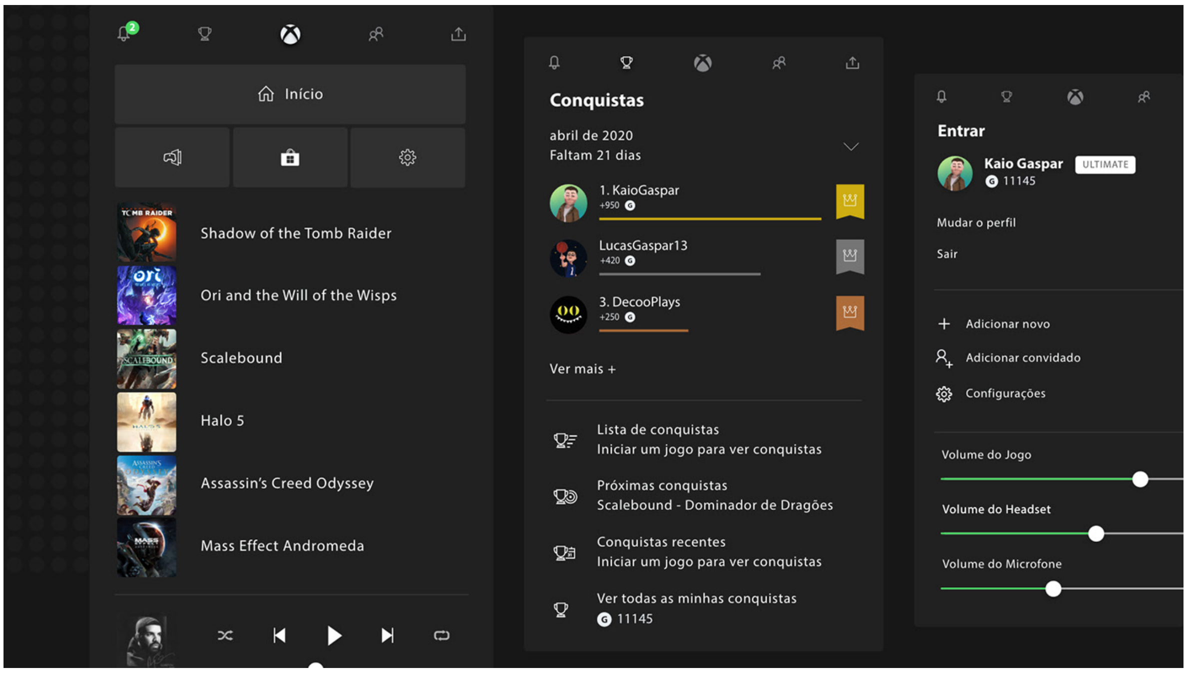
# Layouts



Layouts is one of the most important concepts in any Graphical User Interface (GUI) designing.

An application's User Interface may consist of several screens or pages. The Lay out of each screen is what is commonly referred to as the "layout", so layout is self explanatory. But what you may not know is, that certain parts of the screen may change but not those directly inside a layout. View allows positions of an item inside of it to change, but layouts don't, that is to say that there is another UI concept and component called views which will be discussed later.

So, Layouts are static whereas Views are dynamic. Views reside inside layouts, and also views may lay out themselves using layouts. So knowledge of layouts is indispensable.

