Creating a Testimonials Block II.

My name is Belinda and I've been studying JS for about a year and a half, so although I'm by no means an expert, I wanted to share with you what I've learnt about building Gutenberg blocks!..

📚👩 Qucik Recap of part I 👩‍🎓👨🏿‍🎓

This is my second talk in the series, so I'm going to quickly recap what we learnt in the first talk

```
import './editor.scss';
import './style.scss';

const { __ } = wp.i18n;
const { registerBlockType } = wp.blocks;

export default registerBlockType(
    'namespace/blockname',
    {
        title: __('Block Name', '_vt'),
        description: __('Description of the block', '_vt'),
        category: 'common',
        icon: dashicon || svgIcon,
        keywords: [
            __( 'Key Word 1', '_vt' ),
            __( 'Key Word 2', '_vt' ),
            __( 'Key Word 3', '_vt' ),
        ],
        edit: props => {
            const { className } = props;
            return (
                <div className={ className }>Block Back End </div>
            );
        },
        save: props => {
            const { className } = props;
            return (
                <div className={ className }>Block Front End</div>
            );
        },
    },
);
```

We looked at how to import styles, internationalisation and the register block component. We learnt about namespacing and allowed characters for block names. We looked at the title attribute, description, block categories, icons & keywords. We looked at how the edit object displays the content in the back end, and how the save object is what is saved to the database.

```
edit: props => {
    const { className, isSelected } = props;
    return (
        <div className={ className }>
            <div className="image">
                <img src="/wp-content/plugins/vt_testimonials/assets/images/David-Brent.jpg" />
            </div>
            <div className="info">
                <h2>David Brent</h2>
                <h3>General Manager</h3>
                <div className="text">
                    <p>People see me, and they see the suit, and they go: 'you're not
                    fooling anyone', they know I'm rock and roll through and through.</p>
                    <p>But you know that old thing, live fast, die young? Not my way.
                    Live fast, sure, live too bloody fast sometimes, but die young? Die old.</p>
                </div>
                <a href="https://www.wernham-hogg-limited.com" className="website">Wernham Hogg</a>
            </div>
            {
                isSelected && (
                    <div className="warn"><p>Sorry you can't edit this block</p></div>
                )
            }
        </div>
    );
},
```

We looked at props and what they are, we learnt about JSX and how to use it in the Gutenberg Component, and we looked at the isSelected prop. We also looked at deconstructed variables

Adding editable Text Block

Now are going to move onto how to make a block editable. [Show Static & Editable Block.]

```
import './editor.scss';
import './style.scss';
import icon from './icon';

const { __ } = wp.i18n;
const { registerBlockType } = wp.blocks;
const { RichText } = wp.editor;

export default registerBlockType(
    'testimonials/title',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
            title: {
                type: 'array',
                source: 'children',
                selector: 'h2',
            },
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

Attributes https://github.com/verytwisty/vt*gutenberg*testimonials/blob/master/blocks/2-edit-text-block/index.js

- Call in new component 'RichText' from wp.editor library
- Add in new object called attributes
- Attributes is where the data is stored while the user is editing, before it is saved to the DB
- This is also where existing data is stored that is pulled out of the DB while the user is editing a previously made block
- We can give our attributes any name we wish, we are changing the title, so are going to name the attribute title
- Pink unicorn would also work, but wouldn't make sense for other developers
- Our title attribute has three properties: type, source, children

```
attributes: {
    attribute: {
        type:      'array' || 'string' || 'number',
        source:    'children' || 'attribute' || 'text',
        attribute: 'href' || 'alt' || 'src' || 'any html attribute',
        selector:  'tag' || '.className' || '#ID',
        default:   'arbitary string',
    },
}
```

- Lets have a look at the type of properties an attribute can have
- Type this is the type of info that it will recieve . Can take an array (for paragraph and h tags because the text is an array of children inside the tag). Can also take string, number
- Source can take children, attribute or text. Children is used for p and h tags, attributes for things like images and links, and text for input fields
- If your source is set to attribute, you specify which attribute here... for a link it's the href, for an image it's src etc. We don't need this for paragraph tags
- selector can be a HTML tag, a className or an ID, this is important as this is how Gutenberg will know where to re-import the attributes from the database once the page has loaded
- default. If you want a default value on page load, then add it in here.

```
edit: props => {
    const { className, isSelected } = props;
    return (
        <div className={ className }>
            <div className="image">
                <img src="/wp-content/plugins/vt_testimonials/assets/images/David-Brent.jpg" />
            </div>
            <div className="info">
                <h2>David Brent</h2>
                <h3>General Manager</h3>
                <div className="text">
                    <p>People see me, and they see the suit, and they go: 'you're not
                    fooling anyone', they know I'm rock and roll through and through.</p>
                    <p>But you know that old thing, live fast, die young? Not my way.
                    Live fast, sure, live too bloody fast sometimes, but die young? Die old.</p>
                </div>
                <a href="https://www.wernham-hogg-limited.com" className="website">Wernham Hogg</a>
            </div>
        </div>
    );
},
```

Making the title editable https://github.com/verytwisty/vt*gutenberg*testimonials/blob/master/blocks/2-edit-text-block/index.js

So we want to change this line of Static text into editable text - How?

