Bug 1: Select Dropdown Scroll Issue

Issue: Select dropdown doesn't scroll with rest of the page

How to reproduce:

1. Make your viewport smaller in height. Small enough to have a scroll bar
2. Click on the Filter by employee select to open the options dropdown
3. Scroll down the page

Expected Behavior: Options dropdown moves with its parent input as you scroll the page

Actual Behavior: Options dropdown stays in the same position as you scroll the page, losing the reference to the select input

Found Problem: The issue is occurring in the '`/src/index.css`' file `line 95`, which contains the CSS styling of the dropdown container:

```css
.RampInputSelect--dropdown-container {
  display: none;
  position: fixed;
  width: 100%;
  max-width: 700px;
  border: 1px solid var(--color-darker-shade);
  margin-top: 0.5rem;
  max-height: 16rem;
  overflow: auto;
  box-shadow: rgb(0 0 0 / 10%) 0px 0px 1px, rgb(0 0 0 / 13%) 0px 4px 8px;
}
```

The problem stems from the use of `position: fixed` in the CSS. When an element has fixed positioning, it's removed from the normal document flow and positioned relative to the viewport, not its parent element. This causes the dropdown container to maintain its original viewport position when the user scrolls the page, while the parent select element moves with the scroll. As a result, the visual and functional connection between the dropdown and its trigger element is broken.

Solution:
The solution changes the positioning method to maintain the relationship between the dropdown and its parent element during scrolling:

```css
.RampInputSelect--dropdown-container {
  display: none;
  position: absolute;
  width: 100%;
  max-width: 700px;
  border: 1px solid var(--color-darker-shade);
  margin-top: 0.5rem;
  max-height: 16rem;
  overflow: auto;
  box-shadow: rgb(0 0 0 / 10%) 0px 0px 1px, rgb(0 0 0 / 13%) 0px 4px 8px;
  z-index: 100
}
```

1. **Better Positioning Context**: Using `position: absolute` instead of `fixed` positions the dropdown relative to its nearest positioned ancestor (likely the parent select container) rather than the viewport. This maintains proper parent-child relationship during scroll events.
2. **Enhanced User Experience**: The dropdown now behaves according to user expectations, staying attached to its trigger element during scrolling. This creates a more intuitive and consistent interface that aligns with standard web UI patterns.
3. **Improved Stacking Context**: The addition of `z-index: 100` ensures the drop down properly appears above other elements on the page, preventing potential overlap issues with nearby content.
4. **Performance Efficiency**: This CSS-only solution eliminates the need for JavaScript event listeners to reposition the dropdown on scroll events, resulting in better performance compared to script-based workarounds.

Bug 2: Approve Checkbox Issue

Issue: Approve checkbox not working

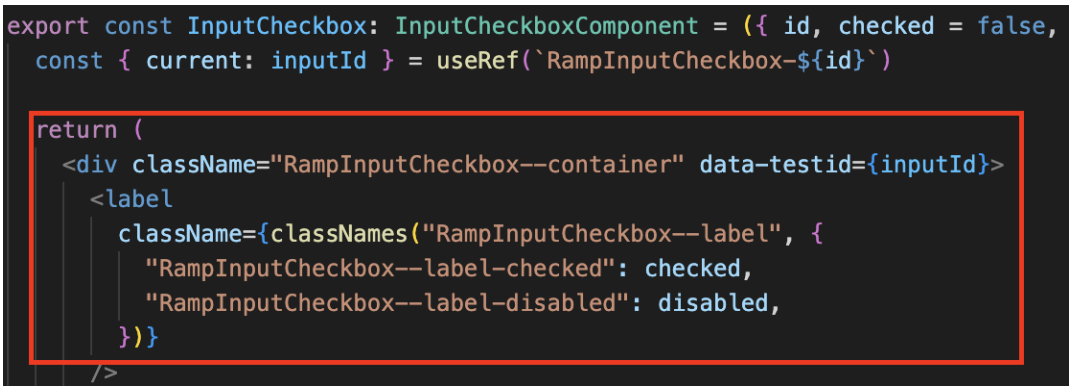
How to reproduce:

1. Click on the checkbox on the right of any transaction

Expected Behavior: Clicking the checkbox toggles its value

Actual Behavior: Nothing happens

Found Problem: The issue is occurring in the '`/src/components/InputCheckbox/index.tsx`' file `lines 8-16`, which contains the behavior of the `InputCheckbox` component:

