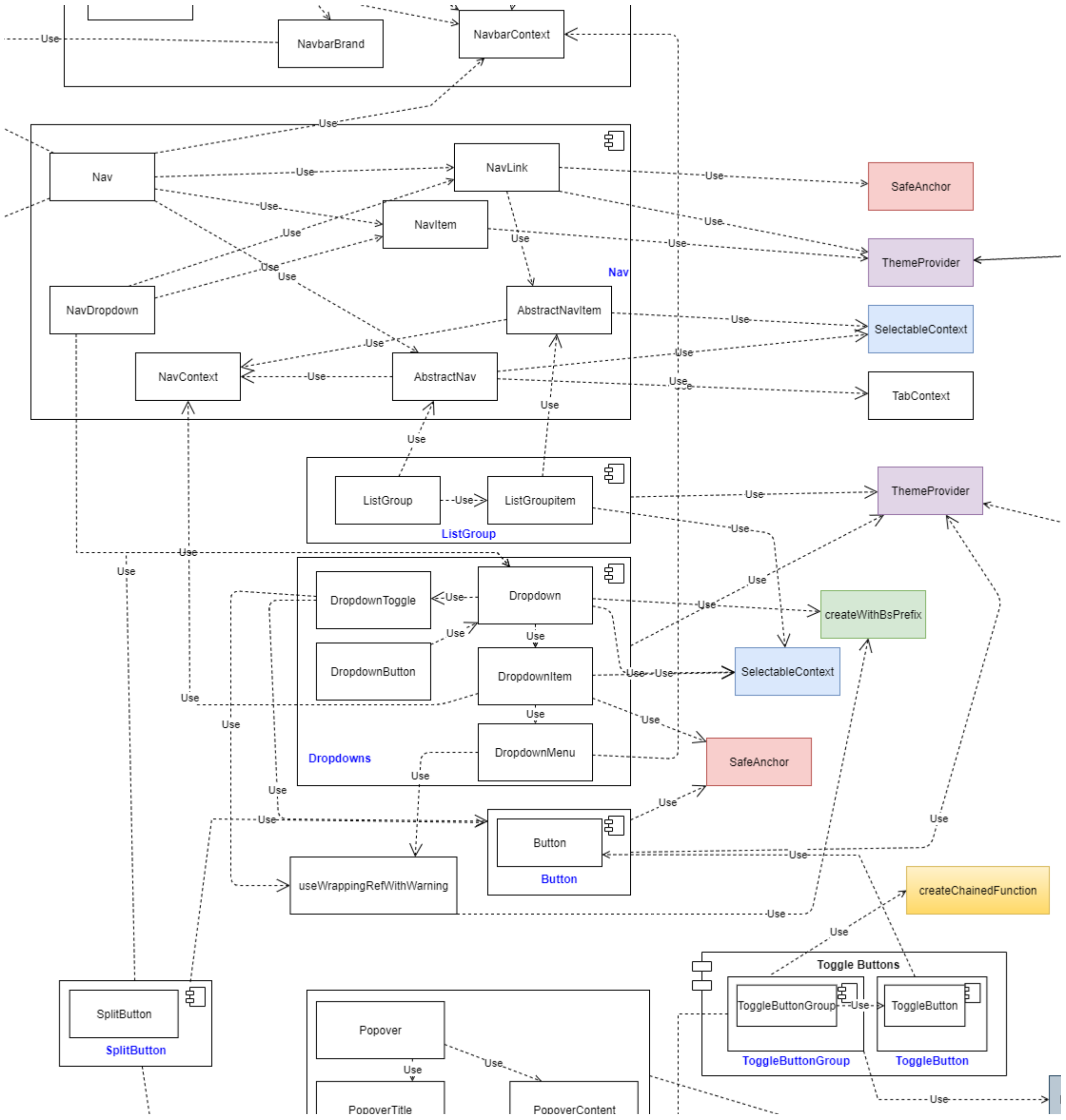


Figure 12: Middle left side of Figure 10

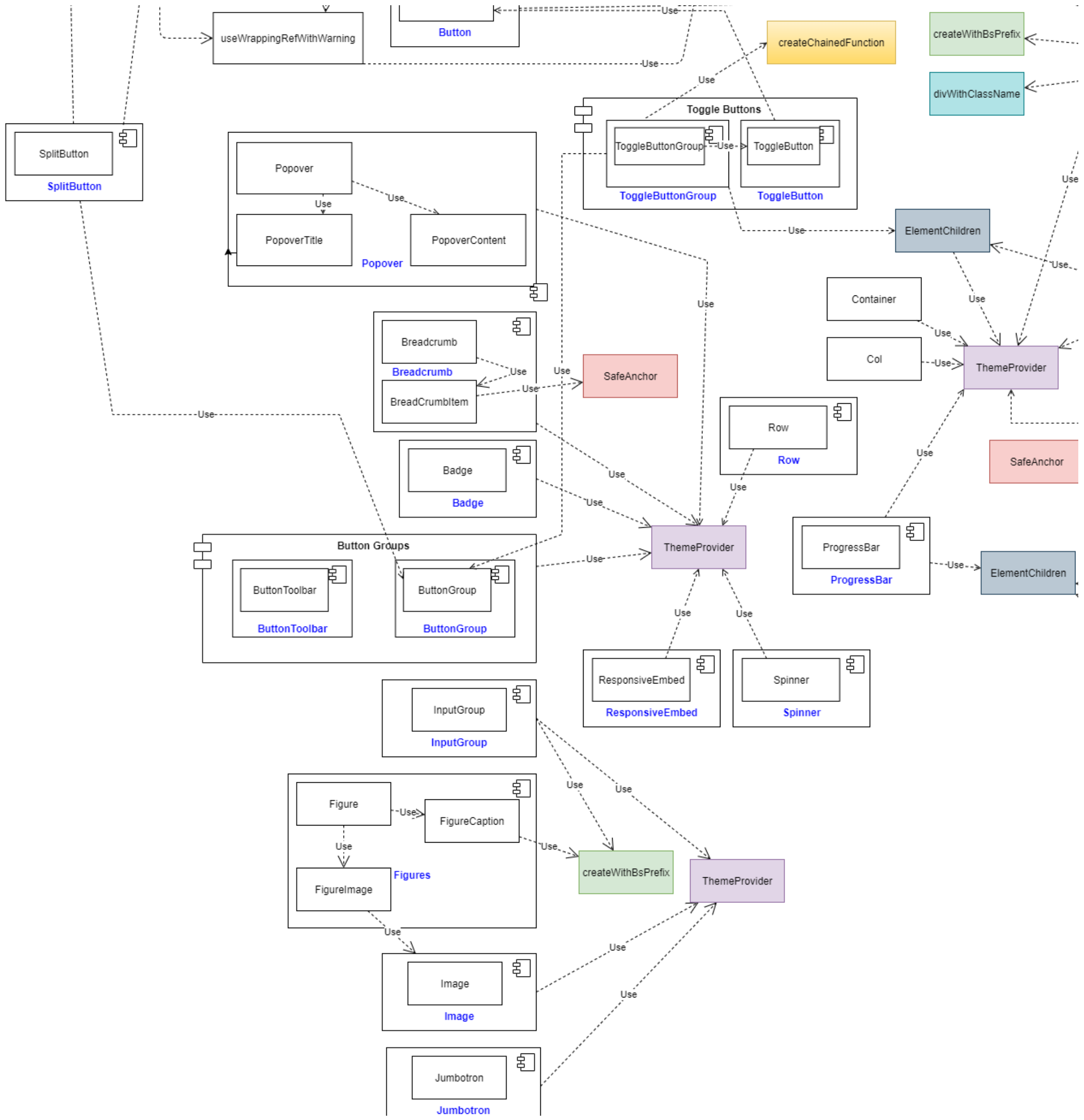


Figure 13: Bottom left corner of Figure 10

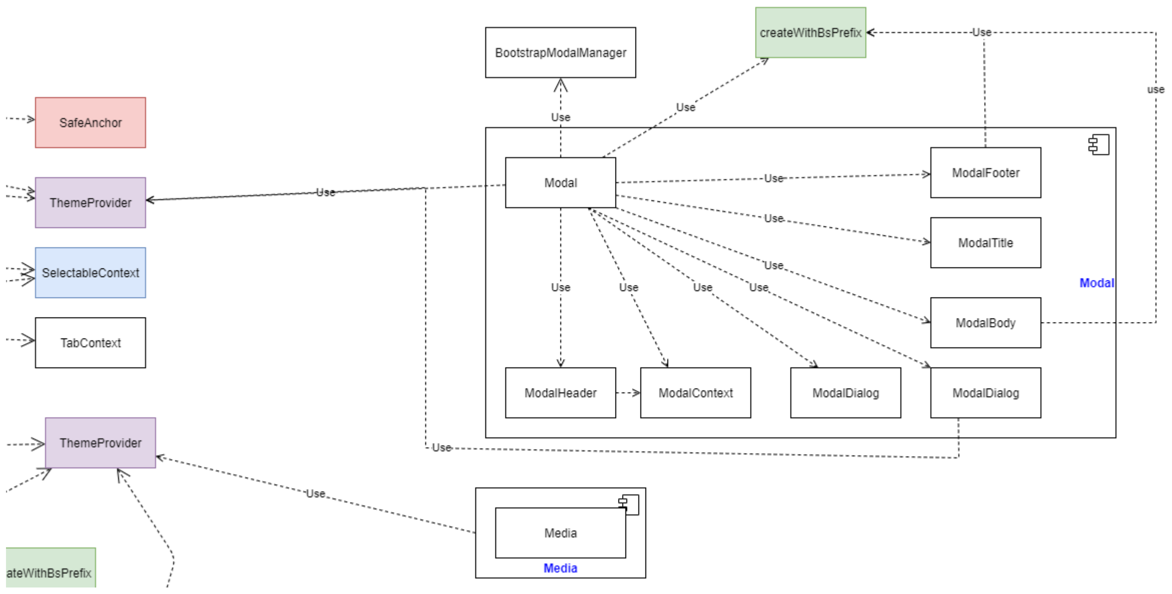


Figure 14: Top right corner of Figure 10

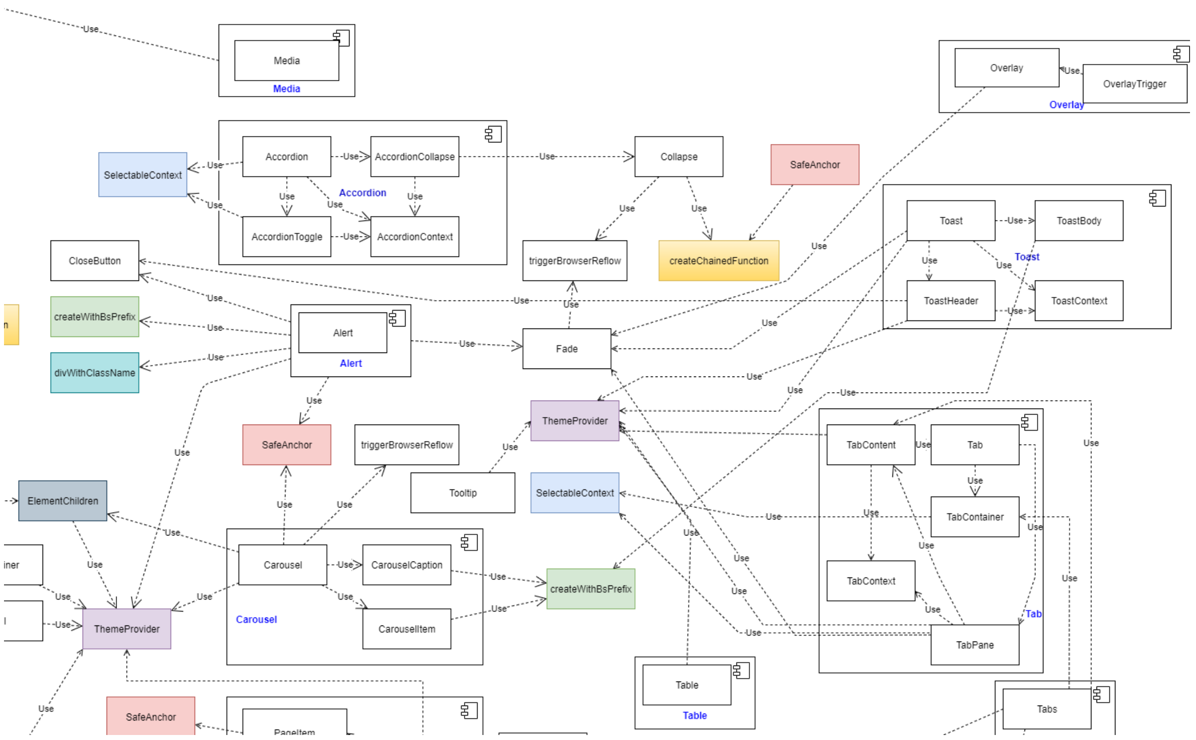


Figure 15: Middle right of Figure 10

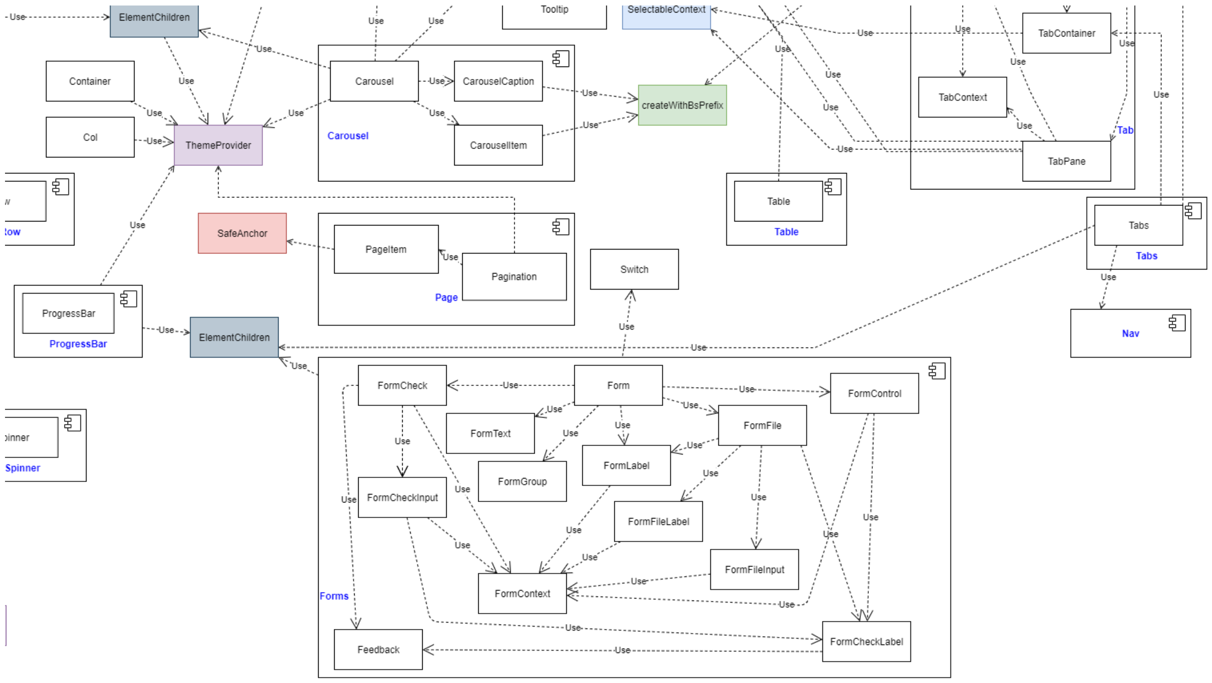


Figure 16: Bottom right corner of Figure 10