```
edit: props => {
    const { className, isSelected, attributes: { title } } = props;
    const onChangeHeader = title => { setAttributes( { title } ) };
    return (
        <div className={ className }>
            <div className="image">
                <img src="/wp-content/plugins/vt_testimonials/assets/images/David-Brent.jpg" />
            </div>
            <div className="info">
                <RichText
                    tagName="h2"
                    placeholder={ __( "Add Person Title", "_vt" ) }
                    onChange= { onChangeHeader }
                    value={ title }
                />
                <h3>General Manager</h3>
                <div className="text">
                    <p>People see me, and they see the suit, and they go: 'you're not
                    fooling anyone', they know I'm rock and roll through and through.</p>
                    <p>But you know that old thing, live fast, die young? Not my way.
                    Live fast, sure, live too bloody fast sometimes, but die young? Die old.</p>
                </div>
                <a href="https://www.wernham-hogg-limited.com" className="website">Wernham Hogg</a>
            </div>
        </div>
    );
},
```

Making the title editable https://github.com/verytwisty/vt*gutenberg*testimonials/blob/master/blocks/2-edit-text-block/index.js

Add in Attributes from props, and pull out title as a deconstructed variable

```
edit: props => {
    const { className, isSelected, attributes: { title } } = props;
    const onChangeHeader = title => { setAttributes( { title } ) };
    return (
        <div className={ className }>
            <div className="image">
                <img src="/wp-content/plugins/vt_testimonials/assets/images/David-Brent.jpg" />
            </div>
            <div className="info">
                <RichText
                    tagName="h2"
                    placeholder={ __( "Add Person Title", "_vt" ) }
                    onChange= { onChangeHeader }
                    value={ title }
                />
                <h3>General Manager</h3>
                <div className="text">
                    <p>People see me, and they see the suit, and they go: 'you're not
                    fooling anyone', they know I'm rock and roll through and through.</p>
                    <p>But you know that old thing, live fast, die young? Not my way.
                    Live fast, sure, live too bloody fast sometimes, but die young? Die old.</p>
                </div>
                <a href="https://www.wernham-hogg-limited.com" className="website">Wernham Hogg</a>
            </div>
        </div>
    );
},
```

Making the title editable https://github.com/verytwisty/vt*gutenberg*testimonials/blob/master/blocks/2-edit-text-block/index.js

We are going to add in the Gutenberg Component called RichText.

- we have a some props we can give this component here

- tagName - the wrapper for the text area

- Placeholder text

- onChange, what function to call when the user inputs text

- Value is the title attribute we pulled in from props

What is a React Component?

A component is a JavaScript class or function that optionally accepts inputs i.e. properties(props) and returns a React element that describes how a section of the page should appear.

```
// greeting.js
export default Greeting{
    return(
        <h2>Hello { this.props.name }</h2>
    );
}

// index.js
import Greeting from greeting;

ReactDOM.render(
        <Greeting name="World"/>,
        document.getElementById('root')
);
```

Here we have 2 files, one is called Greeting.js, and this is going to be our React Component. This works in a similar fashion as our Gutenberg Component. It will take one property or props called name. The second file is our React App. This is similar to our Gutenberg block, it will render out our Greeting component. We will pass our Greeting component name property. So what will be rendered to the page is a H2 tag with the text "Hello World".

Questions?

```
edit: props => {
    const { className, isSelected, attributes: { title } } = props;
    const onChangeHeader = title => { setAttributes( { title } ) };
    return (
        <div className={ className }>
            <div className="image">
                <img src="/wp-content/plugins/vt_testimonials/assets/images/David-Brent.jpg" />
            </div>
            <div className="info">
                <RichText
                    tagName="h2"
                    placeholder={ __( "Add Person Title", "_vt" ) }
                    onChange= { onChangeHeader }
                    value={ title }
                />
                <h3>General Manager</h3>
                <div className="text">
                    <p>People see me, and they see the suit, and they go: 'you're not
                    fooling anyone', they know I'm rock and roll through and through.</p>
                    <p>But you know that old thing, live fast, die young? Not my way.
                    Live fast, sure, live too bloody fast sometimes, but die young? Die old.</p>
                </div>
                <a href="https://www.wernham-hogg-limited.com" className="website">Wernham Hogg</a>
            </div>
        </div>
    );
},
```

On Chnage Event https://github.com/verytwisty/vt*gutenberg*testimonials/blob/master/blocks/2-edit-text-block/index.js

onChangeHeader function sets the attributes from user input from the header component

onChange event

```
// ES6 / React function
const onChangeHeader = title => { setAttributes( { title } ) };
const onChangeHeader = updatedText => { setAttributes( { title: updatedText } ) };

<RichText onChange={ onChangeHeader } />

// Traditional function
function onChangeHeader( title ){
    setAttributes( { title: title }
};
```

```
<input onkeydown="onChangeHeader()">
```

- this function put in a variable
- it only takes one variable - title, so we can get rid of the brackets
- Then we call the setAttributes function.
- Because the name of the variable is the same as the attribute we want to update, we can just write title once
- if we passed in a variable with a different name, we could write it like this
- Any Questions about this function?