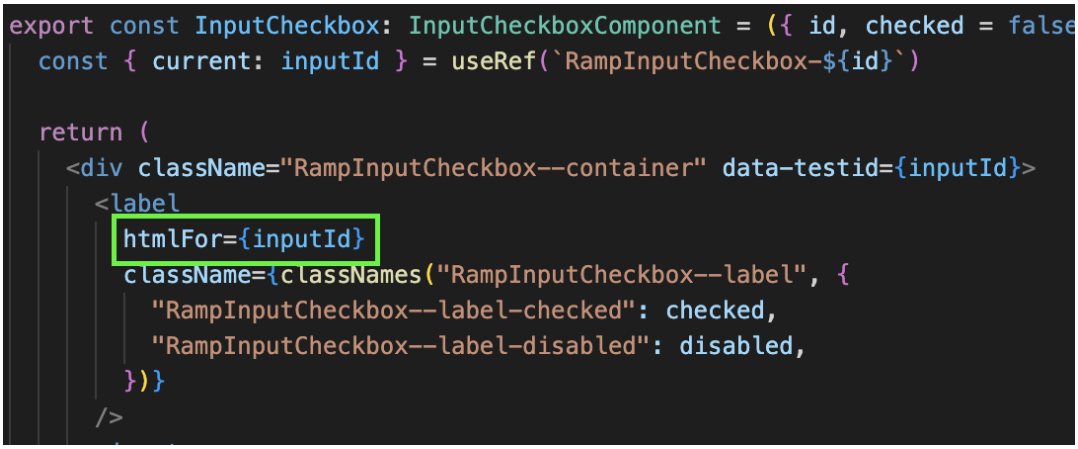


```
export const InputCheckbox: InputCheckboxComponent = ({ id, checked = false,
  const { current: inputId } = useRef(`RampInputCheckbox-${id}`)

  return (
    <div className="RampInputCheckbox--container" data-testid={inputId}>
      <label
        className={classNames("RampInputCheckbox--label", {
          "RampInputCheckbox--label-checked": checked,
          "RampInputCheckbox--label-disabled": disabled,
        })}
      />
```

The label is not connected to the input element because it's missing the `htmlFor` attribute. When using a hidden input and a styled label as a replacement, the label must be linked to the input using the `htmlFor` attribute that matches the input's `id`, otherwise clicking the label won't trigger the input.

Solution:
Add the `htmlFor` attribute to the label element to connect it to the hidden input:



```
export const InputCheckbox: InputCheckboxComponent = ({ id, checked = false
  const { current: inputId } = useRef(`RampInputCheckbox-${id}`)

  return (
    <div className="RampInputCheckbox--container" data-testid={inputId}>
      <label
        htmlFor={inputId}
        className={classNames("RampInputCheckbox--label", {
          "RampInputCheckbox--label-checked": checked,
          "RampInputCheckbox--label-disabled": disabled,
        })}
      />
```

1. **Maintains Design While Fixing Functionality**: This solution preserves the custom visual styling while making the checkbox functional.
2. **Accessibility Improvement**: Proper label association is an accessibility best practice, making the component more usable for all users.
3. **No Additional JavaScript**: This is a clean HTML solution that doesn't require additional event handlers or JavaScript logic.
4. **Industry Standard Approach**: This is the standard way to implement custom-styled form controls - hiding the native element and styling a label that's properly associated with the input.

Bug 3: All Employees Selection Issue

Issue: Cannot select *All Employees* after selecting an employee

How to reproduce:

1. Click on the Filter by employee select to open the options dropdown
2. Select an employee from the list
3. Click on the Filter by employee select to open the options dropdown
4. Select All Employees option

Expected Behavior: All transactions are loaded

Actual Behavior: The page crashes

Found Problem: The issue is occurring in the '`/src/App.tsx`' file `line 35`, which contains the `useTransactionsByEmployee` hook when transitioning from a specific employee back to "All Employees"

```
const loadTransactionsByEmployee = useCallback(
  async (employeeId: string) => {
    paginatedTransactionsUtils.invalidateData()
    await transactionsByEmployeeUtils.fetchById(employeeId)
  },
  [paginatedTransactionsUtils, transactionsByEmployeeUtils]
)
```

When selecting "All Employees" (which has an empty ID), the app attempts to fetch transactions with this empty ID rather than loading all transactions. The fetchById function in `useTransactionsByEmployee.ts` doesn't validate the empty ID before making the API request, causing the server to throw "Employee id cannot be empty" when it should instead be handled as a special case to show all transactions.