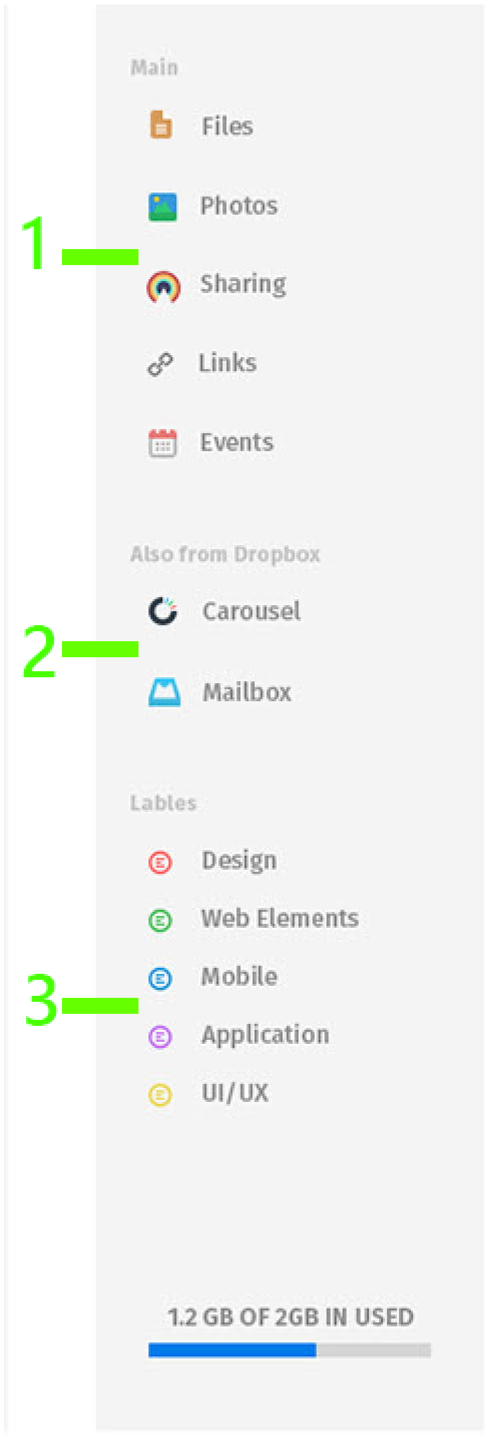
Qml has four(4) layout types, you can combine then to create whatever layout you want. The types are RowLayout (row layout), ColumnLayout (column layout), GridLayout (grid layout), and StackLayout (stack layout).

## ColumnLayout

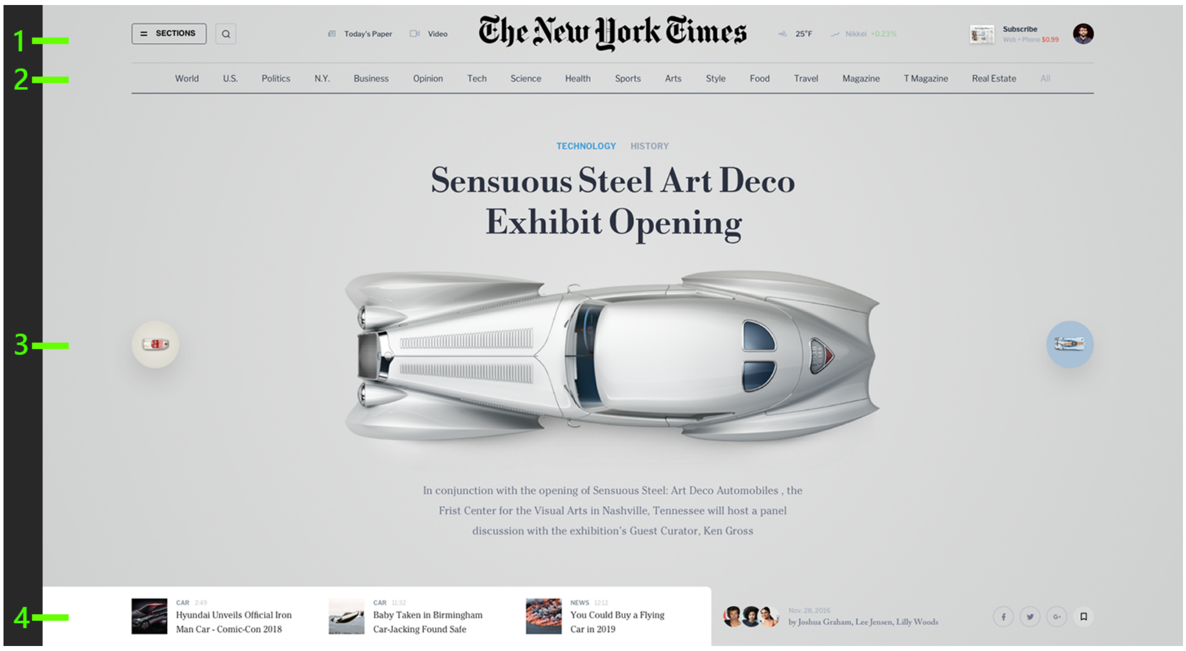
**ColumnLayout** - arranges its contents so that we get a column, but also each of the items in there is a row inside the column. So we have something like shown in the images below.

