Popper

A Popper can be used to display some content on top of another. It's an alternative to react-popper.

Some important features of the Popper component:

- 🕷 Popper relies on the 3rd party library (Popper.js) for perfect positioning.
- 💄 It's an alternative API to react-popper. It aims for simplicity.
- Its child element is a Portal on the body of the document to avoid rendering problems. You can disable this behavior with `disablePortal`.
- The scroll isn't blocked like with the Popover component. The placement of the popper updates with the available area in the viewport.
- Clicking away does not hide the Popper component. If you need this behavior, you can use the Click-Away Listener - see the example in the menu documentation section.
- The `anchorEl` is passed as the reference object to create a new `Popper.js` instance.

View as Markdown Feedback Bundle size Source

Basic Popper

Toggle Popper

Edit in Chat JS TS Collapse code

```tsx
import * as React from 'react';
import Box from '@mui/material/Box';
import Popper from '@mui/material/Popper';

export default function SimplePopper() {
  const [anchorEl, setAnchorEl] = React.useState<null | HTMLElement>(null);

  const handleClick = (event: React.MouseEvent<HTMLElement>) => {
    setAnchorEl(anchorEl ? null : event.currentTarget);
```

```
  };

  const open = Boolean(anchorEl);
  const id = open ? 'simple-popper' : undefined;

  return (
    <div>
      <button aria-describedby={id} type="button" onClick={handleClick}>
        Toggle Popper
      </button>
      <Popper id={id} open={open} anchorEl={anchorEl}>
        <Box sx={{ border: 1, p: 1, bgcolor: 'background.paper' }}>
          The content of the Popper.
        </Box>
      </Popper>
    </div>
  );
}
```

Transitions

The open/close state of the popper can be animated with a render prop child and a transition component. This component should respect the following conditions:

- Be a direct child descendent of the popper.
- Call the `onEnter` callback prop when the enter transition starts.
- Call the `onExited` callback prop when the exit transition is completed. These two callbacks allow the popper to unmount the child content when closed and fully transitioned.

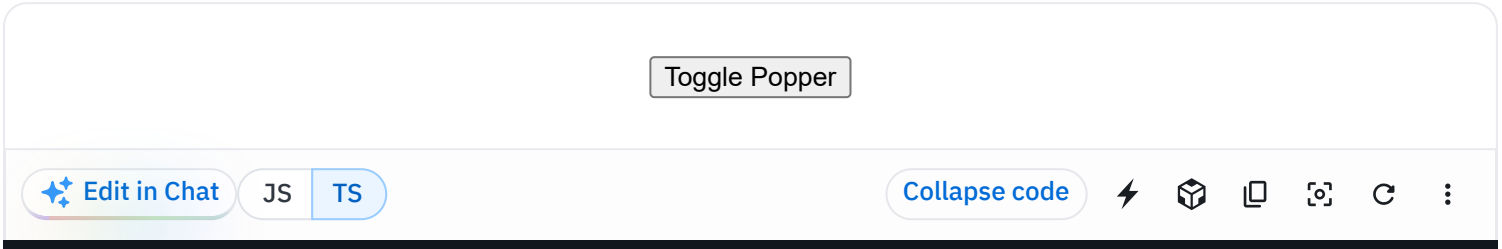
Popper has built-in support for react-transition-group.

Edit in Chat JS TS Collapse code

```
import * as React from 'react';
import Box from '@mui/material/Box';
import Popper from '@mui/material/Popper';
import Fade from '@mui/material/Fade';

export default function TransitionsPopper() {
  const [open, setOpen] = React.useState(false);
  const [anchorEl, setAnchorEl] = React.useState<null | HTMLElement>(null);

  const handleClick = (event: React.MouseEvent<HTMLElement>) => {
    setAnchorEl(event.currentTarget);
    setOpen((previousOpen) => !previousOpen);
```

```
  };

  const canBeOpen = open && Boolean(anchorEl);
  const id = canBeOpen ? 'transition-popper' : undefined;

  return (
    <div>
      <button aria-describedby={id} type="button" onClick={handleClick}>
        Toggle Popper
      </button>
      <Popper id={id} open={open} anchorEl={anchorEl} transition>
        {({ TransitionProps }) => (
          <Fade {...TransitionProps} timeout={350}>
            <Box sx={{ border: 1, p: 1, bgcolor: 'background.paper' }}>
              The content of the Popper.
            </Box>
          </Fade>
        )}
      </Popper>
    </div>
  );
}
```

Alternatively, you can use react-spring.



