Lab 4: Interaction in the web

Goals

- practice layouting in the web
- practice dynamically loading/changing of the views
- practice MVC paradigm with JavaScript

Assignment

Through lab 2 and 3 you have learned how to implement separate view (layout), controller (behavior) and model (data) in the desktop/mobile applications. MVC paradigm can be applied to the web as well and many popular server-side development fameworks (like Ruby on Rails, Play, etc.) are based on it.

In this lab you will be working only with the client side development meaning you will be working with HTML and JavaScript to achieve and interactive website. Modern web applications (gmail, google docs) try to achive as much as possible on the client side becuase that provides better user experience (not needing to wait for the server response and page reload).

Your assignment will be to implement the layout of the provided prototype and dynamically (via JavaScript) load different layouts and achieve interaction through MVC pattern.

How

For the web version of the Dinner Planner we have selected solution submitted by Group 26 (and simplified it a bit).

You can use the commentiong option on the balsamiq site to ask questions and clarifications on the prototype.

Step 0. Setup the development environment

First get the starting by:

- using git to clone it from the repository
- **OR** download it as a zip archive directly

When you unzip the code you will find following content:

- index.html - the only HTML page you will have in this project (though while testing you can create more to make it easier). You will need to implement the skeleton layout there and then through JavaScript load the rest
- js/model/dinnerModel.js - is JavaScript file that contains the model code. You will not need to change anything here, but you will need to check the code to see what functions are available to you. The model already implements the observable pattern (check the bottom of the file). In general this model is similar to the models for desktop/mobile just adapted to JavaScript language (so no interfaces etc.).
- js/view/ - here you can find a JavaScript code of a example view and its controller. The view sets up some initial components and their values, implements update function (from observer pattern) and registers itself as observer to the dinner model. The controller has click listeners for the buttons in the example view.
- js/app.js - this is the overall code of the application. It is responsible for inital setup of the app (when the page loads for the first time) and for navigation logic (showing and hiding the views).

To work with the code you can use any plain text editor (my favorite is SublimeText, but there also examples like Kate for Linux, TextMate for Mac and Notepad++ for Windows) or HTML editor.

You can test the source code by running the index.html file in your browser. You should see a total price, number of guests and two buttons for increassing and decreasing the number of guest.

Step 1. Implementing the layout and different views

Your first development task is to implement the 5 different screens as closelly as possible following the prototype. We suggest taking the advantage of Bootstrap framework for good layouting and styling solutions. The framework is already included in the index.html, so you just need to check the documentation and start using the examples.

In the prototype you will notice that there are a minimu of 6 different veiws (more if you consider single dish in the lists as a separate view) in 5 different screens. All the HTML code for the views needs to be in **one HTML file** and you will just use JavaScript to populate and generate the code that you need. However, for now you can, just for easier development and division of the work, create each screen in a separate file.

Step 2. Populating the views with the data

You can take a exampleView.js as an example of how the view JavaScript code should look like. In the beginning of the view constructor function you need to get the relevant objects (by their respective IDs) or you create them (demonstration of both you can see in the exampleView).

Typically, the componentes that are static (their number doesn't change) you create in the HTML and just load them using container.find() method. The dynamic components (like dishes or ingrediants) in different lists you will need to create through code.

After you create or get the elements, you need to load the data from the model. You will see that in the exampleView you already have the models so you can just access the information you need. For example:

```
model.getDishes("starter"); //gets all the starters
model.getNumberOfGuests(); //gets the number of guest
```

At the end of the exampleView you will see a code that registers the view with the model as an observer and the **update** function. In your views you need to make sure that the update function changes the values of the elements that are affected by changes in the model.

Step 3. Implementing the controllers (interaction)

Finally you will need to add the interaction by implementing the controllers for your views. In the controller you need to pass the view and the models. Then you use the passed view to access its element that you want to add some intaraction too (and that you stored in the object variable in the pervious step) and attach the listener. In the listener then you do what is expected (change the model, load next view, etc.).

For example in the exampleViewController you can see how listeners are added to two buttons and how they change the model.

To completelly handle the navigation (showing/hiding the views) you might need some overal logic for the application that will keep a track of your views and switch between them. You can use the app.js for that. It is already included in the index.html and it is responsible for creating the exampleView and controller. A global variable called the variables and functions you declare on "widnow" object (like wit the example stage variable) will be available in your other controllers, so you can access them in your listeners.

Uploading the results

When you have finished with developing either mobile or desktop application, just push your changes to GitHub and submit the repository link in the comment of **Bilda assignment** Lab 4: Web interaction.

If you didn't use GitHub, **zip** your whole project folder and upload it to the Bilda assignment.

Show earlier events (2) ›

Teacher Filip Kis changed the permissions | Friday 18:50

Kan därmed läsas av studerande och lärare och ändras av lärare.

... or write a new post