```
save: props => {
    const { attributes: { title }, className } = props;
    return (
        <div className={ className }>
            <div className="image">
                <img src="/wp-content/plugins/vt_testimonials/assets/images/austin-distel-675050-unsplash.jpg" />
            </div>
            <div className="info">
                <h2>{ title }</h2>
                <h3>General Manager</h3>
                <div className="text">
                    <p>People see me, and they see the suit, and they go: 'you're not
                    fooling anyone', they know I'm rock and roll through and through.</p>
                    <p>But you know that old thing, live fast, die young? Not my way.
                    Live fast, sure, live too bloody fast sometimes, but die young? Die old.</p>
                </div>
                <a href="https://www.wernham-hogg-limited.com" className="website">Wernham Hogg</a>
            </div>
        </div>
    );
},
```

Save Function https://github.com/verytwisty/vt*gutenberg*testimonials/blob/master/blocks/2-edit-text-block/index.js

Going back to our editable block componentent - in the save section we will pull in the title attribute from props, like we did in the edit section, and then add the title variable value to the JSX. Questions?

Adding the extra text elements

```
export default registerBlockType(
    'testimonials/title',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
            title: {
                type: 'array',
                source: 'children',
                selector: 'h2',
            },
            jobTitle:{
                type: 'array',
                source: 'children',
                selector: 'h3',
            },
            text: {
                type: 'array',
                source: 'children',
                selector: '.text',
            }
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

Now we have done one text area, we need to add in the other text areas. We need to add attributes for these areas, a h3 for the job title and a classname for the text area.

```
edit: props => {
    const { attributes: { title, jobTitle, text }, className, setAttributes } = props;
    const onChangeHeader = title => { setAttributes( { title } ) };
    const onChangeTitle = jobTitle => { setAttributes( { jobTitle } ) };
    const onChangeText = text => { setAttributes( { text } ) };
    return (
        <div className={ className }>
            <div className="image">
                <img src="/wp-content/plugins/vt_testimonials/assets/images/austin-distel-675050-unsplash.jpg" />
            </div>
            <div className="info">
                <RichText
                    tagName="h2"
                    placeholder={ __( "Add Person Title", "_vt" ) }
                    onChange= { onChangeHeader }
                    value={ title }
                />
                <RichText
                    tagName="h3"
                    placeholder={ __( "Add Job Title", "_vt" ) }
                    onChange= { onChangeTitle }
                    value={ jobTitle }
                />
                <RichText
                    tagName="div"
                    placeholder={ __( "Add Testimonial here", "_vt" ) }
                    onChange= { onChangeText }
                    value={ text }
                    wrapperClassName=".text"
                />
                <a href="https://www.wernham-hogg-limited.com" className="website">Wernham Hogg</a>
            </div>
        </div>
    );
},
```

Change all Text Inputs https://github.com/verytwisty/vt*gutenberg*testimonials/blob/master/blocks/2-edit-text-block/index.js

- pull in the other attributes from props
- add two more RichText Components
- Change the tagName to h3 and div
- give the div a class name (so if you have more than one div, Gutenberg knows where to add the text saved in attributes)
- Add two more onChange functions

```
save: props => {
    const { attributes: { title, jobTitle, text }, className } = props;
    return (
        <div className={ className }>
            <div className="image">
                <img src="/wp-content/plugins/vt_testimonials/assets/images/austin-distel-675050-unsplash.jpg" />
            </div>
            <div className="info">
                <h2>{ title }</h2>
                <h3>{ jobTitle }</h3>
                <div className="text">
                    { text }
                </div>
                <a href="https://www.wernham-hogg-limited.com" className="website">Wernham Hogg</a>
            </div>
        </div>
    );
},
```

Save Function https://github.com/verytwisty/vt*gutenberg*testimonials/blob/master/blocks/2-edit-text-block/index.js

As before we are going to bring in the attributes into the save function and then add it to the JSX. Any Questions about that? Congratluations, that is our first editable block. Now we are going to look how to edit the image and url

Adding new componants to the project

1. Import new componant
2. Add new attributes
3. Add in Component to the edit object
4. Add in new update attributes function
5. Change the save function to include variables

Now we are going to bring in more components to make the image and link box editable. We are going to go through the same steps every time we bring in a new component...

Adding editable Image

The first thing we are going to do is make the image editable [show block]

```
const { RichText, MediaUpload } = wp.editor;
const { Button } = wp.components;
export default registerBlockType(
    'testimonials/editableblock',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
            imgURL: {
                type: 'string',
                source: 'attribute',
                attribute: 'src',
                selector: 'img',
            },
            imgID: {
                type: 'number',
            },
            imgAlt: {
                type: 'string',
                source: 'attribute',
                attribute: 'alt',
                selector: 'img',
            },
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

Import Componants `https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/3-editable-block/index.js`

Import the Media Upload Component

```
const { RichText, MediaUpload } = wp.editor;
const { Button } = wp.components;
export default registerBlockType(
    'testimonials/editableblock',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
            imgURL: {
                type: 'string',
                source: 'attribute',
                attribute: 'src',
                selector: 'img',
            },
            imgID: {
                type: 'number',
            },
            imgAlt: {
                type: 'string',
                source: 'attribute',
                attribute: 'alt',
                selector: 'img',
            },
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

Image Attributes https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/3-editable-block/index.js

Add in the image Attributes, there are 3

- image url the source is an attribute, which in this case is the src
- image Alt the source is also an attribute, but is the alt tag
- image ID is the WP id of the image

```
edit: props => {
    attributes... functions...
    return(
        <div className="image">
            {
                ! imgID ? (
                    <MediaUpload onSelect={ onSelectImage } type="image" value={ imgID }
                        render={ ( { open } ) => (
                            <Button className={ "button button-large" } onClick={ open } >
                                { __( 'Add Image', '_vt' ) }
                            </Button>
                        ) } >
                    </MediaUpload>
                ) : (
                    <React.Fragment>
                    <img src={ imgURL } alt={ imgAlt } />
                    { isSelected ? (
                        <Button className="remove-image" onClick={ onRemoveImage }>
                            Remove Image
                        </Button>
                    ) : ( null )
                    }
                    </React.Fragment>
                )
            }
        </div>
    )
}
```