Edit in Chat JS TS Collapse code

```
import * as React from 'react';
import Box from '@mui/material/Box';
import Popper from '@mui/material/Popper';
import { useSpring, animated } from '@react-spring/web';

interface FadeProps {
  children?: React.ReactElement<unknown>;
  in?: boolean;
  onEnter?: () => void;
  onExited?: () => void;
}

const Fade = React.forwardRef<HTMLDivElement, FadeProps>(function Fade(props, ref) {
  const { in: open, children, onEnter, onExited, ...other } = props;
  const style = useSpring({
    from: { opacity: 0 },
    to: { opacity: open ? 1 : 0 },
    onStart: () => {
      if (open && onEnter) {
        onEnter();
      }
```

```
    },
    onRest: () => {
      if (!open && onExited) {
        onExited();
      }
    },
  });

  return (
    <animated.div ref={ref} style={style} {...other}>
      {children}
    </animated.div>
  );
});

export default function SpringPopper() {
  const [open, setOpen] = React.useState(false);
```

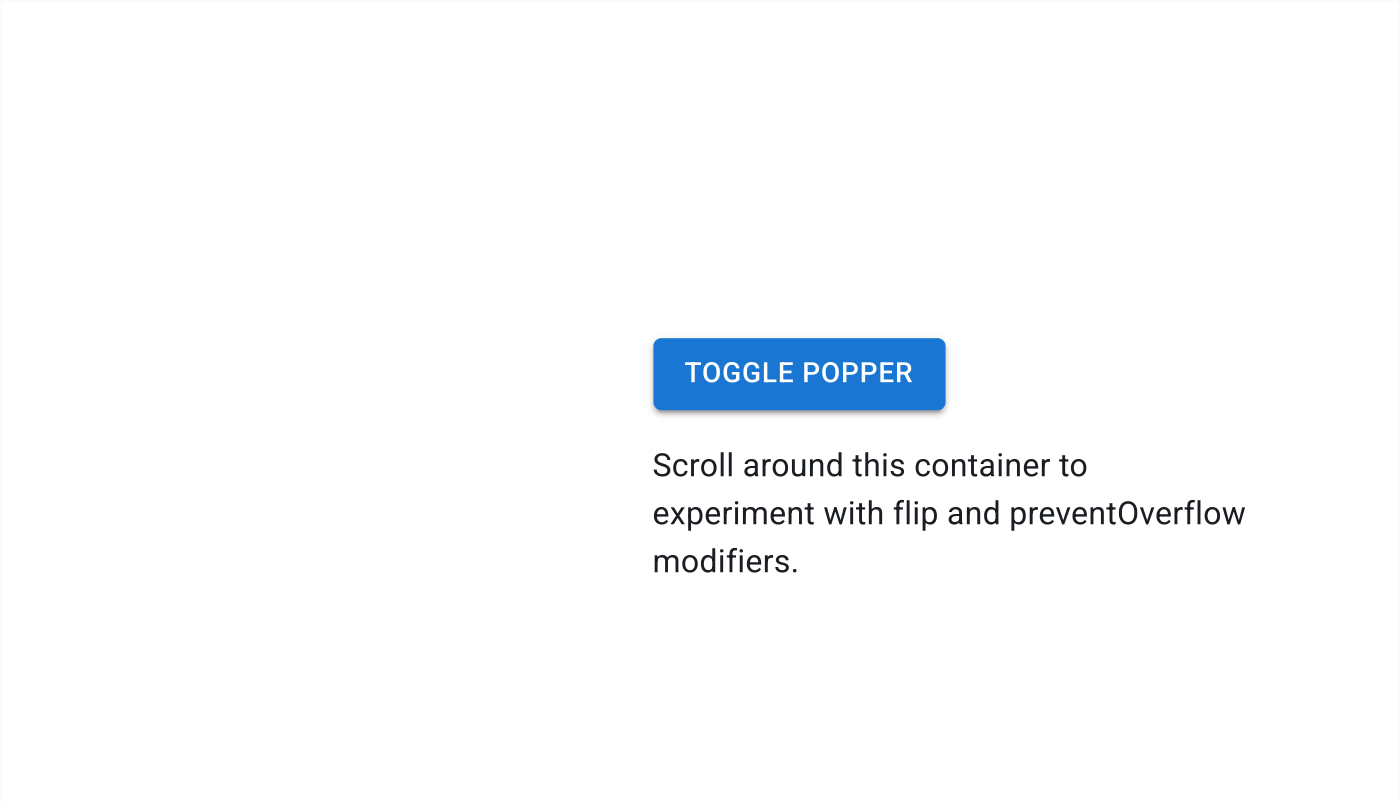
Positioned popper

Edit in Chat | JS | TS | Hide code

```
import * as React from 'react';
import Box from '@mui/material/Box';
import Popper, { PopperPlacementType } from '@mui/material/Popper';
import Typography from '@mui/material/Typography';
import Grid from '@mui/material/Grid';
import Button from '@mui/material/Button';
import Fade from '@mui/material/Fade';
import Paper from '@mui/material/Paper';

export default function PositionedPopper() {
  const [anchorEl, setAnchorEl] = React.useState<HTMLButtonElement | null>(null);
  const [open, setOpen] = React.useState(false);
  const [placement, setPlacement] = React.useState<PopperPlacementType>();

  const handleClick =
    (newPlacement: PopperPlacementType) =>
    (event: React.MouseEvent<HTMLButtonElement>) => {
      setAnchorEl(event.currentTarget);
      setOpen((prev) => placement !== newPlacement || !prev);
```

```
    setPlacement(newPlacement);
  };

return (
  <Box sx={{ width: 500 }}>
    <Popper
      // Note: The following zIndex style is specifically for documentation purposes and may not be
      sx={{ zIndex: 1200 }}
      open={open}
      anchorEl={anchorEl}
      placement={placement}
      transition
    >
      {({ TransitionProps }) => (
        <Fade {...TransitionProps} timeout={350}>
          <Paper>
            <Typography sx={{ p: 2 }}>The content of the Popper.</Typography>
          </Paper>
        </Fade>
```

Scroll playground



Appearance

Placement

bottom

Disable portal

(the children stay within their parent DOM hierarchy)

Modifiers (options from Popper.js)

Prevent Overflow

Enable

Alt axis

Alt Boundary

Tether

Root Boundary

document

Flip

Enable

Alt Boundary

Root Boundary

document

Arrow

Enable

```
<Popper
  placement="bottom"
  disablePortal={false}
  modifiers={[
    {
      name: 'flip',
      enabled: true,
      options: {
        altBoundary: true,
        rootBoundary: 'document',
        padding: 8,
      },
    },
    {
      name: 'preventOverflow',
      enabled: true,
      options: {
        altAxis: true,
        altBoundary: true,
        tether: true
```

Copy

Virtual element

The value of the `anchorEl` prop can be a reference to a fake DOM element. You need to create an object shaped like the `VirtualElement`.