The numbers shows the direct contents of the Column layout. These are rows inside a column. Together they form a column.

*App sample 1 - the white pane is a background for the numbers*



*App sample 2 - the black pane is a background for the numbers*



Sensuous Steel Art Deco Exhibit Opening

In conjunction with the opening of Sensuous Steel: Art Deco Automobiles , the Frist Center for the Visual Arts in Nashville, Tennessee will host a panel discussion with the exhibition's Guest Curator, Ken Gross

CAR 2:49 Hyundai Unveils Official Iron Man Car - Comic-Con 2018

CAR 11:32 Baby Taken in Birmingham Car-Jacking Found Safe

NEWS 12:12 You Could Buy a Flying Car in 2019

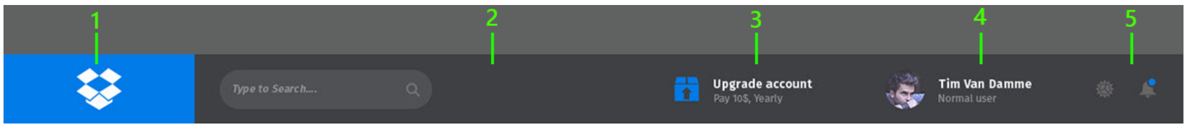
Nov. 28, 2016
by Joshua Graham, Lee Jensen, Lilly Woods

## RowLayout

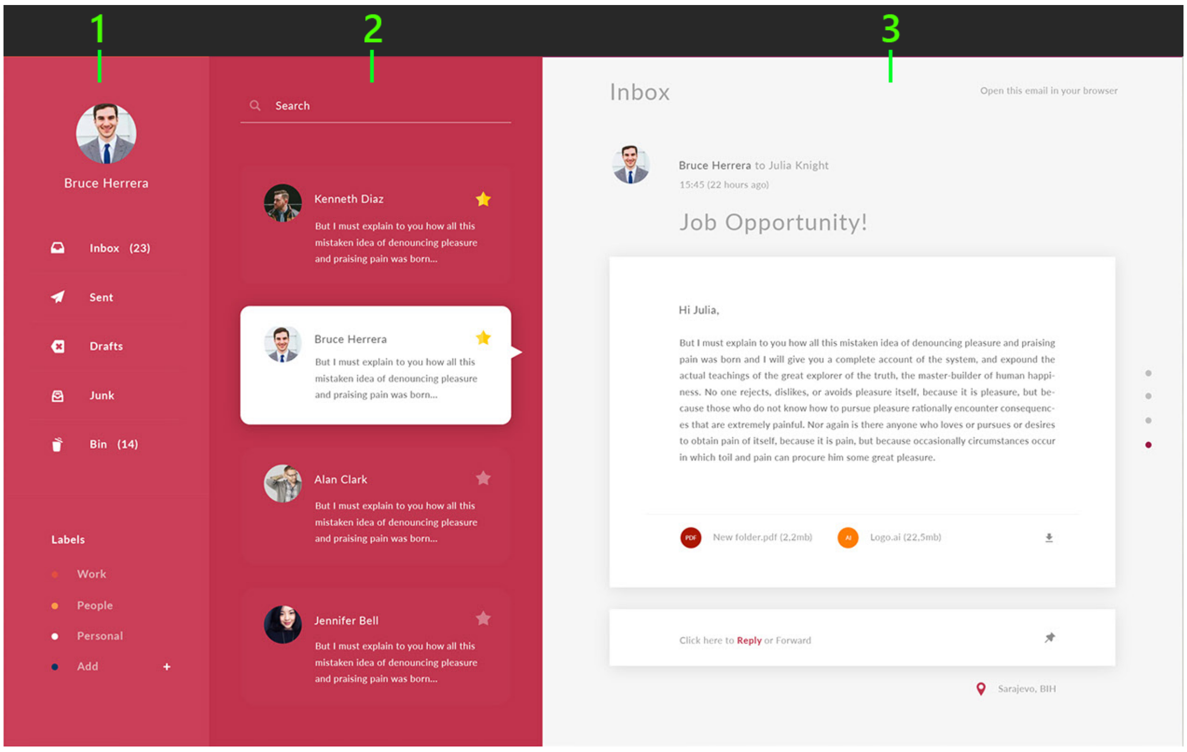
**RowLayout** - arranges its contents so that we get a row, but each of the items in there is a column inside the row. So we have something like shown in the images below.