Edit - Add in Image JSX https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/3-editable-block/index.js

in the edit object, there are two parts to this code, what to display when no image has been chosen, and what to display when an image has been selected

```
<div className="image">
    {
        ! imgID ? /* if there is no image already uploaded */ (
            <MediaUpload
                onSelect={ onSelectImage }  // on Change event
                type="image" // Type of file to choose - can also be video or file
                value={ imgID } // ID of the chosen image
                render={ ( { open } ) => ( // Small render section pass in open function
                    <Button
                        className={ "button button-large" }
                        onClick={ open } > // Add a button
                        { __( 'Add Image', '_vt' ) }
                    </Button>
                ) } >
            </MediaUpload>
        ) : (
                ...
        )
    }
</div>
```

Media Upload https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/3-editable-block/index.js

The first part tests to see if the imageID has been set, the explaination mark means not set. If this has not been set we are going to show the media uploader component.

- onSelect - which function to run when the user has selected an image (we will look at this later)
- type - this can be image, video, file or all
- the value is the ID of the image chosen
- Render is this components own mini render method - so we can have a render inside of a render!
- The render method takes one variable - open, which is a function that will open the media library when it is clicked

```jsx
<div className="image">
    {
        ! imgID ? (
            ...
        ) : ( /* If an image has been set */
            <React.Fragment>
            <img // image attributes
                src={ imgURL }
                alt={ imgAlt }
            />
            { isSelected ? (
                    // if the block is selected, add a remove image button
                    <Button
                        className="remove-image"
                        onClick={ onRemoveImage }> Remove Image </Button>
                ) : ( null )
            }
            </React.Fragment>
        )
    }
</div>
```

Image JSX https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/3-editable-block/index.js

the second part of the image component is what to display when an image has been selected...

- So if there is an imageID we want the image to be displayed
- we have a standard image tag that has the imageURL and imageAlt attributes
- when the block is selected we want a button to appear allowing the user to remove the image if they wish
- We wrap the whole thing in a React fragment, JSX elements can't be siblings, and the button and image are siblings in this instance. We could wrap them both in a div but this is extra markup we don't need, and React Fragments don't output additional HTML.
- even though we still have a top level wrapper, the JSX within the brackets count as new return statement, so we can only return one top level element
- the button calls the onRemoveImage function that we are going to look at next
- Questions?

```
edit: props => {
    const { attributes: {  imgURL, imgID, imgAlt }, className, setAttributes, isSelected } = props;

    const onSelectImage = img => { /* when image is selected */
        setAttributes({ // change the attributes
            imgID: img.id,
            imgURL: img.sizes.medium.url,
            imgAlt: img.alt,
        })
    };
    const onRemoveImage = () => { /* Remove Image */
        setAttributes({ // reset the attributes
            imgID: null,
            imgURL: null,
            imgAlt: null,
        })
    };
    return(...)
}
```

onChange functions https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/3-editable-block/index.js

Now we can look at the two functions that are called
- onSelectImage - this is passed the image object that has the information about the image in it
- we can set multiple attributes with one variable.
- we can pull variables out of the image object - here we are pulling out image id, the url of the medium image size croppped image, and the alt tag
- If you explore the rest API you will be able to see all the image attributes you can pull in
- onRemoveImage - this doesn't take any variables as we are resetting all the image attributes to null

```
save: props => {
    const { attributes: { title, jobTitle, text, imgURL, imgAlt }, className } = props;
    return (
        <div className={ className }>
            <div className="image">
                <img
                    src={ imgURL }
                    alt={ imgAlt }
                />
            </div>
            <div className="info">
                <h2>{ title }</h2>
                <h3>{ jobTitle }</h3>
                <div className="text">
                    { text }
                </div>
                <a href="https://www.wernham-hogg-limited.com" className="website">Wernham Hogg</a>
            </div>
        </div>
    );

},
```

Save Functions https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/3-editable-block/index.js

The save object

- we can pull in the attributes as normal and add it to the image -
you could test to see if the ID has been set here as well, but I didn't do it in this example
- Questions?

Adding a URL input field

Next we are going to add in the url input field

```
const { RichText, MediaUpload, URLInput } = wp.editor;
const { Button, Tooltip } = wp.components;
export default registerBlockType(
    'testimonials/editableblock',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
            linkText: {
                type: 'string',
                source: 'text',
                selector: 'a',
            },
            url: {
                type: 'string',
                source: 'attribute',
                attribute: 'href',
                selector: 'a',
            }
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

Import URL Components https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/3-editable-block/index.js

We will import the URL Input component

```
const { RichText, MediaUpload, URLInput } = wp.editor;
const { Button, Tooltip, TextControl } = wp.components;
export default registerBlockType(
    'testimonials/editableblock',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
            linkText: {
                type: 'string',
                source: 'text',
                selector: 'a',
            },
            url: {
                type: 'string',
                source: 'attribute',
                attribute: 'href',
                selector: 'a',
            }
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

Link URL Attributes https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/3-editable-block/index.js

