Introducing layout

The concept of **layout** is borrowed from Marionette.js framework. It is basically a high-level view that consists of an array of other views. Layout dynamically controls what is being rendered where and in the end we will get rid of the hard-coded structure inside the *index.html*.

Create a layout view

Create a new layout view:

```
Organizer.EventsLayoutView = Backbone.View.extend({

});
```

We want this layout to render two views: a form and a list of events inside the specific blocks. So move the code

```
new Organizer.EventsListView({collection: Organizer.events});
new Organizer.NewEventView();
```

from the router into the `render` function of this layout.

Inside the route instantiate the new layout and pass the collection to it for some more flexibility:

```
new Organizer.EventsLayoutView({collection: Organizer.events});
```

Inside the layout remove the hard-coded collection:

```
new Organizer.EventsListView({collection: this.collection});
```

We also want to control where these views are being rendered so store them in variables:

```
var eventsListView = new Organizer.EventsListView({collection: this.collection});
var newEventView = new Organizer.NewEventView();
```

Now just render and return `this`:

```
render: function() {
  var eventsListView = new Organizer.EventsListView({collection: this.collection});
  var newEventView = new Organizer.NewEventView();

  $('#events-list').append(eventsListView.render().el);
  $('#new-event').append(newEventView.render().el);

  return this;
}
```

We want this layout to be automatically rendered as soon as its instantiated:

```
initialize: function() {
  this.render();
},
```

On contrary, `newEventView` should not be rendered automatically anymore, so remove the `render` function call from `initialize`.

Now remove the hard-coded markup into a separate template that will be rendered by the layout:

```
<script id="index-template" type="text/x-handlebars-template">
  <div class="page-header"><h1>Welcome to Organizer</h1></div>
  <h2>New event</h2>
  <div id="new-event"></div>
  <h2>List of events</h2>
  <div id="events-list"></div>
</script>
```

Add another div:

```
<div id="app" class="container">
  <div id="index"></div>

  <div id="show-event"></div>
</div>
```

Now provide the element's id to the layout inside router:

```
new Organizer.EventsLayoutView({
  collection: Organizer.events,
  el: '#index'
});
```

Now render to that element:

```
render: function() {
  var template = Handlebars.compile($('#index-template').html());
  this.$el.html( template() );

  var eventsListView = new Organizer.EventsListView({collection: this.collection});
  var newEventView = new Organizer.NewEventView();

  var events = this.$('#events-list');
  var new_event = this.$('#new-event');

  events.append(eventsListView.render().el);
  new_event.append(newEventView.render().el);

  this.$el.append(events);
  this.$el.append(new_event);

  return this;
}
```

First of all we take our template and render it. Then we take the nested views and instantiate them. After that inside our template we look for `#events-list` and `#new-event blocks`. Inside those blocks we add contents from the corresponding views.

This works, but the code is still not looking good. We will need to introduce a separate layout for a show event route and that will mean more duplication. Moreover, we still have hard-coded DOM elements and that's not good. In the next step we'll introduce yet another base class and also work with regions.