# Exercises: Single Page Apps with AJAX, REST and Kinvey

Problems for exercise at the [“JavaScript Applications” course @ SoftUni](https://softuni.bg/courses/javascript-applications). Submit your solution in the SoftUni [instance](https://softuni.bg/trainings/1505/js-applications-november-2016) as homework. You can use this [presentation](https://softuni.bg/downloads/svn/js-core/Sept-2016/JS-Apps-Nov-2016/7.%20JS-Apps-SPA-with-AJAX-REST-Kinvey/7.%20JS-Apps-SPA-with-AJAX-REST-Kinvey.pptx) to help you throughout this exercise.

**Exam preparation: creating single-page app (CRUD + login / logout, without framework)**

## Prodavachnik

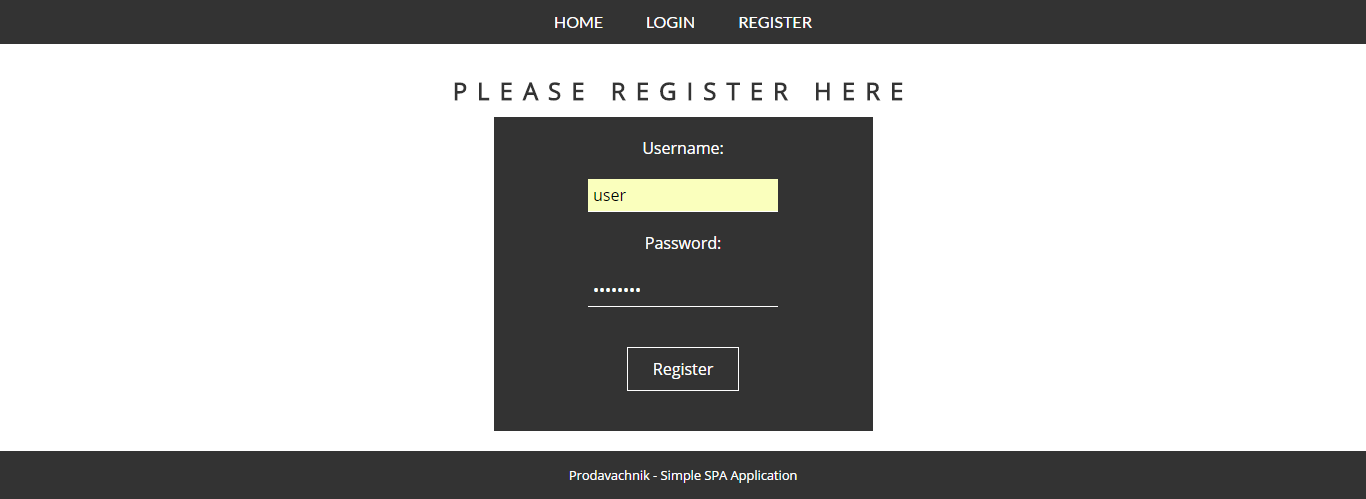
Your task is to implement a Web page that serves the functionally of an **advertisement** **site**, which specializes in **selling** **products**/**services**. The API will consist of creating, listing, editing and deleting of **advertisements**(**CRUD**). Every **advert** should contain the following properties:

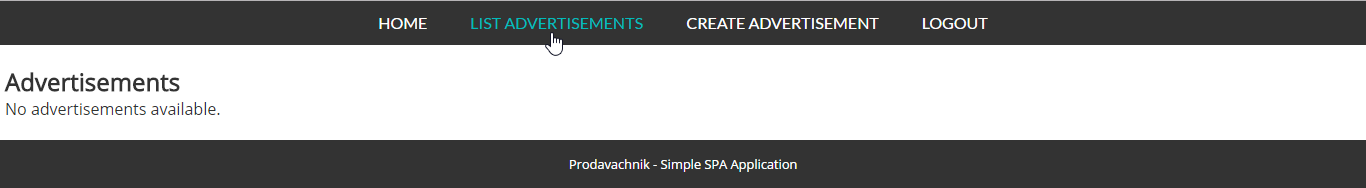
* Title - String
* Description - String
* Publisher - String
* Date of Publishing – Date (format “**mm/dd/yyyy**”)
* Price – **floating** point number, to the **second** **digit** after decimal separator.

You can download the skeleton (HTML and CSS) [here](https://softuni.bg/downloads/svn/js-core/Sept-2016/JS-Apps-Nov-2016/8.%20JS-Apps-SPA-with-AJAX-REST-Kinvey-Exercises/8.%20JS-Apps-SPA-with-AJAX-REST-Kinvey-Exercises.docx).

