Interesting cameras (test task)

It is required to develop a single page web application that allows View the list of public cameras connected to Ivideon and add them to the Favorites.

General requirements

The task is accepted as a zip archive containing all the source files. In the archive there should also be a README text file describing how to run the result the task.

If you would like to add something beyond the task, or something to improve in the proposed implementation, but you do not have enough time - just write us about it.

Presentation Requirements

The dimensions of the elements on the page must proportionally adapt to the width browser window, the maximum width of the page is 600px.

Technology requirements

JavaScript

ReactJS, Redux.

You can’t use languages, that are compiled in JavaScript (CoffeeScript, TypeScript, etc.). The use of ES2015 + is welcome.

CSS

You can use any methodologies, but you can’t use ready-made CSS frameworks.

Architecture Requirements

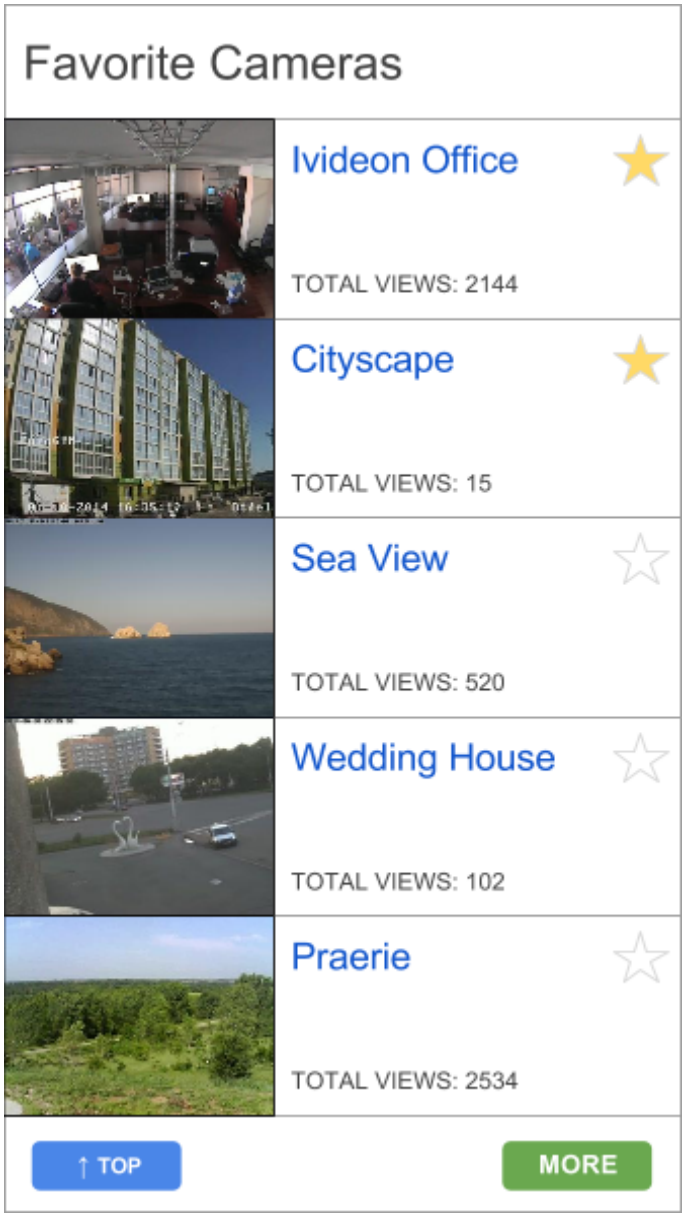
The application architecture should allow you to easily extend it (for example, add New screens, change the behavior of controls).

Imagine that the components you develop are part of a large application (Potentially reusable). Tip: The standard boilerplate for Redux is not Scale to large applications.

Functionality requirements

The application should consist of one screen with a vertical list of cameras.

The layout is presented below:



The list should display public cameras received through the open API Ivideon. The documentation for the API for obtaining a list of cameras is below.

For each camera, the preview image should be displayed in the list. Viewing ratio of 4: 3. It should be noted that Ivideon has connected cameras with Different resolution and aspect ratio of the frame. API documentation for receiving Images - below.

For each camera, the camera name and total viewed number should be displayed in the list.

Add to favorites

For each camera, a star must be displayed in the list, which when pressed "Adds the camera to favorites" (or, if the camera is added, deletes from favorites). Adding to favorites should lead to the fact that the camera after a page reboot will not disappear from the list and will appear at the top.

«TOP» button

Under the camera list, the "TOP" button should be located, which scrolls Page to the title.

«More» button

Under the camera list, the "MORE" button should be located, which loads the new camera list and adds them to the end of the current list.

