Code guidelines

> For consistency please read and follow these guidelines.

Creating files and folders

Do:

- lowercase
- dash-dividers
- `.module.js`, `.component.js`, `.jsx.js`, `.style.js`, `.service.js`, `.slice.js`, `.mock.js`, `.state.js` file endings
- descriptive file names

Don't:

- UPPERCASE
- CamelCase
- Random non-descriptive names

Components

Use function components, not class components.

Example:

```
const ExampleComponent = () => {
  return (
    ...
  );
}
```

Export & Import

> Spending 2 minutes organizing your exports and imports saves a lot of time later when you or someone else reads, refactors or debugs your code!

Use `export default` when exporting a component:

```
const ExampleComponent = () => {...}

export default ExampleComponent;
```

Place the exports in the bottom of the document.

With `export default` we can import like so:

```
import ExampleComponent from 'src/views/.../example.component.js';

<ExampleComponent />
```

Use the whole path to the file and include the extension at the end.

Export all variables from a file with `export {}` like so:

```
const example1 = '...';
const example2 = '...';
const example3 = '...';

export {
  example1,
  example2,
  example3,
}
```

Importing variables like so:

```
import {
  example1,
  example2,
  example3,
} from 'src/global/constants/example.js';
```

Stack multiple exports and imports!

What does this mean?

```
// Don't do this
import { example1, example2, example3 } from '...';

// Do this
import {
  example1,
  example2,
  example3,
} from '...';
```

Props(paramaters)

If you have 1 or 2 props in a component they can be destructured like this:

```
const ExampleComponent = ({ navigation, data }) => {...}
```

But once you have more props this quickly becomes unreadable.

You can then destructure them within the component like so:

```
const ExampleComponent = (props) => {

  const {
    navigation,
    data,
    clickHandler,
    ...
  } = props;

}
```

This goes for sending props as well; `props.data.person.name` is a no go!

Destructure the props and even the data if needed.

Sending props is also much more readable when stacked:

```
<PrevButton
  absolute={true}
  backgroundColor='#00b6cd'
  color='#fff'
  horizontal={true}
  onPress={() => console.log('PrevButton pressed!')}
  position={{
    top: '50%',
    left: '3%',
  }}
/>
```

The alternative would be a long unreadable line that goes off screen:

```
<PrevButton absolute={true} backgroundColor='#00b6cd' color='#fff'
horizontal={true} onPress={() => console.log('PrevButton pressed!')}
position={{ top: '50%', left: '3%' }} />
```

Variables

> Use descriptive variable names, if a variable name is too long change it.
>
> Difficult to write good variable names? Add a comment..

Using variables in string:

```
const firstName = 'Ola';
const lastName = 'Nordmann';

// Don't do this
console.log('First name: ' + firstName + ' Last name: ' + lastName);

// Do this
console.log(`First name: ${firstName} Last name: ${lastName}`);
```

Git guidelines

> ***Main/Beta branch are considered production branches!***
>
> Don't work directly in a production branch..

Refactor your code before merging!

Branches

> Make a pull request before creating a new branch to make sure you're working on the latest version.

Naming branches something that describes what your working on helps others in the team understanding what changes are being made in which branch. It's also very helpful later for anyone viewing 50 different branches, to know what's going and where.

On the note of 50 branches; delete branches when they have been merged.

Although there is no need to do this immediately, and we tend to let them go stale before deletion to be sure it's been merged and that we don't delete anything important.

Pushing to git

Whenever you push to git you should write a descriptive commit message.
This commit message should in short explain what changes you have made to the project.

The exception here is taking backup of your work at any point with a commit message like:

```
Saving changes
```

Which is something I tend to do because I'm afraid to lose any work.

Merging

When merging/creating a pull request to main/beta, the code should be production ready!

What do we mean by "production ready code"?

- Your code is clean, solid and dynamic.
- There is no `console.log()` throughout the project.
- You have added comments explaining your code where it's needed.
- You have added info in the change log.
- You have tested your code on both iOS and Android(developing on an emulator is fine, but testing is not!).