Solution:
Modify the `loadTransactionsByEmployee` function to handle the empty employee ID case differently:

```
const loadTransactionsByEmployee = useCallback(
  async (employeeId: string) => {
    paginatedTransactionsUtils.invalidateData()

  if (employeeId === EMPTY_EMPLOYEE.id) {   // Check if the empty employee (All Employees) is selected
    await loadAllTransactions();           // Load all transactions instead of filtering by employee
  } else {                                 // Only fetch by employee ID for actual employees
    await transactionsByEmployeeUtils.fetchById(employeeId);
  }

  },
  [paginatedTransactionsUtils, transactionsByEmployeeUtils, loadAllTransactions]
)
```

1. **Proper Handling of Special Cases**: This solution properly distinguishes between selecting a specific employee and selecting "All Employees".
2. **Prevents API Errors**: By not sending the empty employee ID to the API, we prevent the "Employee id cannot be empty" error.
3. **Consistent Behavior**: The application now behaves consistently, showing all transactions when "All Employees" is selected regardless of previous selections.
4. **Improved User Experience**: Users can freely switch between viewing all transactions and transactions for specific employees without encountering errors.

Bug 4: View More Button Data Issue

Issue: Clicking on View More button not showing correct data

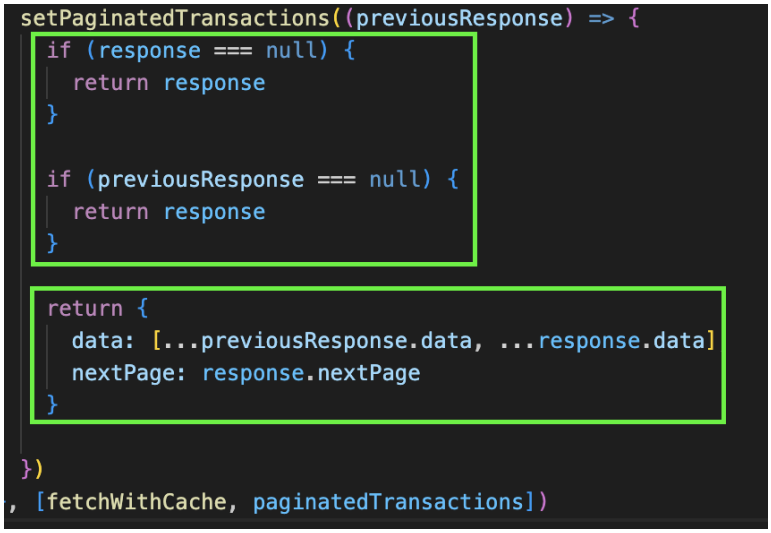
How to reproduce:

1. Click on the View more button
2. Wait until the new data loads

Expected Behavior: Initial transactions plus new transactions are shown on the page

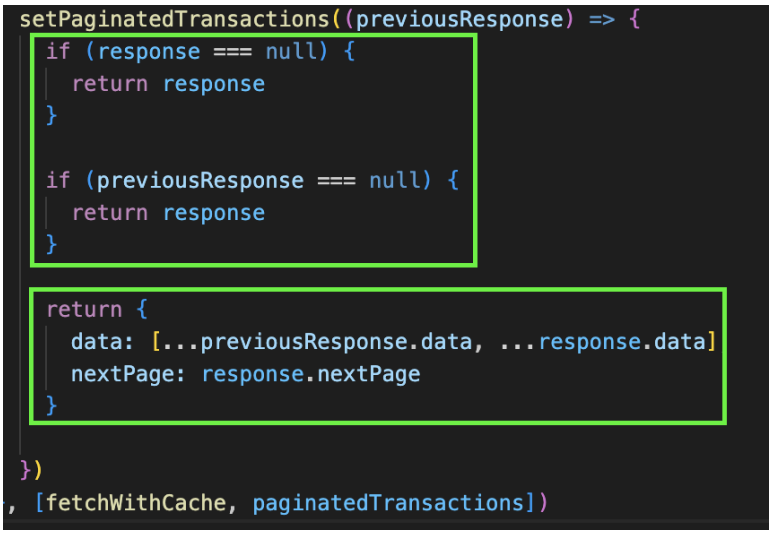
Actual Behavior: New transactions replace initial transactions, losing initial transactions

Found Problem: The issue is occurring in the '`src/hooks/usePaginatedTransactions.ts`' file in `lines 20 - 26`, when setting the new paginated transactions, the function doesn't properly merge the new data with existing data:



This code only takes the `response.data` but doesn't concatenate it with the existing `previousResponse.data`, causing new transactions to replace the old ones instead of appending to them.