We are going to add two attributes, LinkText and URL

- The link text will set the value of the text between the A tags
- the URL has an attribute as the source and will set the href attribute of the link

```
edit: props => {
    attributes... functions...
    return(
    {
        isSelected ? ( /* if the block is selected */
            <React.Fragment>
                <TextControl // add text control
                    id="linkText"
                    label="Add name here"
                    value={ linkText }
                    onChange={ linkText => setAttributes( { linkText } ) } /> // set attributes inline
                <Tooltip text="Add Company Website">
                    { icon } Website
                </Tooltip>
                <URLInput // add url input
                    className="link-url"
                    value={ url }
                    onChange={ url => setAttributes( { url } ) } /> // set attributes inline
                </React.Fragment>
            ) : ( // what to display when not selected
                <a href="{ url }" className="website">{ linkText }</a>
            )
    }
)}
```

Edit - Link JSX https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/3-editable-block/index.js

- Again we want to test whether the block is selected, so the user can change the url easily.
- The text control is an input field where the user can write the link text
- The URL Input brings up the in built link pop-up

```
edit: props => {
    attributes... functions...
    return(
    {
        isSelected ? ( /* if the block is selected */
            <React.Fragment>
                <TextControl // add text control
                    id="linkText"
                    label="Add name here"
                    value={ linkText }
                    onChange={ linkText => setAttributes( { linkText } ) } /> // set attributes inline
                <Tooltip text="Add Company Website">
                    { icon } Website
                </Tooltip>
                <URLInput // add url input
                    className="link-url"
                    value={ url }
                    onChange={ url => setAttributes( { url } ) } /> // set attributes inline
            </React.Fragment>
        ) : ( // what to display when not selected
            <a href="{ url }" className="website">{ linkText }</a>
        )
    }
)}
```

Link setAttributes https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/3-editable-block/index.js

- you can also add the update function inline
- the link has the url and name from the attributes props
- Questions?

```
save: props => {
    const { attributes: { title, jobTitle, text, imgURL, imgAlt, linkText, url }, className } = props;
    return (
        <div className={ className }>
            <div className="image">
                <img
                    src={ imgURL }
                    alt={ imgAlt }
                />
            </div>
            <div className="info">
                <h2>{ title }</h2>
                <h3>{ jobTitle }</h3>
                <div className="text">
                    { text }
                </div>
                <a href={ url } className="website">{ linkText }</a>
            </div>
        </div>
    );

},
```

Update save function `https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/3-editable-block/index.js`

In the save function we grab the link attributes from props Questions?

Extending the functionality

Show the Block

Text Alignment Toolbar

```
import classnames from 'classnames';
const { RichText, MediaUpload, URLInput, AlignmentToolbar, BlockControls } = wp.editor;
const { Button, IconButton, Tooltip, TextControl } = wp.components;
export default registerBlockType(
    'testimonials/extendedblock',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
            textAlignment: {
                type: 'string'
            },
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

Import Componants `https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js`

Import Block controls from wp.editor

```
import classnames from 'classnames';
const { RichText, MediaUpload, URLInput, AlignmentToolbar, BlockControls } = wp.editor;
const { Button, IconButton, Tooltip, TextControl } = wp.components;
export default registerBlockType(
    'testimonials/extendedblock',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
            textAlignment: {
                type: 'string'
            },
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

Add Attributes https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

Add the textAlignment to the attributes

```
edit: props => {
    attributes... functions...
    return[
        <BlockControls>
            <AlignmentToolbar
                value={ textAlignment }
                onChange={ textAlignment => setAttributes( { textAlignment } ) }
            />
        </BlockControls>,

        <div className={ className }> ...
            <div className="info" style={ { textAlign: textAlignment } }>...</div>
        </div>
    ];
}
```

Add Block Controlls JSX https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

In the edit object add in the Block Controls component

- within the block controls we are going to add in the Alignment toolbar

- Remember when I said that the return statement can only have one parent element? I was kind of lying!

- It can also return an array of elements, so long as each item in the array only has one top level element we are golden!

```
edit: props => {
    attributes... functions...
    return[
        <BlockControls>
            <AlignmentToolbar
                value={ textAlignment }
                onChange={ textAlignment => setAttributes( { textAlignment } ) }
            />
        </BlockControls>,

        <div className={ className }> ...
            <div className="info" style={ { textAlign: textAlignment } }>...</div>
        </div>
    ];
}
```

Add Change Attributes https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

- We can add in a change attribute function inline as well
- in the JSX we can add a new inline style to a div to update how the text is going to be displayed
- note in css there should be a hyphen between text-align, in JS you can't have hypens in this context (this is the same with vanilla JS) so we have to use cammel case for style attributes. This will output as normal css in the browser.

```
save: props => {
    const { attributes: { title, jobTitle, text, imgURL, imgAlt,
        linkText, url, textAlignment }, className } = props;
    return (
        <div className={ className }>
            <div className="image">
                <img src={ imgURL } alt={ imgAlt } />
            </div>
            <div className="info" style={ { textAlign: textAlignment } }>
                <h2>{ title }</h2>
                <h3>{ jobTitle }</h3>
                <div className="text">
                    { text }
                </div>
                <a href={ url } className="website">{ linkText }</a>
            </div>
        </div>
    );
},
```

Save https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

in the save object we can pull in the textAlign attribute and add it to the inline style of a div - Questions?