Camera view

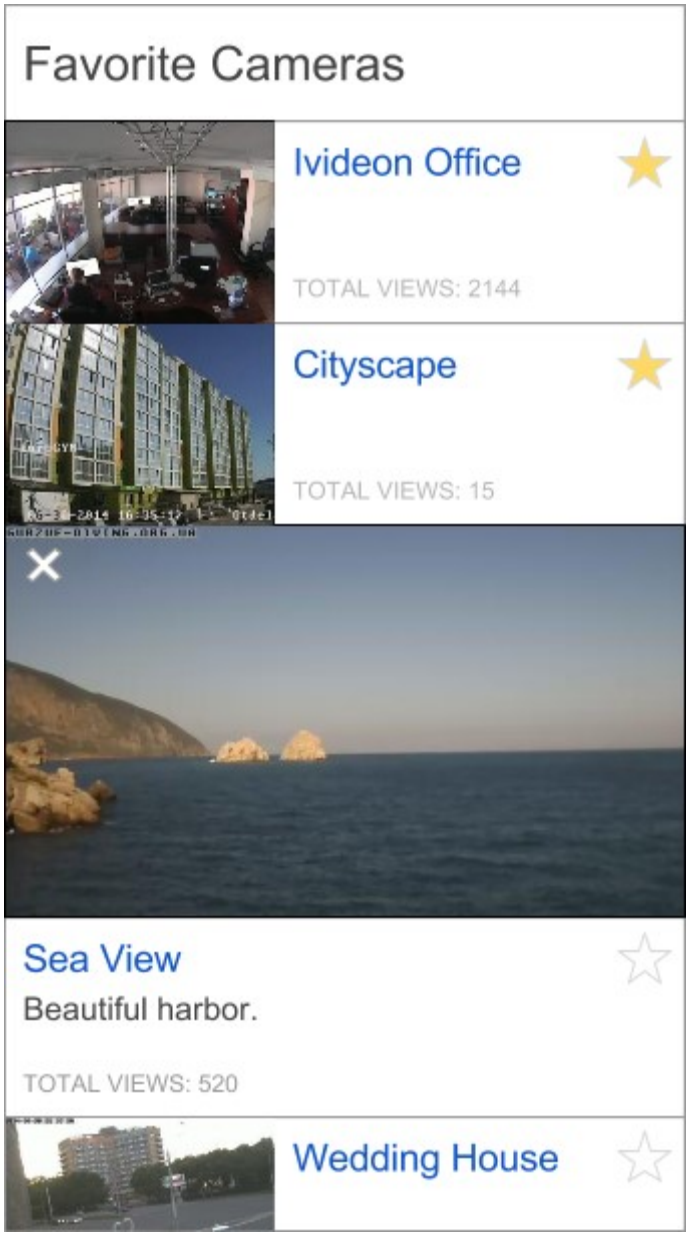
When you click on the camera in list, the preview image should turn around in the full width of the list. At the same time, an image must be displayed In its original proportions (even if it was cut off on the list).

The name, number of views and asterisk should be under the image. A description of the camera should appear between the title and the number of views.

The open image should be updated every 10 seconds. Optinal: updated Image make a smooth transition (crossfade) from old to new.

On the open image should appear a cross, when you click on it, camera view is returned to the collapsed state.

The layout is presented below:

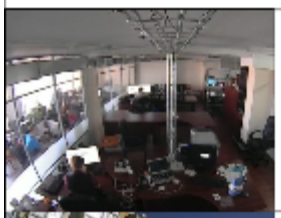


If the image fails to load, instead of the image, display the black background, message that the camera is not available, and "UPDATE" button.

When you click on the "UPDATE" button, you should try to download the image again.

The layout is presented below.

Favorite Cameras



Ivideon Office

TOTAL VIEWS: 2144

Cityscape

TOTAL VIEWS: 15

Camera is offline.

REFRESH

Random Camera

Shows random images.

TOTAL VIEWS: 10

API to get a list of random public online cameras

The documentation below shows only those parameters and data that are necessary for the task.

Since the API of the list of public cameras produces real-world cameras of users, availability of which depends on their owners, the returned list may vary with time. For the same reason, the API does not guarantee the complete absence of duplicates when successive requests - the answers may include cameras that have already been returned earlier.

Request	`GET http://api.ivideon.com/tv/cameras?jsonp={callback}&limit={limit}&seed={seed}`
Params	jsonp — the name of the JavaScript function that takes a JS object with response. limit — maximum number of cameras in the returned list. Optional. seed — page identifier for page navigation. Optional. To download the following cameras, as a `seed` param value pass `response.seeds.next` from the response to the previous request.

Response	<pre>{
	    // Success flag (false on error):
	    "success": true,
	    // response body (contains error details on exception):
	    "response": {
	        // camera list:
	        "cameras": [
	            {
	                // server id :
	                "server": "...",
	                // camera id :
	                "camera": "...",
	                // camera name :
	                "camera_name": "Cool Public Camera",
	                "misc": {
	                    // camera description:
	                    "description": "Yeah, Ivideon is cool!",
	                    // ...
	                },
	                // total count camera viewed :
	                "total_views": 2144,
	                // original camera image widt:
	                "width": 1280,</pre>

	<pre>// original camera image height:
	"height": 720,
	// ...
	},
	// ...
	],
	// pages ids:
	"seeds": {
	// ID of current page:
	"this": "68935489",
	// ID of next page:
	"next": "68935499"
	}
	}
	}</pre>

API for previewing images

The documentation below shows only those parameters and data that are necessary for the task.

Request	`GET https://streaming.ivideon.com/preview/live?server={server_id}&camera={camera_id}`
params	server — server ID. camera — camera ID.
Response	*JPEG image*