Solution:
This change ensures that the new transactions loaded from the "View More" button click are appended to the existing transactions rather than replacing them:



```
setPaginatedTransactions((previousResponse) => {
  if (response === null) {
    return response
  }

  if (previousResponse === null) {
    return response
  }

  return {
    data: [...previousResponse.data, ...response.data]
    nextPage: response.nextPage
  }

})
, [fetchWithCache, paginatedTransactions])
```

1. **Data Persistence**: Users won't lose sight of previous transactions when loading more data, providing a continuous browsing experience.
2. **Improved UX**: The "View More" button now behaves as users expect, showing an expanded list of transactions rather than a completely new set.
3. **Consistent Data Viewing**: Users can now scan through all transactions loaded so far without having to remember what they saw previously.
4. **True Pagination**: This implements proper pagination behavior where new pages of data are appended to existing data, rather than replacing it.

Bug 5: Employee Filter Availability Issue

Issues: Employees filter not available during loading more data

- **How to reproduce** Bug #1:
    1. Open devtools to watch the simulated network requests in the console
    2. Refresh the page
    3. Quickly click on the Filter by employee select to open the options dropdown

    Expected Behavior: The filter should stop showing "Loading employees.." as soon as the request for employees is succeeded

    Actual Behavior: The filter stops showing "Loading employees.." until paginatedTransactions is succeeded

- **How to reproduce** Bug #2:
    1. Open devtools to watch the simulated network requests in the console
    2. Click on View more button
    3. Quickly click on the Filter by employee select to open the options dropdown

    Expected Behavior: The employees filter should not show "Loading employees..." after clicking View more, as employees are already loaded

    Actual Behavior: The employees filter shows "Loading employees..." after clicking View more until new transactions are loaded

Found Problem: The issue is occurring in the '`/src/App.tsx`' file in `lines 22 - 30`, where the `isLoading` state is used incorrectly.

```
const loadAllTransactions = useCallback(async () => {
  setIsLoading(true)
  transactionsByEmployeeUtils.invalidateData()

  await employeeUtils.fetchAll()
  await paginatedTransactionsUtils.fetchAll()

  setIsLoading(false)
}, [employeeUtils, paginatedTransactionsUtils, transactionsByEmployeeUtils])
```

The component is using a single `isLoading` state variable to control loading for both employees and transactions. In the `loadAllTransactions` function, `setIsLoading(true)` is called at the beginning and `setIsLoading(false)` at the end, after awaiting both employees and transactions. This means the employee dropdown shows a loading state even when only transactions are loading

Solution: The solution is to separate the loading states for employees and transactions, or use a more granular approach to determining when the employee dropdown should show its loading state:

```
const [isEmployeesLoading, setIsEmployeesLoading] = useState(false)

const loadAllTransactions = useCallback(async () => {
  if (employees === null) {                    // Only set employee loading if employees aren't loaded yet
    setIsEmployeesLoading(true)
  }

  transactionsByEmployeeUtils.invalidateData()

  if (employees === null) {                    // If employees need to be loaded, do that first
    await employeeUtils.fetchAll()
    setIsEmployeesLoading(false)
  }

  await paginatedTransactionsUtils.fetchAll() // Loading transactions doesn't affect employee dropdown
```

Then to update the InputSelect component to use the correct loading state:

```
<InputSelect<Employee>
  isLoading={isEmployeesLoading}
  defaultValue={EMPTY_EMPLOYEE}
  items={employees === null ? [] : [EMPTY_EMPLOYEE, ...employees]}
  label="Filter by employee"
  loadingLabel="Loading employees"
  parseItem={(item) => ({
    value: item.id,
    label: `${item.firstName} ${item.lastName}`,
  })}
  onChange={async (newValue) => {
    if (newValue === null) {
      return
    }

    await loadTransactionsByEmployee(newValue.id)
  }}
```

1. **Correct Loading Indicators**: The employee dropdown will only show "Loading employees..." when employees are actually being loaded, not during transaction loading.
2. **Enhanced User Experience**: Users can interact with the employee filter as soon as the employee data is available, without waiting for transactions to load.
3. **Improved UI Accuracy**: The interface accurately reflects the actual loading state of each component, making the application more intuitive.
4. **Reduced Confusion**: Users won't be confused by seeing "Loading employees..." when employees are already loaded but transactions are still loading.

