# Assignment for Henrik Wassdahl

## Fixing a grid layout bug

Somewhere inside our dashboard layout code we have a function called `setGridAttributes` which is used by each dashboard tile (i.e. separate charts in dashboard) to assign layout attributes before drawing. Grid options are stored in model's `grid` attribute which contains data like this:

```
{
  height: 1,
  width: 1,
  x: 0,
  y: 4,
}
```

This data is either loaded from DB or assigned to a newly added dashboard tile. Then this data is enriched by `setGridAttributes` function and added as an attribute to actual DOM nodes where it's going to be used by the grid rendering library. In some cases `grid` position attributes `x` and `y` are not defined - for example, when we're adding a new chart to the dashboard. In this case `autoPosition` option should be used for this tile so grid layout library can automatically place tile at the end of the grid.

The issue we're having with this code is that it unexpectedly moves some tiles to a wrong position. Can you explain why this might be happening? *The issue must be fixed by changing the provided code sample.*

```
export function setGridAttributes() {
  const options = { ...this.model.get('grid'), id: this.model.id };

  if (!options.x || !options.y) {
    options.autoPosition = true;
  }

  Object.entries(options).forEach(([key, value]) => {
    this.$el.attr(`data-gs-${kebabCase(key)}`, value);
  });
}
```

## Fixing a chart drawing bug

Somewhere inside our chart drawing code we have a helper function called `getBoundingBoxFor`. It is used when we want to get screen dimensions for certain chart elements.

```
import { clone } from 'underscore';

function getBoundingBoxFor(selector) {
  let bBox;
  $(selector).each(function () {
    const eleBb = this.getBoundingClientRect();
    if (bBox === undefined) {
      bBox = clone(eleBb);
    }
    bBox.top = Math.min(bBox.top, eleBb.top);
    bBox.left = Math.min(bBox.left, eleBb.left);
    bBox.right = Math.max(bBox.right, eleBb.right);
    bBox.bottom = Math.max(bBox.bottom, eleBb.bottom);
  });

  bBox.width = bBox.right - bBox.left;
  bBox.height = bBox.bottom - bBox.top;

  return bBox;
}
```

Some time ago we were doing minor refactoring where we decided to drop unnecessary library functions, including `clone`. So the code above was modified into the following.

```
function getBoundingBoxFor(selector) {
  let bBox;
  $(selector).each(function () {
    const eleBb = this.getBoundingClientRect();
    if (bBox === undefined) {
      bBox = { ...eleBb };
    }
    bBox.top = Math.min(bBox.top, eleBb.top);
    bBox.left = Math.min(bBox.left, eleBb.left);
    bBox.right = Math.max(bBox.right, eleBb.right);
    bBox.bottom = Math.max(bBox.bottom, eleBb.bottom);
  });

  bBox.width = bBox.right - bBox.left;
  bBox.height = bBox.bottom - bBox.top;

  return bBox;
}
```

This modification introduced a bug so a new version wasn't returning a proper value. Can you explain why this happened and suggest a better solution?

This code is available as interactive demo at https://codepen.io/thevasya/pen/WNGwJJQ

## Fixing an issue with customer metadata

This is a simplified version of a function we have in our application code. This function is used to collect metadata on our own customers for internal use.

```
def custom_params
  {
    'Account owner' => "#{owner.name} #{owner.email}",
    'Account status' => account_details.account_status,
    'Active customers' => account_details.active_customer_count,
    'Billing system(s)' => billing_connectors.presence || ['None configured'],
    'Import status' => main_connector.try(:import_status).try(:humanize) || 'Never imp
orted',
    'Last active' => @account.last_active_at,
    'Revenue recognition access' => @account.rev_rec_enabled?,
    'Sample Data Present?' => sample_data_present?
  }.reject { |_k, v| v.blank? }
end
```

Can you spot any issues with this code? What would you suggest to do?

## Programming in Ruby

We have a small data set containing a number of different payment events related to different accounts. This data represents a typical SaaS product where every account can only have a single subscription, `subscription` and `reactivation` events make subscription active, `cancellation` event makes it inactive.

Please write code in Ruby to use this data set and answer the following questions:

- Which accounts have at least one reactivation?
- As of April 2nd, 2020 which accounts have active subscriptions?

```
[
  { "event": "subscription", "account": "A001", "date": "2020-01-15" },
  { "event": "subscription", "account": "B001", "date": "2020-01-23" },
  { "event": "subscription", "account": "C001", "date": "2020-01-25" },

  { "event": "cancellation", "account": "B001", "date": "2020-02-23" },
```

```
  { "event": "reactivation", "account": "B001", "date": "2020-04-01" },
  { "event": "cancellation", "account": "B001", "date": "2020-04-03" },
  { "event": "reactivation", "account": "B001", "date": "2020-05-12" },
  { "event": "cancellation", "account": "B001", "date": "2020-06-18" },

  { "event": "cancellation", "account": "C001", "date": "2020-03-10" },
  { "event": "reactivation", "account": "C001", "date": "2020-05-17" }
]
```

## Design a system to display metrics dashboard

Imagine we're building a new startup which small SaaS businesses will use to get various business metrics from their payment data. For that we need to ingest payment data from multiple payment providers into our system, normalize it, and generate metrics. We have a few metrics like recurring revenue or churn. Customers want to check them regularly and they want to see how they change over time, so we offer a dashboard with different graphs.

Let's design a system like this. We're less interested in what to use for application UI or specifics of integrations with different payment providers. We're more interested in the data flow: what happens with the data from the moment it gets into our system and until user gets their metrics on screen.

*We expect something like a page of text (if you were to print it) for the answer.*