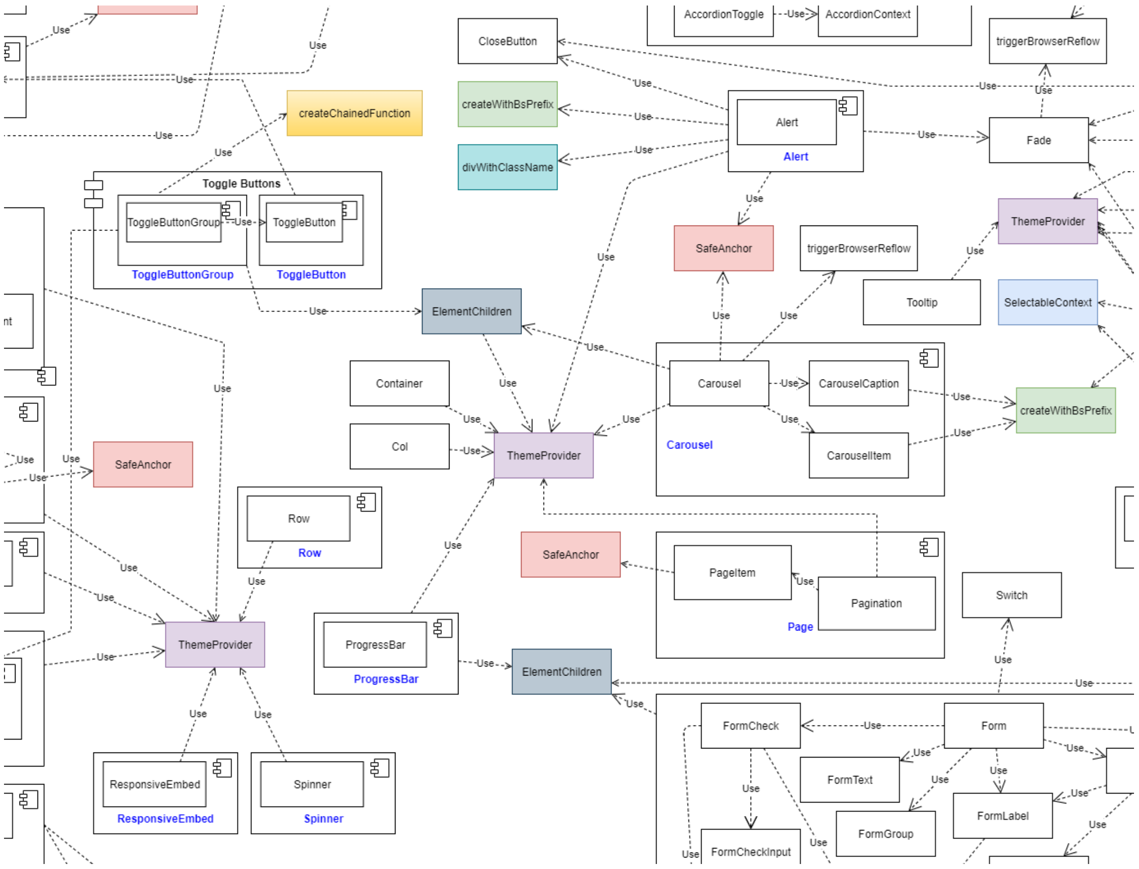


Figure 17: Middle section of Figure 10

3.3 Design Patterns

The overall structure of this project is very simple. There are a couple structural patterns that are used throughout the project, of which we will go over, but you will notice that there isn't any overly complicated design choices that were made through the development of this project. First, we mentioned how there are two groups that each file can be split into. This should be expanded on a little more; keep in mind that the two groups that we mentioned were direct sub-components and wrapped React functionalities.

3.3.1 Wrapper

Whilst this is true, there is more to it than meets the eye. Every file in itself can be considered as a wrapper of some React functionality. Each sub-component relies on some form of React functionality in order to work as expected. Therefore each file in itself can be considered as a wrapper.

3.3.2 Decorator

Although the reason as to why we split it into these two groups is that the sub-components are also decorated for their Bootstrap functionality. This means that the developers definitely use the decorator design pattern. The functionally that is added, or rather taken away is the making of certain props uncontrollable from the outside.

3.3.3 Composite

Lastly, if you notice the overall structure, the developers use the Composite design pattern for sub-component files. If you've ever used React you would know that in order for a `<Form>` component to do it's desired job you would need to include it's dependencies so it works as expected. This slightly follows the composite design pattern as all sub-components must be treated as part of one overarching component.