Adding a new Event Form

At this point we have a list of events being rendered each time the corresponding collection changes. It it time to implement a form to add new events.

First of all, remove the `add` method call inside the *app.js*.

Next create a new form template:

```
<script id="event-form-template" type="text/x-handlebars-template">
    <div class="form-group">
      <label for="event_title">Title</label>
      <input type="text" id="event_title" class="form-control" name="title">
    </div>
    <input type="submit" class="btn btn-primary" value="Add an event">
</script>
```

Rework the markup:

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

Define the new view:

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

});
```

and instantiate it:

```
new Organizer.NewEventView();
```

What we want to happen is to render this view as soon as it was instantiated:

```
Organizer.NewEventView = Backbone.View.extend({
  initialize: function() {
    this.render();
  }
});
```

Now the `render` function:

```
Organizer.NewEventView = Backbone.View.extend({
tagName: 'form',
  initialize: function() {
    this.render();
  },
  render: function() {
    var template = Handlebars.compile($('#event-form-template').html());
    this.$el.html( template() );
    $('#new-event').html(this.el);
    return this;
  }
});
```

Handling Submit Form Event

Now we need to handle the `submit` form event.

```
events: {
  'submit': 'createEvent'
},
```

As you see I write just `submit` because I want this event to be bound to the top-level element which is `form` in our case.

The first thing to do inside the handler is preventing the default action from happening:

```
createEvent: function(e) {
  e.preventDefault();
}
```

Now we have to grab the title of an event and save the model:

```
var title = this.$el.find('#event_title').val();
```

Backbone provides us with a shortcut to simplify this line a bit

```
var title = this.$('#event_title').val();
```

Now instantiate a new model and save it with the provided title:

```
var model = new Organizer.Event();

model.save({
    title: title
})
```

For model saving to actually work we also have to provide `localStorage` attribute for collection because otherwise Backbone won't know where to send the request.

```
localStorage: new Backbone.LocalStorage('events')
```

If you have a back-end server use `url` attribute instead.

Testing in the browser

Now reload the browser and add a new event. No errors in the console, but new item will not be added to the list. This happens because we have not actually updated the collection after saving the model. `save` provides the `success` callback:

```
this.model.save({
  title: title
}, {
  success: function () {
    Organizer.EventsList.add(this.model);
    this.el.reset();
  }
});
```

I've also added one line of code to reset the form's contents using JavaScript's `reset` method. Note that I am not using `$el` here. As you remember `$el` returns a wrapped jQuery set that responds to its own special methods and `reset()` is JavaScript's basic method, not jQuery's. Therefore I use `el` here to call reset on it. You can also write

```
this.$el[0].reset();
```

to turn this wrapped set to a simple JS node.

Lastly introduce `that` variable to overcome scoping issue:

```
var that = this;
[...]

success: function () {
  Organizer.EventsList.add(that.model);
  that.el.reset();
}
```