Block Alignment Toolbar

```
const { RichText, MediaUpload, URLInput, AlignmentToolbar, BlockControls, BlockAlignmentToolbar } = wp.editor;
const { Button, IconButton, Tooltip, TextControl } = wp.components;
export default registerBlockType(
    'testimonials/extendedblock',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
            blockAlignment: {
                type: 'string',
                default: 'wide',
            },
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

Import BlockAlignment Componant https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

Lastly we are going to add in the Block Controls component

```
const { RichText, MediaUpload, URLInput, AlignmentToolbar, BlockControls, BlockAlignmentToolbar } = wp.editor;
const { Button, IconButton, Tooltip, TextControl } = wp.components;
export default registerBlockType(
    'testimonials/extendedblock',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
            blockAlignment: {
                type: 'string',
                default: 'wide',
            },
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

Import BlockAlignment Componant `https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js`

Add blockAlignment to the attributes

```
getEditWrapperProps( { blockAlignment } ) {
    if ( 'left' === blockAlignment || 'right' === blockAlignment || 'full' === blockAlignment ) {
        return { 'data-align': blockAlignment };
    }
},
edit: props => {
    attributes... functions...
    return[
            <BlockControls>
                <AlignmentToolbar
                    value={ textAlignment }
                    onChange={ textAlignment => setAttributes( { textAlignment } ) }
                />
                 <BlockAlignmentToolbar
                    value={ blockAlignment }
                    onChange={ blockAlignment => setAttributes( { blockAlignment } ) }
                />
            </BlockControls>,

            <div className={ className } > ... </div>

    ]
}
```

Add Block Alignment Toolbar JSX https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

In the BlockControls toolbar add in the BlockAlignmentToolbar before or after the alignment toolbar

```
getEditWrapperProps( { blockAlignment } ) {
    if ( 'left' === blockAlignment || 'right' === blockAlignment || 'full' === blockAlignment ) {
        return { 'data-align': blockAlignment };
    }
},
edit: props => {
    attributes... functions...
    return[
            <BlockControls>
                <AlignmentToolbar
                    value={ textAlignment }
                    onChange={ textAlignment => setAttributes( { textAlignment } ) }
                />
                 <BlockAlignmentToolbar
                    value={ blockAlignment }
                    onChange={ blockAlignment => setAttributes( { blockAlignment } ) }
                />
            </BlockControls>,

            <div className={ className } > ... </div>

    ]
}
```

Update Attributes https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

- Add in the update attributes function inline as normal.
- You need an extra function to make this work...

Gutenberg wrapps every block in a parent container that you can't change in the edit object - the getEditWrapperProps function adds a data-align attribute to parent container of each block, which will automatically change the width of the parent container of the block - this is needed to make the block full width.

```
save: props => {
    const { attributes: { title, jobTitle, text, imgURL, imgAlt, linkText, url,
        textAlignment, blockAlignment }, className } = props;
    return (
        <div className={ classnames( `align${blockAlignment}`, className )} >
            <div className="image">
                <img src={ imgURL } alt={ imgAlt } />
            </div>
            <div className="info" style={ { textAlign: textAlignment } }>
                <h2>{ title }</h2>
                <h3>{ jobTitle }</h3>
                <div className="text">
                    { text }
                </div>
                <a href={ url } className="website">{ linkText }</a>
            </div>
        </div>
    );
},
```

Save https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

In the save function add in the block alignment class to the parent block. If you write it in exactly this way the classname is the same as the default block, so you will get this to work out of the box.

- If you want to add your own styles that aren't the default, you can change the class name here and write your own styles
- the classnames function allows you to add in more than one classname as React doesn't allow you to do this out of the box
- the way of writing variables with backticks is an ES6y type way of adding in a string of text with one variable between a $ and the curly braces

```
add_action( 'after_setup_theme', 'vt_theme_support' );

function vt_theme_support() {

    // Add support for full and wide align images.
    add_theme_support( 'align-wide' );

}
```

Add Theme Support in THEME functions.php

One last note on this is the align wide button will only be available if it is added to the theme support function in the themes functions.php as not all themes are full width, some have sidebars etc - Questions?

The Side Panel

```
const { RichText, MediaUpload, URLInput, AlignmentToolbar, BlockControls, BlockAlignmentToolbar,
    InspectorControls, PanelBody, PanelRow } = wp.editor;
const { Button, IconButton, Tooltip, TextControl } = wp.components;
export default registerBlockType(
    'testimonials/extendedblock',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

Attributes https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

- You can also add options and extra information in the sidepanel
- Import the InspectorControls, PanelBody, PanelRow from the wp.editor

```
edit: props => {
    attributes... functions...
    return[
        <InspectorControls>
            <PanelBody title={ __( 'Panel Section Name', 'vt' ) } >
                <PanelRow> This is some text </PanelRow>
                <PanelRow> This are options </PanelRow>
            </PanelBody>
        </InspectorControls>,
        <BlockControls>
                <AlignmentToolbar />
                 <BlockAlignmentToolbar />
            </BlockControls>,
            <div className={ className } > ... </div>

    ]
}
```

Edit https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

- Add in the InspectorControls componant in the edit object
- You can add in different sections to the control panel to break up the options
- A Panel body can have a title added to it
- Panel Row will add a section within the panel body
- Questions?

Add Background Colours to the block.

