Generating custom identifiers

In this step we are going to replace that non-user friendly id in the URL.

To start off, remove any events that you currently have in your Local Storage to avoid confusion.

The idea behind a solution that I am going to show you is quite simple: upon creating new event we have to assign him a unique identifier in a form of a positive integer that will be stored inside the `position` attribute. Then we will redefine the `idAttribute` property that we've talked about in the previous lesson. This property simply tells Backbone which attribute to use to populate id field.
Modify view like this:

```
this.model.save({
  title: title,
  description: description,
  position: Organizer.EventsList.nextPosition()
}, {
```

So I am using a dynamic value generated by a `nextPosition` function for this new attribute. We have to ensure that position is unique. The easiest way to do this is to check position of a last item in the collection and increment it by one.

```
nextPosition: function() {
  if (_.isUndefined(this.length) || this.length === 0) {
    return 1
  }
  return this.last().get('position') + 1;
}
```

So we are checking if there are any events in our collection. If no – the event we are creating is the first one. Otherwise get position of the last event in the collection and increment it by one.

Now we have to explain Backbone that this `position` should be used instead of a default id attribute. Modify your model:

```
idAttribute: "position"
```

Now tweak the template to render position instead of an id:

```
<script id="event-template" type="text/x-handlebars-template">
  <a href="#" data-position="{{position}}" class="show">{{title}}</a>
  <a href='#' class='btn btn-danger'>remove</a>
</script>
```

Don't forget to change the handler accordingly:

```
showEvent: function(e) {
  e.preventDefault();
  var position = $(e.currentTarget).data('position');
  Organizer.router.navigate("events/" + position, {trigger: true});
}
```

Our router does not require any changes because as long as we've reassigned `idAttribute`, it treats position as the new id.

We want to use `positing` to maintain the order of the elements in the collection. There is a `comparator` function that can be defined for a collection that is used to achieve this task. In the simplest case we can return just the attribute to be used for sorting:

```
comparator: function(event) {
  return event.get('position');
}
```

Reverse the order of the colection

What if you want to reverse the order of the collection? To achieve this, pass two arguments to the `comparator` function, which represents two models. Next return `-1` if the first model should come before the second, `1` if the first model should come after the second or `0` if these models have the same order. In our case we can't end up with a situation where two models have the same order, therefore we return only `1` or `-1`:

```
return event1.get('position') < event2.get('position') ? 1 : -1
```

Our links to the show event page may simplified like this:

```
<a href="#events/{{position}}" class="show">{{title}}</a>
```

So you may comment our or remove these lines:

```
//'click .show': 'showEvent'

//showEvent: function(e) {
//  e.preventDefault();
//  var position = $(e.currentTarget).data('position');
//  Organizer.router.navigate("events/" + position, {trigger: true});
//}
```