Bug 6: View More Button Issue

Issues: View more button not working as expected

- **How to reproduce** Bug #1:
    1. Click on the Filter by employee select to open the options dropdown
    2. Select an employee from the list
    3. Wait until transactions load

    Expected Behavior: The View more button is not visible when transactions are filtered by user, because that is not a paginated request.

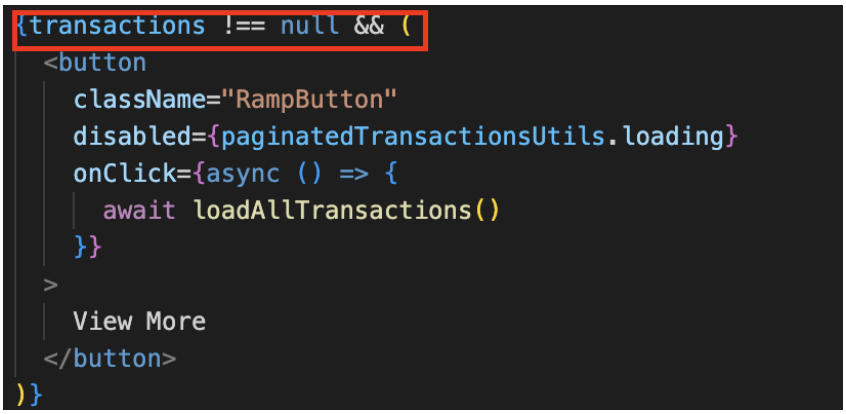
    Actual Behavior: The View more button is visible even when transactions are filtered by employee

- **How to reproduce** Bug #2:
    1. Click on View more button
    2. Wait until it loads more data
    3. Repeat these steps as many times as you can

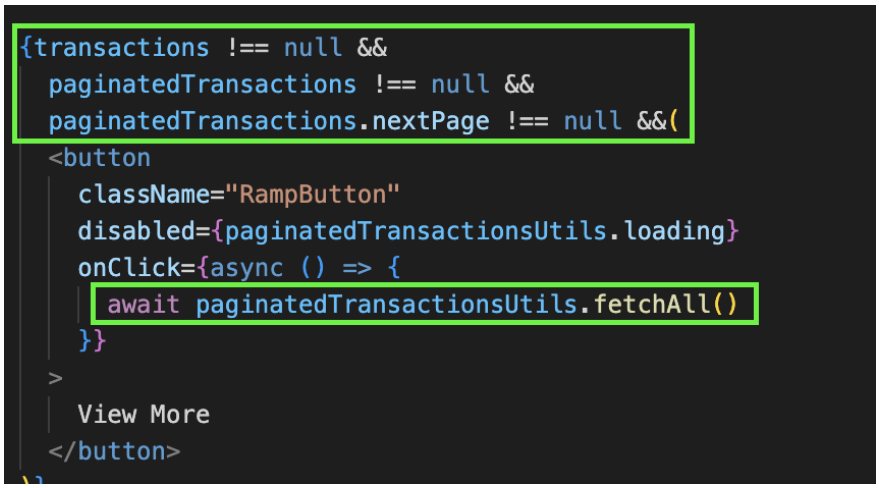
    Expected Behavior: When you reach the end of the data, the View More button disappears and you are not able to request more data.

    Actual Behavior: When you reach the end of the data, the View More button is still showing and you are still able to click the button. If you click it, the page crashes.

Found Problem: The issue is occurring in the '`/src/App.tsx`' file in `lines 89 - 98`, where the `ViewMore` button is conditionally rendered. The button's visibility only checks if transactions exist, not the source of those transactions (paginated vs. filtered) or if there's more data to load. There's also no check to determine if we've reached the end of available data before showing the button.



Solution: The solution is to modify the button to only show when we're viewing paginated transactions (not filtered by employee) and only show when there's more data to load.



```
{transactions !== null &&
  paginatedTransactions !== null &&
  paginatedTransactions.nextPage !== null &&(
  <button
    className="RampButton"
    disabled={paginatedTransactionsUtils.loading}
    onClick={async () => {
      await paginatedTransactionsUtils.fetchAll()
    }}
  >
    View More
  </button>
```

1. **Proper Context-Aware Visibility**: The button only appears when viewing paginated transactions, not when filtering by employee.
2. **Prevents Data Exhaustion Crashes**: The button disappears when there's no more data to load, preventing attempts to fetch non-existent pages.
3. **Improved User Experience**: Users receive appropriate visual feedback about when more data is available to load.
4. **Prevents Unexpected Behavior**: Clicking View More when filtering by employee would previously produce unexpected results; this fix prevents that scenario entirely.
5. **More Efficient Loading**: The button now directly calls `paginatedTransactionsUtils.fetchAll()` rather than `loadAllTransactions()`, which is more appropriate for loading additional pages of data.