The numbers shows the direct contents of the Row layout. These are columns inside a row. Together they form a row.

*App sample 1* - the dark grey pane is a background for the numbers



*App sample 2* - the black pane is a background for the numbers



But I must explain to you how all this mistaken idea of denouncing pleasure and praising pain was born...

Hi Julia,

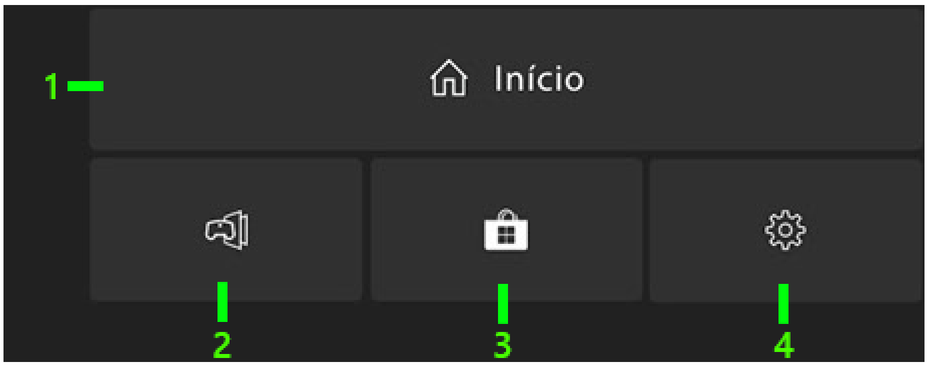
But I must explain to you how all this mistaken idea of denouncing pleasure and praising pain was born and I will give you a complete account of the system, and expound the actual teachings of the great explorer of the truth, the master-builder of human happiness. No one rejects, dislikes, or avoids pleasure itself, because it is pleasure, but because those who do not know how to pursue pleasure rationally encounter consequences that are extremely painful. Nor again is there anyone who loves or pursues or desires to obtain pain of itself, because it is pain, but because occasionally circumstances occur in which toil and pain can procure him some great pleasure.

## GridLayout

**GridLayout** - arranges its contents so that we get a grid, but each of the items in there is a cell inside the grid. When a grid layout is implemented. The number of rows and/or columns are indicated. Which means that the minimum width and/or height of each cell is indicated, so if a cell wants to be larger than the rest it can just say so and take up space belonging to other cells. So we have something like shown in the images below.

The numbers shows the direct contents of the Grid layout. These are cells inside the grid. Together they form a grid.

*App sample 1 - the black pane is a background for the numbers*



## StackLayout

**StackLayout** - arranges its contents so that only one is visible at any particular time, the rest are hidden, behind it, so to speak. Each of the items is called a stack item.

## Import layouts

To start using layouts in any qml file, you will need it imported in that qml file.

```
import QtQuick.Layouts 1.3
```

The digit after the dot corresponds with you QtQuick import. eg:

If you import *QtQuick 2.12*, you should import *QtQuick.Layouts 1.12,* for *QtQuick 2.13*, you import *QtQuick.Layouts 1.13*, and so on. Going back to *QtQuick 2.11*, *QtQuick.Layouts 1.4* is used and not (*QtQuick.Layouts 1.11*), from *QtQuick 2.10*, *QtQuick.Layouts 1.3* is used and so on.