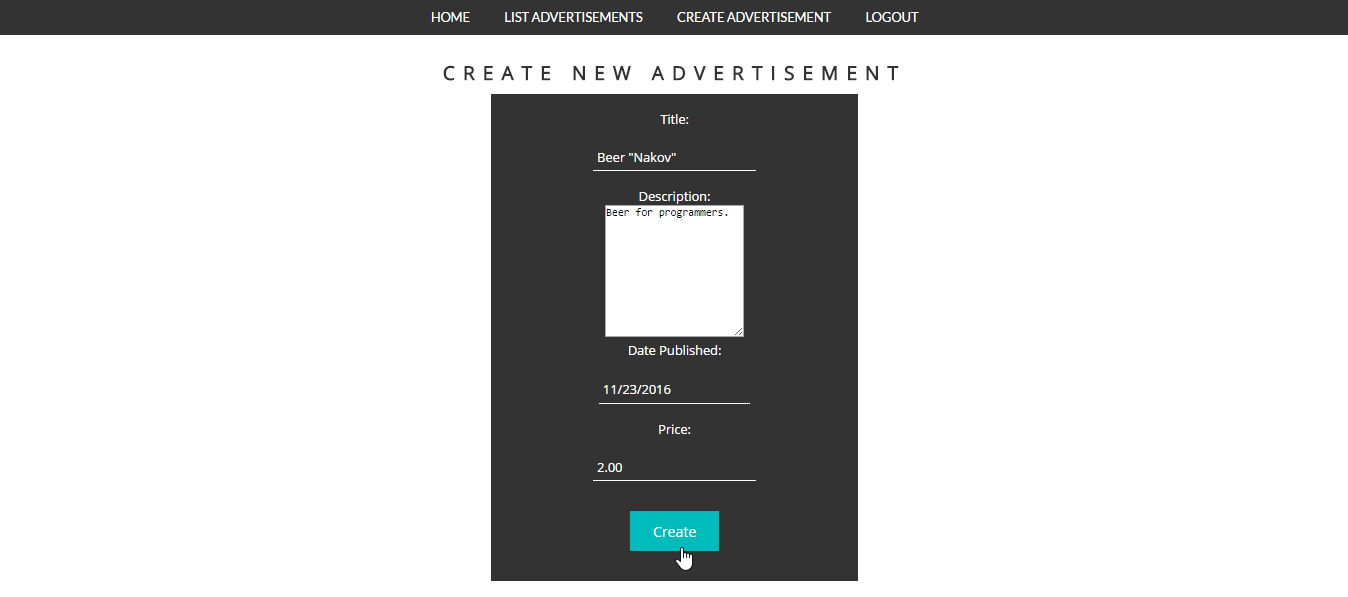
### Screenshots

Home page:

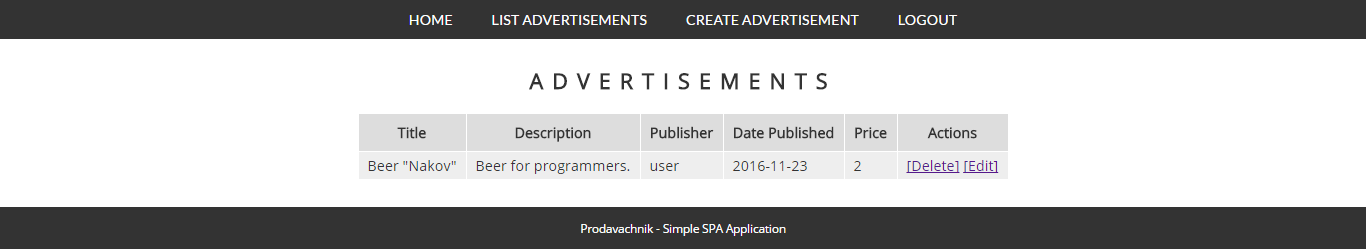
Registration page (Login page is identical):

This is how the website should look like when there are no adverts:

Let’s create one (the view for editing an advert is identical):