Bug 7: Approve Transaction Issue

Issues: Approving a transaction won't persist the new value

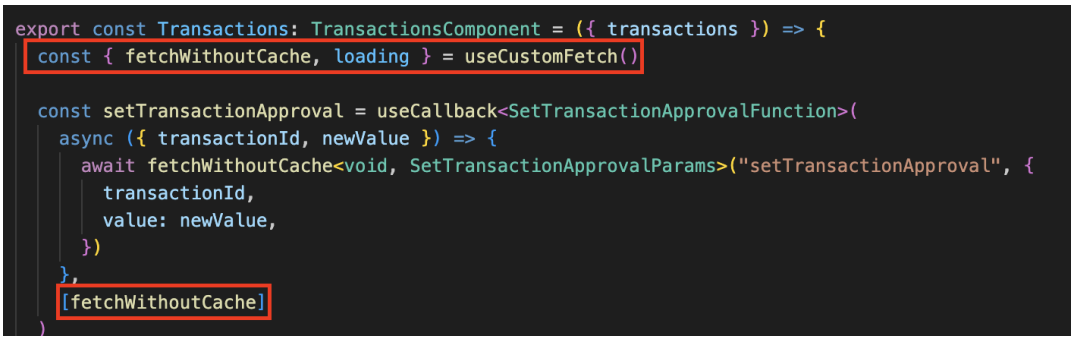
How to reproduce:

1. Click on the Filter by employee select to open the options dropdown
2. Select an employee from the list *(E.g. James Smith)*
3. Toggle the first transaction *(E.g. Uncheck Social Media Ads Inc)*
4. Click on the Filter by employee select to open the options dropdown
5. Select All Employees option
6. Verify values
7. Click on the Filter by employee select to open the options dropdown
8. Verify values

Expected Behavior: In steps 6 and 8, toggled transaction kept the same value it was given in step 2 *(E.g. Social Media Ads Inc is unchecked)*

Actual Behavior: In steps 6 and 8, toggled transaction lost the value given in step 2. *(E.g. Social Media Ads Inc is checked again)*

Found Problem: The issue occurs because transaction approval changes aren't properly synchronized across different data sources in the application. When a user approves/disapproves a transaction, the change is sent to the server via the `setTransactionApproval` API call located within the '`/src/components/Transactions/index.tsx`' file `lines 7 - 18`. However the cached data for "paginatedTransactions" and "transactionsByEmployee" endpoints isn't updated. When switching between views, the application fetches data from these cached endpoints containing stale approval states, causing the user's approval changes to appear to be lost.



```
export const Transactions: TransactionsComponent = ({ transactions }) => {
  const { fetchWithoutCache, loading } = useCustomFetch()

  const setTransactionApproval = useCallback<SetTransactionApprovalFunction>(
    async ({ transactionId, newValue }) => {
      await fetchWithoutCache<void, SetTransactionApprovalParams>("setTransactionApproval", {
        transactionId,
        value: newValue,
      })
    },
    [fetchWithoutCache]
  )
```

Solution: We need to modify the `setTransactionApproval` function in the `Transactions` component to also update all relevant caches:

```tsx
export const Transactions: TransactionsComponent = ({ transactions }) => {
  const { fetchWithoutCache, clearCacheByEndpoint, loading } = useCustomFetch()

  const setTransactionApproval = useCallback<SetTransactionApprovalFunction>(
    async ({ transactionId, newValue }) => {
      await fetchWithoutCache<void, SetTransactionApprovalParams>("setTransactionApproval", {
        transactionId,
        value: newValue,
      })

      clearCacheByEndpoint(["paginatedTransactions", "transactionsByEmployee"])

    },
    [fetchWithoutCache, clearCacheByEndpoint]
  )
```

1. **Consistent Data**: Transaction approval states will be consistent across all views of the application.
2. **Seamless User Experience**: Users can toggle approvals and switch between views without losing their changes.
3. **Single Source of Truth**: The server becomes the single source of truth for transaction approval states.
4. **Proper Cache Management**: By properly invalidating caches after mutations, we ensure that all parts of the application see the most up-to-date data.