So, the top most of your qml file looks like this at the least, if you want to work with layouts.

```
import QtQuick 2.12
import QtQuick.Layouts 1.12
```

From now on you can start declaring any qml object layout types, like we have explained about above.

***The following examples will assume you have a working window code. Like this***

```
import QtQuick 2.12
import QtQuick.Layouts 1.12
import QtQuick.Controls 2.12

ApplicationWindow {
    visible: true
    width: 800
    height: 400
    title: "Layouts"

}
```

With which you can add any example code like this:

```
Rectangle {
    width: 200
    height: 200
    color: "dodgerblue"
}
```

to it like this:

```
import QtQuick 2.12
import QtQuick.Layouts 1.12
import QtQuick.Controls 2.12

ApplicationWindow {
    visible: true
    width: 800
    height: 400
    title: "Layouts"

    Rectangle {
        width: 200
        height: 200
        color: "dodgerblue"
    }

}
```

Lets get going.

## Declaring dimensions

The Layout system imposes certain restrictions on Items declared inside a layout type like this.

```
RowLayout {  //a layout type

    Rectangle { // this is an item declared inside a layout type
        ...
    }

}
```

A layout type may or may not have declared dimensions

Does not have declared dimensions

```
RowLayout {

    Rectangle {
        ...
    }
}
```

**or**

Has declared dimensions

```
RowLayout {
    width: parent.width
    height: parent.height

    Rectangle {
        ...
    }

}
```

The Layout types can themselves have declared *dynamic* dimensions **eg.**

```
RowLayout {
    width: parent.width
    height: parent.height
}
```

But items in them are supposed to be managed by the layout system. So declaring dynamic dimensions will cause errors.

```
RowLayout {
    width: parent.width
    height: parent.height

    Rectangle {
        width: parent.width  // this will cause an error
        height: parent.height  // this too will cause an error
    }

}
```

The `parent` keyword is not allowed for dimensions here. It can provide other properties but not dimension based properties.

Static dimensions however are allowed

```
RowLayout {
    width: parent.width
    height: parent.height

    Rectangle {
        width: 200
        height: 200
    }

}
```

**Hack**: `parent.parent` is allowed (LOL)

```
Rectangle {
    width: 400
    height: 400

    RowLayout {
        width: parent.width
        height: parent.height

        Rectangle {
            width: parent.parent.width
            height: parent.parent.height
        }

    }

}
```

What is **recommended** or even **preferred** is that you use the Layout base type's attached properties. **eg.**

```
RowLayout {
    width: parent.width
    height: parent.height

    Rectangle {
        Layout.fillWidth: true  // good code
        Layout.fillHeight: true  // good code
    }

}
```

These are some of the preferred ways to use the layout types. That is to say, that there are a lot of attached properties, you can use. We will discuss only a few for now.

## Spacing

Items inside a layout can be spaced-out or not spaced-out. There is an important `spacing` property. Default is `5`.

**NB.** Not all layout types has this property.

```
RowLayout {
    width: parent.width
    height: parent.height
    spacing: 0
}
```

## Attached properties

Attached properties are provided by the `Layout` object type. It is used on the items declared inside a layout type.

**NB:** Qml object types always start with an uppercase letter.

### Layout.fillWidth

Provides a full width for items declared inside a layout type. The Layout.fillWidth actually takes all the remaining width.

#### Possible values

```
Layout.fillWidth: true
```

and

```
Layout.fillWidth: false
```

### Layout.fillHeight

Provides a full height for items declared inside a layout type. The Layout.fillHeight actually takes all the remaining height.

#### Possible values

```
Layout.fillHeight: true
```

and

```
Layout.fillHeight: false
```

### Layout.preferredWidth

Provides width for items declared inside a layout type. This accepts any numerical figure, again you cannot use the parent keyword to provide dimensions.