Highlight part of the text to see the popper:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ipsum purus, bibendum sit amet vulputate eget, porta semper ligula. Donec bibendum vulputate erat, ac fringilla mi finibus nec. Donec ac dolor sed dolor porttitor blandit vel vel purus. Fusce vel malesuada ligula. Nam quis vehicula ante, eu finibus est. Proin ullamcorper fermentum orci, quis finibus massa. Nunc lobortis, massa ut rutrum ultrices, metus metus finibus ex, sit amet facilisis neque enim sed neque. Quisque accumsan metus vel maximus consequat. Suspendisse lacinia tellus a libero volutpat maximus.

Edit in Chat | JS | TS | Hide code

```tsx
import * as React from 'react';
import Popper, { PopperProps } from '@mui/material/Popper';
import Typography from '@mui/material/Typography';
import Fade from '@mui/material/Fade';
import Paper from '@mui/material/Paper';

export default function VirtualElementPopper() {
  const [open, setOpen] = React.useState(false);
  const [anchorEl, setAnchorEl] = React.useState<PopperProps['anchorEl']>(null);

  const previousAnchorElPosition = React.useRef<DOMRect>(undefined);

  React.useEffect(() => {
    if (anchorEl) {
      if (typeof anchorEl === 'object') {
        previousAnchorElPosition.current = anchorEl.getBoundingClientRect();
      } else {
        previousAnchorElPosition.current = anchorEl().getBoundingClientRect();
      }
    }
  }, [anchorEl]);

  const handleClose = () => {
    setOpen(false);
  };

  const handleMouseUp = () => {
    const selection = window.getSelection();

    // Resets when the selection has a length of 0
    if (!selection || selection.anchorOffset === selection.focusOffset) {
      handleClose();
      return;
    }

    const getBoundingClientRect = () => {
      if (selection.rangeCount === 0 && previousAnchorElPosition.current) {
        setOpen(false);
```

Supplementary projects

For more advanced use cases you might be able to take advantage of:

material-ui-popup-state

The package `material-ui-popup-state` that takes care of popper state for you in most cases.

Edit in Chat JS TS Hide code

```tsx
import Typography from '@mui/material/Typography';
import Button from '@mui/material/Button';
import Popper from '@mui/material/Popper';
import PopupState, { bindToggle, bindPopper } from 'material-ui-popup-state';
import Fade from '@mui/material/Fade';
import Paper from '@mui/material/Paper';

export default function PopperPopupState() {
  return (
    <PopupState variant="popper" popupId="demo-popup-popper">
      {(popupState) => (
        <div>
          <Button variant="contained" {...bindToggle(popupState)}>
            Toggle Popper
          </Button>
          <Popper {...bindPopper(popupState)} transition>
            {({ TransitionProps }) => (
              <Fade {...TransitionProps} timeout={350}>
                <Paper>
                  <Typography sx={{ p: 2 }}>The content of the Popper.</Typography>
                </Paper>
              </Fade>
            )}
          </Popper>
        </div>
      )}
    </PopupState>
  );
}
```

API

See the documentation below for a complete reference to all of the props and classes available to the components mentioned here.

- `<Popper />`

Popover

Portal