```
const { RichText, MediaUpload, URLInput, AlignmentToolbar, BlockControls, BlockAlignmentToolbar,
    InspectorControls, PanelBody, PanelRow,PanelColorSettings, ColorPalette } = wp.editor;
const { Button, IconButton, Tooltip, TextControl } = wp.components;
export default registerBlockType(
    'testimonials/extendedblock',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
            colorPalette: {
                type: "string",
                default: "#FF6F61"
              },
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

import colour panel block https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

Import the ColorPalette in from wp.editor

```
const { RichText, MediaUpload, URLInput, AlignmentToolbar, BlockControls, BlockAlignmentToolbar,
    InspectorControls, PanelBody, PanelRow,PanelColorSettings, ColorPalette } = wp.editor;
const { Button, IconButton, Tooltip, TextControl } = wp.components;
export default registerBlockType(
    'testimonials/extendedblock',
    {
        title: ... , description: ..., category: ..., icon: ..., keywords: ...,
        attributes: {
            colorPalette: {
                type: "string",
                default: "#FF6F61"
             },
        },
        edit: props => { ... },
        save: props => { ... },
    },
);
```

Add Attributes https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

- Add in new attributes, the type is a string and we can add in a default colour if you like

```
edit: props => {
    attributes... functions...
    return[
        <InspectorControls>
            <PanelColorSettings
              title={__("Background Colour", "_vt")}
              colorSettings={[
                {
                  value: colorPalette,
                  onChange: colorPalette => { setAttributes({ colorPalette }); },
                  label: __("Select BG Colour")
                }
              ]}
            />
            </InspectorControls>,
            <BlockControls>
                <BlockAlignmentToolbar />
            </BlockControls>,
        <div className={ className } style={ { backgroundColor: colorPalette } }> ... </div>
    ]
}
```

Edit https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

- In the inspector controls pallet, we don't need to add in the panel body as for PanelColorSettings has a panel body included, where we can add the title.
- The value takes the colorPalette attribute, which will display the default colour or the selected colour if it is chosen.
- the label the text above the input

```
edit: props => {
    attributes... functions...
    return[
        <InspectorControls>
            <PanelColorSettings
              title={__("Background Colour", "_vt")}
              colorSettings={[
                {
                  value: colorPalette,
                  onChange: colorPalette => { setAttributes({ colorPalette }); },
                  label: __("Select BG Colour")
                }
              ]}
            />
            </InspectorControls>,
            <BlockControls>
                <BlockAlignmentToolbar />
            </BlockControls>,
        <div className={ className } style={ { backgroundColor: colorPalette } }> ... </div>
    ]
}
```

Edit https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

- the onChange function changes the colorPallet attribute
- the we can add an inline style to the parent div (or any other div)
- The css attribute has to be cammel case again and the value is taken from the colorPalette attribute
- Any Questions?

```
save: props => {
    const { attributes: { title, jobTitle, text, imgURL, imgAlt, linkText, url,
        textAlignment, blockAlignment, colorPalette }, className } = props;
    return (
        <div
        className={ classnames(
          `align${blockAlignment}`,
           className,
        )}
        style={ { backgroundColor: colorPalette } }
        >
            ...

        </div>
    );
},
```

Save https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

- In the save object, we can pull in the colorPalette from props
- Then we can add an inline style to the parent div

```php
add_theme_support(
    'editor-color-palette',
    array(
        array(
            'name'  => __( 'Coral', 'twentynineteen' ),
            'slug'  => 'coral',
            'color' => '#FF6F61',
        ),
        array(
            'name'  => __( 'Magenta Haze', 'twentynineteen' ),
            'slug'  => 'magenta-haze',
            'color' => '#9D446E',
        ),
        array(
            'name'  => __( 'Mauvewood', 'twentynineteen' ),
            'slug'  => 'mauvewood',
            'color' => '#A75D67',
        ),
        array(
            'name'  => __( 'Forest Biome', 'twentynineteen' ),
            'slug'  => 'forest-biome',
            'color' => '#184A47',
        ),
        array(
            'name'  => __( 'Grey', 'twentynineteen' ),
            'slug'  => 'grey',
            'color' => '#B5BAB6',
        ),
    )
);
```

Add theme Support in THEME functions.php https://github.com/verytwisty/vt_gutenberg_testimonials/blob/master/blocks/4-extended-block/index.js

- In the theme functions we can add in some colours that will appear in the colour pallet block. this can then be customised on a theme by theme basis.

🎉 Congratulations 🎉

Congratulations! We have a fully editable block!
- Any Questions about what we have learnt so far?

Addendum Component Libraries

How do we know what components to add in? We can discover them in different ways.
[go to gutenberg and in the terminal type in wp]

Exploring available blocks

- wp.editor
  - https://github.com/WordPress/gutenberg/tree/master/packages/editor/src/components
- wp.components
  - https://github.com/WordPress/gutenberg/tree/master/packages/components/src
  - https://wordpress.org/gutenberg/handbook/designers-developers/developers/components/
- Data
  - https://github.com/WordPress/gutenberg/tree/master/packages/data/src/components
  - https://wordpress.org/gutenberg/handbook/designers-developers/developers/data/

Or we can go to the Gutenberg Git Hub library or the WordPress Gutenberg handbook

wp.editor

AlignmentToolbar	Autocomplete	AutosaveMonitor	BlockAlignmentToolbar
BlockControls	BlockEdit	BlockFormatControls	BlockIcon
BlockInspector	BlockList	BlockMover	BlockNavigationDropdown
BlockSelectionClearer	BlockSettingsMenu	BlockTitle	BlockToolbar
ColorPalette	ContrastChecker	CopyHandler	DefaultBlockAppender
DocumentOutline	DocumentOutlineCheck	EditorGlobalKeyboardShortcuts	EditorHistoryRedo
EditorHistoryUndo	EditorNotices	EditorProvider	ErrorBoundary
FontSizePicker	InnerBlocks	Inserter	InspectorAdvancedControls