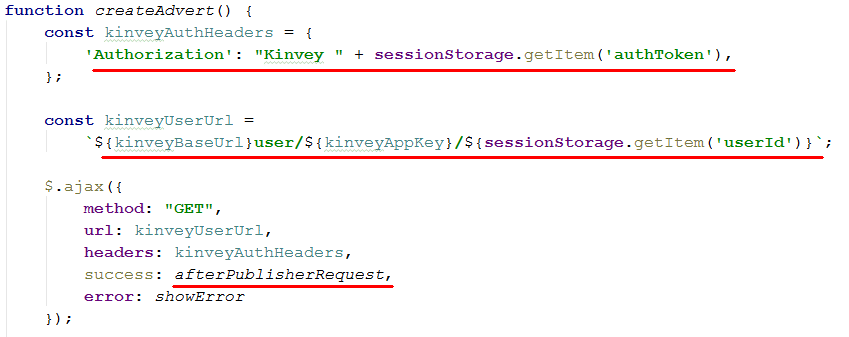
And this is how it should be listed:



The advertisement site should have the following functionality:

* Perform any **CRUD** operations over the advertisement entity.
  + Note that the **publisher** of the advertisement is the **currently** logged **user**.
  + Publisher can edit/delete his own adverts only.
* Have **Register**/**Login**/**Logout**.

### Hints

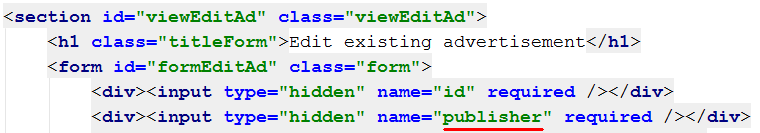
* When **create** an **advertisement** the current logged user should be it’s **publisher**. We can do it by making one request to the database (by user’s **“\_id”** property) and retrieve the whole **user** information. It may look something like that:

Once we got the user information we can use it in our “afterPublisherRequest” method:

And the rest is **parsing** the **information** from the form and sending **another** **request** – this time to **save** our **advert** in database.

**Note** that the above method should be called when the form for **advert** **creation** is **submitted**.

* When **editing** an advert in which we don’t want to change the **publisher** you can include it in the html, but mark it as “hidden”, here is how the html might look:



What is left is to **initially** **set** the **value** when printing the edit form **and** then **parsing** the whole **data** when the form is **submitted**.

## Extend “Prodavachnik” (Optional)

Your task is to extend the functionality of the above project (when completed).

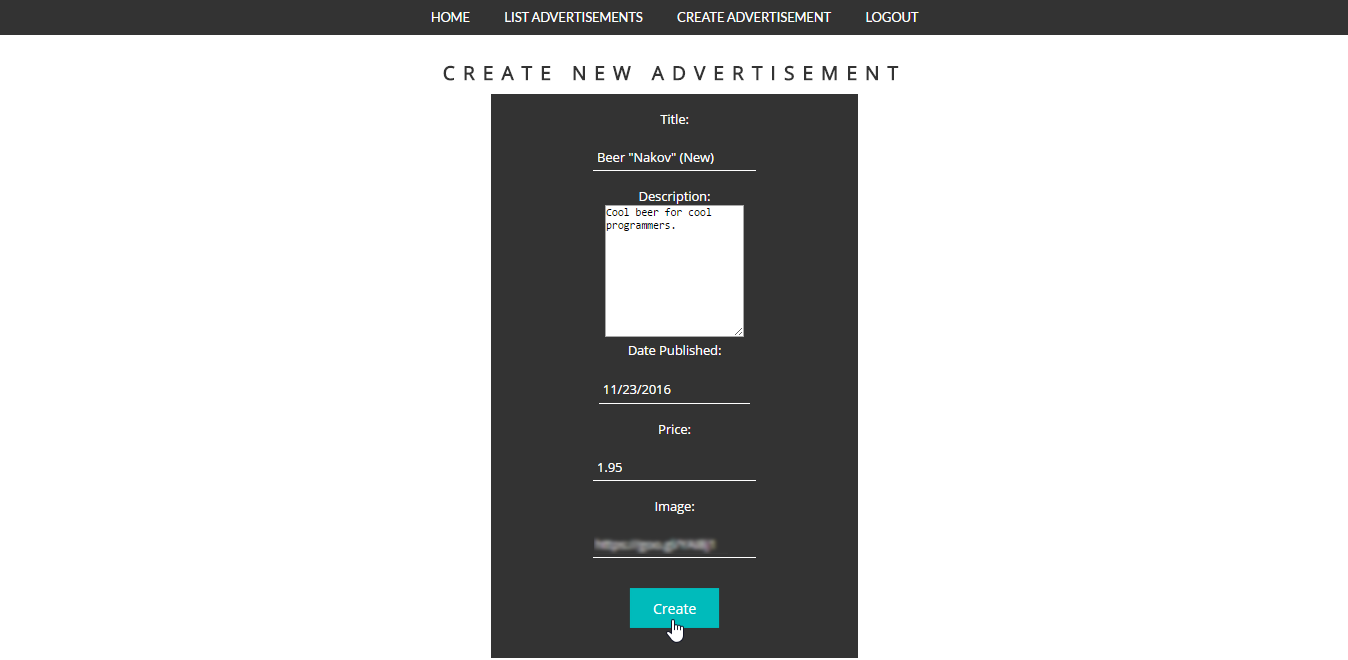
What have to be done is:

1. Creating **one** **more** **view**(“section”), which will be used to **display** one **single** **advertisement**. The view should be accessible when you list all advertisements (by creating another **link** – “**Read more**” e.g.).
2. **Add** another **property** to the advertisements which should be **integer** – the **count** of **views** for every advert. Every advert should have **0** views **initially**. And that count will **grow** when someone tries to **display** it’s **single** view (when the “Read more” link is clicked).
3. Add also another **string** property containing **link** to some **image** (image will be used to display more information about current advert).
4. Whenever you list all the adverts, **sort** them **by** **views** count descending.

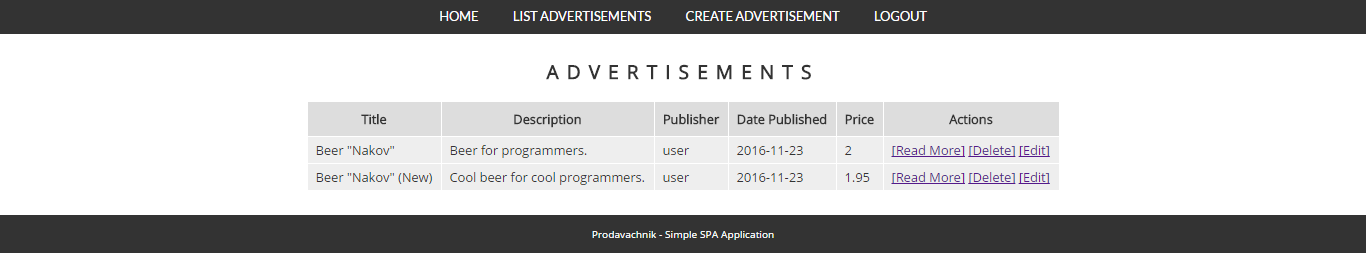
For more clarity check the screenshots below.

### Screenshots

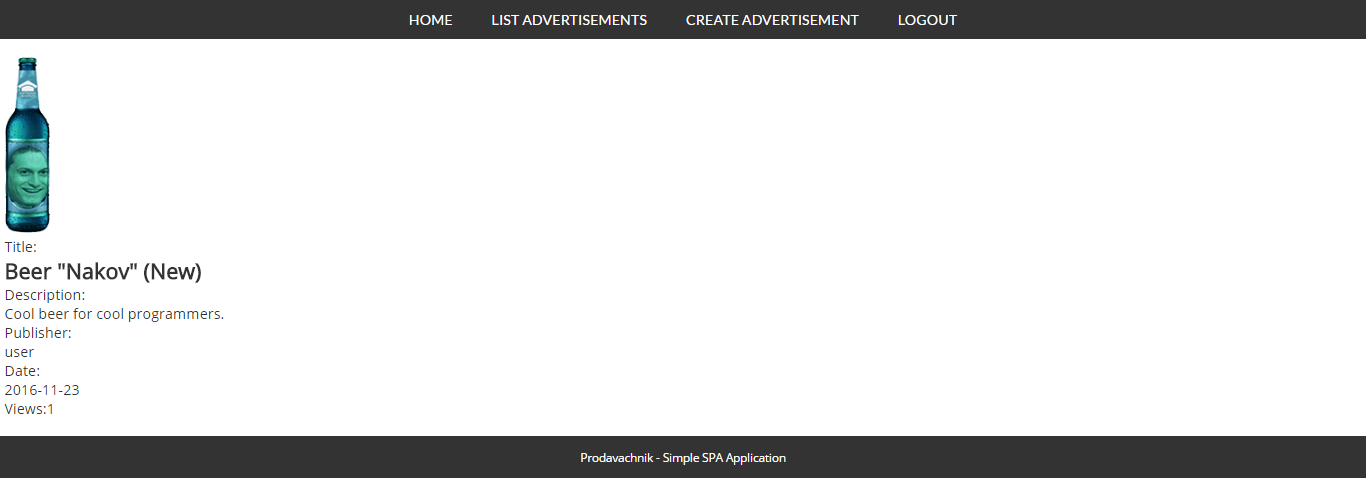
We should extend the create advert functionality. Now we should have **image** field:



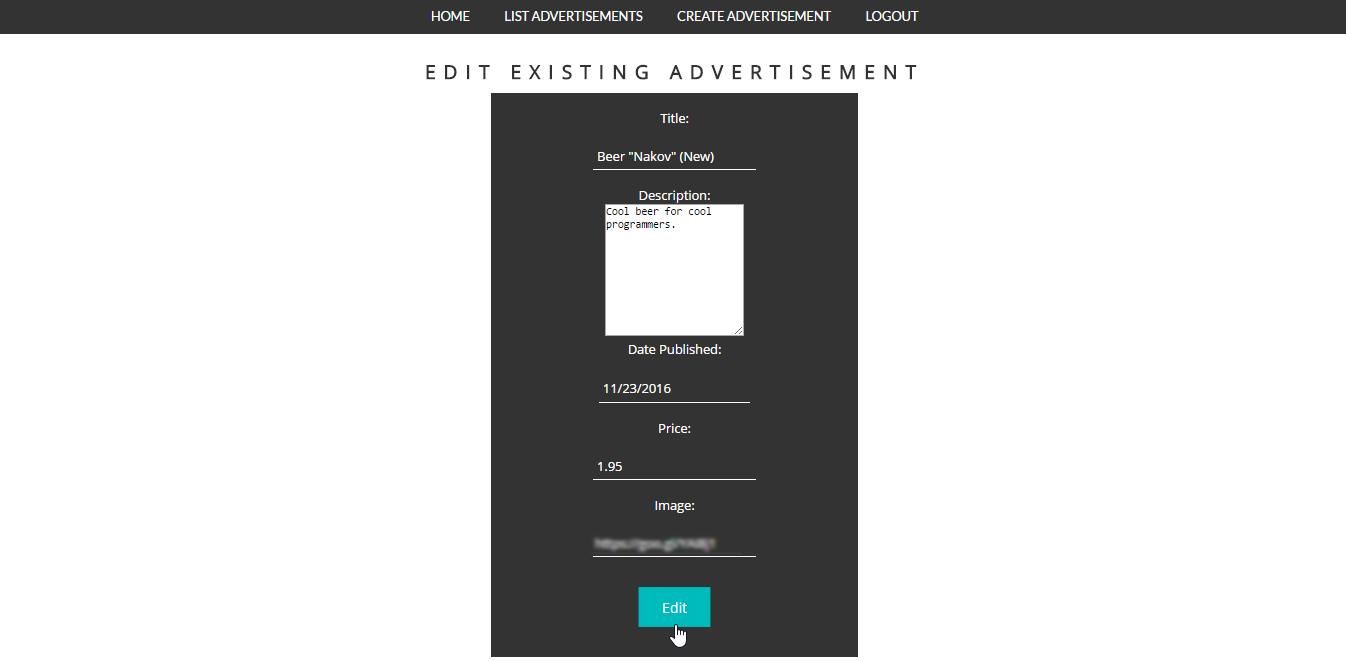
And when all adverts are listed: new “Read more” link:



Cool. Let’s check the “**Read more**” out:



Here we have our **detailed** **view** about particular advert. What we have in addition to the previous one is that we have a **photo** and a **views** counter.

We should be able to **change** the **image** later on (if we want to):

### Hints

* For the details view you can just add another section and give it some id, so you can later on refer it.
* Then on the part where **listing** of adverts is done, just add another action(<a> tag) – next to **delete** and **edit**:

**Create** the “Read more” **link** and attach action:

Add it to all links:



* The action should **render** some **html** – nothing special: some html tags with some content in it. Here is an **example** of how it might look like:

**\*Remember** later on when you add the new properties to include them here as well.

* As far as for the **count** of **views** per advert you can do asynchronous **request** to the database **updating** the advert’s **views** count everytime someone clicks on the “Read more” (aka when the **upper method** is **called**).
* Editing an advert should now get the **views** **count** from the html (another “**hidden**” input filed) and also change the **image** url for the advert.

Note that the variables’ names might differ from the ones in your project.