#### Possible values

```
Layout.preferredWidth: 200
```

### Layout.preferredHeight

Provides height for items declared inside a layout type

#### Possible values

```
Layout.preferredHeight: 200
```

## You should try out the following pieces of code

## ColumnLayout

Lets use Column Layout

### Declaration

```
ColumnLayout {
    code goes
    here
}
```

### Basic Usage

Say we have a file, *columnlayout.qml*

```
import QtQuick 2.15
import QtQuick.Layouts 1.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 400
    height: 400

    ColumnLayout {
        width: 400
        height: 400
        spacing: 0

        Rectangle {
            Layout.fillWidth: true
            Layout.preferredHeight: 40
            color: "darkgrey"
        }

        Rectangle {
            Layout.fillWidth: true
            Layout.fillHeight: true
            color: "dodgerblue"
        }

    }

}
```

When you run this in Ninja-Preview and you see that we have a basic Column Layout

## RowLayout

Lets use Row Layout

### Declaration

```
RowLayout {
    code goes
    here
}
```

### Basic Usage

Say we have a file, *rowlayout.qml*

```
import QtQuick 2.15
import QtQuick.Layouts 1.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 400
    height: 400

    RowLayout {
        width: parent.width
        height: parent.height
        spacing: 0

        Rectangle {
            Layout.preferredWidth: 40
            Layout.fillHeight: true
            color: "darkgrey"
        }

        Rectangle {
            Layout.fillWidth: true
            Layout.fillHeight: true
            color: "dodgerblue"
        }

    }

}
```

When you run this in Ninja-Preview and you see that we have a basic Row Layout

## GridLayout

Lets use Grid Layout

### Declaration

```
GridLayout {
    code goes
    here
}
```

## Basic Usage

Say we have a file, *gridlayout.qml*

```
import QtQuick 2.15
import QtQuick.Layouts 1.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 400
    height: 400

    GridLayout {
        width: 400
        height: 400
        rowSpacing: 8
        columnSpacing: 8
        columns: 2

        Rectangle {
            Layout.fillWidth: true
            Layout.fillHeight: true
            color: "dodgerblue"
        }

        Rectangle {
            Layout.fillWidth: true
            Layout.fillHeight: true
            color: "dodgerblue"
        }

        Rectangle {
            Layout.fillWidth: true
            Layout.fillHeight: true
            color: "dodgerblue"
        }

        Rectangle {
            Layout.fillWidth: true
            Layout.fillHeight: true
            color: "dodgerblue"
        }

    }

}
```

When you run this in Ninja-Preview and you see that we have a basic Grid Layout

# StackLayout

Lets use Stack Layout

## Declaration

```
StackLayout {
    code goes
    here
}
```

## Basic Usage

Say we have a file, *stacklayout.qml*

```
import QtQuick 2.15
import QtQuick.Layouts 1.15
import QtQuick.Controls 2.15

ApplicationWindow {
    visible: true
    width: 400
    height: 400

    StackLayout {
        width: 400
        height: 400
        currentIndex: 0

        Rectangle {
            Layout.fillWidth: true
            Layout.fillHeight: true

            Text {
                text: "You can see me"
            }

        }

        Rectangle {
            Layout.fillWidth: true
            Layout.fillHeight: true
            color: "gold"

            Text {
                text: "You can't see me unless you set currentIndex to 1"
            }

        }

    }

}
```

When you run this in Ninja-Preview and you see that we have a basic Stack Layout

## Layout

Layout type exists only to provide properties to use when defining your layouts. Actually all the other layout types inherent from this base type. You will not use it directly. When studying the other types you will see it being declared as