These are things that you might find in a typical Editor for example RichText editor, MediaUploader, Font Size Picker

wp.editor

InspectorControls	MediaPlaceholder	MediaUpload	MediaUploadCheck
MultiBlocksSwitcher	MultiSelectScrollIntoView	NavigableToolbar	ObserveTyping
PageAttributesCheck	PageAttributesOrder	PageAttributesParent	PageTemplate
PlainText	PostAuthor	PostAuthorCheck	PostComments
PostExcerpt	PostExcerptCheck	PostFeaturedImage	PostFeaturedImageCheck
PostFormat	PostFormatCheck	PostLastRevision	PostLastRevisionCheck
PostLockedModal	PostPendingStatus	PostPendingStatusCheck	PostPingbacks
PostPreviewButton	PostPublishButton	PostPublishButtonLabel	PostPublishPanel

wp.editor

PostSavedState	PostSchedule	PostScheduleCheck	PostScheduleLabel
PostSticky	PostStickyCheck	PostSwitchToDraftButton	PostTaxonomies
PostTaxonomiesCheck	PostTextEditor	PostTitle	PostTrash
PostTrashCheck	PostTypeSupportCheck	PostVisibility	PostVisibilityCheck
PostVisibilityLabel	PreserveScrollInReorder	RichText	RichTextInserterItem
RichTextShortcut	RichTextToolbarButton	ServerSideRender	SkipToSelectedBlock
TableOfContents	URLInput	URLInputButton	URLPopover
UnsavedChangesWarning	Warning	WordCount	WritingFlow

wp.editor

autop:	blockAutocompleter	cleanForSlug	getColorClassName
getColorObjectByColorValue	getContent	getDefaultSetting	getFontSize
getFontSizeClass	initialize:	mediaUpload	remove
userAutocompleter	withColorContext	withColors	withFontSizes
_BlockSettingsMenuFirstItem	_BlockSettingsMenuPluginsExtension		

wp.components

Autocomplete	BaseControl	Button	ButtonGroup
CheckboxControl	Circle	ClipboardButton	ColorIndicator
ColorPalette	ColorPicker	Dashicon	DatePicker
Disabled	Draggable	DropZone	DropZoneProvider
Dropdown	DropdownMenu	ExternalLink	Fill
FocusableIframe	FontSizePicker	FormFileUpload	FormToggle
FormTokenField	G	Icon	IconButton
KeyboardShortcuts	MenuGroup	MenuItem	MenuItemsChoice

These are smaller items that make up elements of the editor, like icons, buttons, spinners & dropdowns

wp.components

Modal	NavigableMenu	Notice	NoticeList
Panel	PanelBody	PanelHeader	PanelRow
Path	Placeholder	Polygon	Popover
QueryControls	RadioControl	RangeControl	Rect
ResizableBox	ResponsiveWrapper	SVG	SandBox
ScrollLock	SelectControl	ServerSideRender	Slot
SlotFillProvider	Spinner	TabPanel	TabbableContainer
TextControl	TextareaControl	TimePicker	ToggleControl

wp.components

Toolbar	ToolbarButton	Tooltip	TreeSelect
createSlotFill	navigateRegions	withConstrainedTabbing	withFallbackStyles
withFilters	withFocusOutside	withFocusReturn	withNotices
withSpokenMessages			

wp.blocks

children	cloneBlock	createBlock	doBlocksMatchTemplate
findTransform	getBlockAttributes	getBlockContent	getBlockDefaultClassName
getBlockMenuDefaultClassName	getBlockSupport	getBlockTransforms	getBlockType
getBlockTypes	getCategories	getChildBlockNames	getDefaultBlockName
getFreeformContentHandlerName	getPhrasingContentSchema	getPossibleBlockTransformations	getSaveContent
getSaveElement	getUnregisteredTypeHandlerName	hasBlockSupport	hasChildBlocks
hasChildBlocksWithInserterSupport	isReusableBlock	isUnmodifiedDefaultBlock	isValidBlockContent
isValidIcon	node	normalizeIconObject	parse

This is where you create your block from Register Block Type. Also has some more admin functions for blocks

wp.blocks

parseWithAttributeSchema	pasteHandler	rawHandler	registerBlockStyle
registerBlockType	serialize	setCategories	setDefaultBlockName
setFreeformContentHandlerName	setUnregisteredTypeHandlerName	switchToBlockType	synchronizeBlocksWithTemplate
unregisterBlockStyle	unregisterBlockType	unstable__bootstrapServerSideBlockDefinitions	updateCategory
withBlockContentContext			

wp.i18n

setLocaleData	sprintf	__
_n	_nx	_x

Useful functions for translations

wp.date

date	dateI18n	format
getDate	gmdate	isInTheFuture
setSettings		

If you need to grab the date, do so here.

wp.data

RegistryConsumer	RegistryProvider	combineReducers
createRegistry	dispatch	plugins
registerGenericStore	registerStore	select
subscribe	use	withDispatch
withSelect		

Allows you to grab & create dynamic data, such as withSelect

All information from this talk has been learnt on Zac Gordon's Gutenberg Block Development Course, please check it out for more detail and information...

50% off all Gutenberg courses for one week!

https://javascriptforwp.com/product/gutenberg-block-development-course/

Resources

Gutenberg Handbook
Gutenberg starter theme
Gutenberg block reference
Gutenberg starter tutorial on CSS Tricks
Coding a Custom Block Type for Gutenberg Block Editor (video)
WordPress Webinar: Building your First Gutenberg Block (video)
Learn React Tutorial
About Bable
Gutenberg Git Hub Repro