ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Institute of Information Systems
Prof. Moira C. Norrie
Maria Husmann, Linda Di Geronimo, Alfonso Murolo, Amir E. Sarabadani Tafreshi

globis

Web Engineering

Spring Term 2015

Exercise 5

In this exercise your task is to build a gallery application that uses real-time communication between multiple devices. The application consists of two parts. The first is intended to be loaded onto large screens and will only display a single image. The seconded part is designed for mobile devices and displays a set of images. The user can select an image which will be shown on the large screen. The mobile device thus functions as a remote control to the large screen. You can find a video on the course website that demonstrates the desired behaviour.

Technologies

In this exercise you will use NodeJs[1] and Socket.io[2]. If you have never used Socket.io, we recommend that you do the chat tutorial[3], before you do the exercise. You can download a skeleton application on the course website[4].

Exercise 5 has only one part and consists of the following steps.

Install the skeleton app

Install NodeJs. Download the skeleton application from the course website. Open a console in the application folder and run *npm install*. Then you can start the server by typing *node index.js*. *index.js* is the server-side code that you need to edit. The client can be found in the *public* folder and consists of HTML, JavaScript, and CSS files for screens and remotes which are named accordingly. After starting the server you can access the screen by pointing a browser to *http://yourip:8080/screen.html?name=devicename*. You can choose a value for the parameter *devicename* which will be used later to identify the screen. The remote is available at *http://yourip:8080/remote.html*.

Integrate Socket.io

Install Socket.io and integrate it both in the server and the client. Make sure that dependency is configured correctly in the file *package.json*.

Connecting

Connect both the remote and the screen to the server using Socket.io.

Synchronise the current image

When a image is selected on the remote, display the chosen image on the screen.

Advanced connection management

So far we have only considered one remote control and one screen and implicitly connected the two. However, in a realistic scenario there could be multiple instances of remotes and screens. Show a list of available screens on each remote and let the user choose a screen to connect to. Screens should only react to commands from connected remotes. A remote can control multiple screens at a time. It is not required that you implement any conflict management for the case where multiple remotes try to control the same screen. It should also be clear from your list which screens a remote is connected to and allow disconnecting from these. If a screen is disconnected by the remote, the screen should be cleared (it should show no image). If a screen disconnects from the system (e.g. the browser windows is closed), it should be removed from the list. If a remote is removed from the system, all screens that were connected to this remote should also be cleared. Please have a look at the video which illustrates this behaviour.

[1] `https://nodejs.org/`
[2] `http://socket.io/`
[3] `http://socket.io/get-started/chat/`
[4] `https://globis.ethz.ch/#!/course/web-engineering/`

Workmode and Grading This is the first and only part of Exercise 5. **The final team solution for Exercise 5 will need to be presented during the exercise session on 21.05.2015 from 12:15 to 14:00**. You do not need to hand in the source code, but you need to be prepared to show it and the assistants may ask questions about how you implemented some of the features when you demo your solution in the exercise session. **We will use the following grading scheme.**

Grading Scheme For each part of the graded exercises, we specify the amount of points your team can achieve and a set of requirements. These requirements represent the minimal set of goals you need to accomplish to get full points. If you fulfill the requirements only partially or fail to answer corresponding questions during the presentation, points will be deducted. No or wrong solutions get zero points.

The maximum number of points may differ between parts and thus reflect their respective weights. For Exercises 5, the set of requirements are specified as follows:

Set of Requirements - (Max. points: 4)

- **Dependency management** Package.json should be configured correctly to include Socket.io. If the *node_modules* folder is deleted, running *npm install* should install Socket.io.
- **Image synchronisation** Selecting an image on the remote should update screens accordingly.
- **Advanced connection management** The remote should list the available screens in the system. New screens should be added to the list in real-time and without user interaction (no refresh buttons!). The list should indicate for each screen if the remote is connected to it or not. The remote should be able to connect to and disconnect from screens, depending on the current status. Image updates should only be sent to connected screens. Disconnecting a screen should clear the image from the screen. If a screen leaves the system it should be removed from the list on all remotes. If a remote leaves the systems, all screens that were controlled by the remote should be cleared.

Please note that this exercise can be solved using only Socket.io and JavaScript. However, you are allowed to use frameworks such as jQuery or CSS frameworks if you consider them helpful.