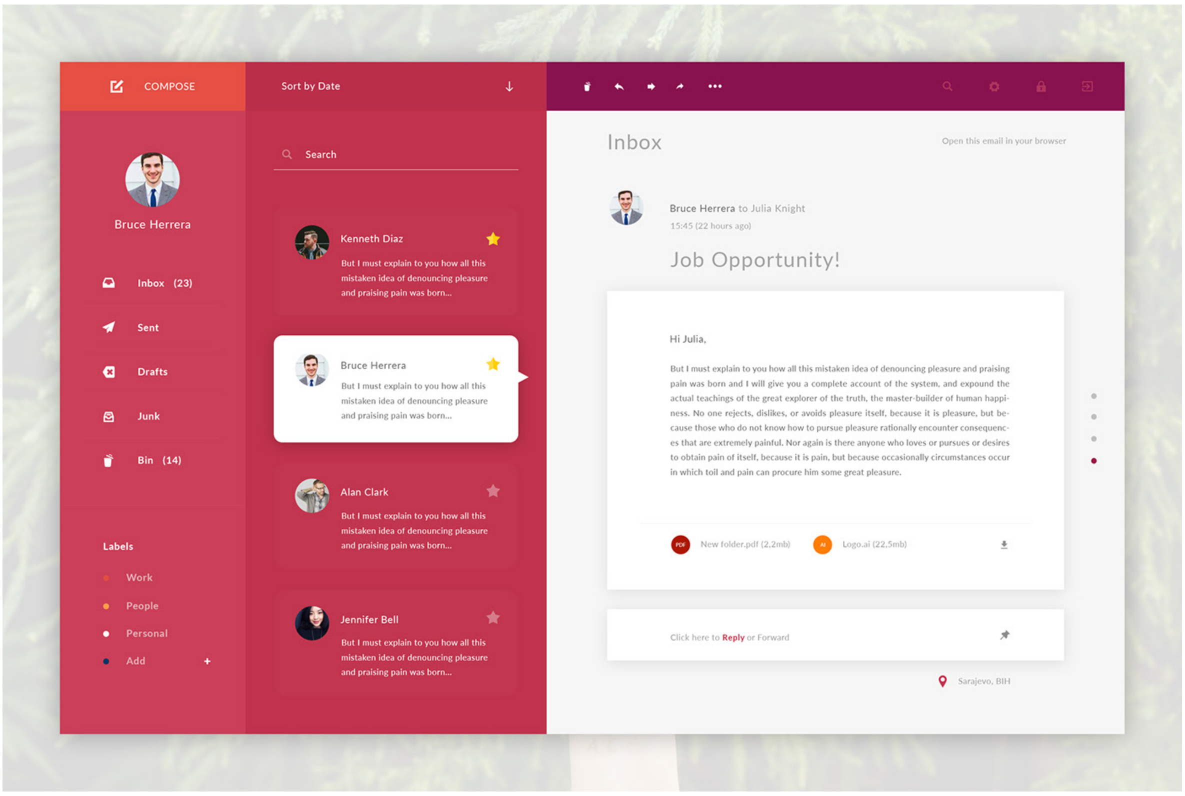
```
Layout.property: value
```

## Common Usage

The complete layouting of an application is designed using multiple combinations of row and column layouts sometimes also stack layouts, but almost never grid layout. That is to say that the Grid layout is the least used layout type, if the application seems to have a grid then it is a Grid view and not a Grid Layout.

The first layout first, then the other layouts will be inside of it. In that case we look for what layout should be the parent layout, then the other layout will conform to the various parts where they fit. The parent layout would be either of  RowLayout or ColumnLayout, as stack layout also may not be suitable as an overall parent layout, since it will require switching between what can be seen and what can't

*App sample 1*



But I must explain to you how all this mistaken idea of denouncing pleasure and praising pain was born...

Bruce Herrera to Julia Knight
15:45 (22 hours ago)

Job Opportunity!

Hi Julia,

But I must explain to you how all this mistaken idea of denouncing pleasure and praising pain was born and I will give you a complete account of the system, and expound the actual teachings of the great explorer of the truth, the master-builder of human happiness. No one rejects, dislikes, or avoids pleasure itself, because it is pleasure, but because those who do not know how to pursue pleasure rationally encounter consequences that are extremely painful. Nor again is there anyone who loves or pursues or desires to obtain pain of itself, because it is pain, but because occasionally circumstances occur in which toil and pain can procure him some great pleasure.

The above image shows an Application, with most probably a RowLayout as its parent layout. That is to say that it could have also been a columnLayout but based on the fact that the top nav seem dependent on the content below, a RowLayout is best suited for the job, even though there is also a drawer, which just snaps itself, and whatever parent layout you have is up to you.