Lab assignment 4

TI1506 2017/18; ti1506-ewi@tudelft.nl

Introduction

In the first part of the assignment, you will complete the first prototype of your Web application, by adding CSS.

In the second part of this assignment you will reflect on the design of your application from the perspective of the data managed by it, and produce an updated application and/or database design.

The third part of the assignment will require you to make another step towards a fully functional Web application: you will improve the node.js solution created in the previous assignment with the code required to retrieve data from the database.

The last two parts build on the database-enabled version of your Habit application. Your goal is to add analytics[1] functionalities by creating a data dashboard. Given that your application persistently stores in the database all the created/completed habits, your objective is to make use of the full power of SQL to provide users with insights about their usage behavior.

Deadline: when this assignment is due depends on the cluster your team is in. Every team is assessed biweekly. Make sure you know your cluster and what this means for your lab deadlines. All of this information is available in the course syllabus (on blackboard).

In order to pass the assessment, you have to make a valid attempt at working through the lab assignment.

[1] http://en.wikipedia.org/wiki/Analytics

1. CSS

Now that we (finally) covered CSS, you will continue to work on your two HTML documents (the splash screen and habit tracker page) and style them with CSS. To ensure that your CSS is of high quality, you can make use of CSS Lint: http://csslint.net/.

Choose three browsers (e.g. the latest versions of Safari, Google Chrome and Microsoft Edge) on which to test your Web application - the application should look and run in the same manner across those three browsers.

You are not allowed to use external libraries or preprocessing tools. Your application should have a modern look and feel (i.e. use sufficient CSS styling).

1.1. Splash screen

First, work on your splash screen and style the page with CSS according to your design. To ensure that every student learns the basics of CSS, we provide a list of **must-have** CSS properties. Your code must include at least one instance of each of the following:

- Pseudo-classes `:hover :active`
- Pseudo-elements: `::after ::before`
- Box model: margin, padding, border
- At least two different position attributes, e.g. `position:relative` and `position:absolute`.
- At least one CSS animation

You are of course welcome to use more CSS properties to style your splash screen. Once you have completed your CSS implementation, make a **screenshot** and add your implemented design to **your Brightspace discussion forum thread** (where you already posted your initial design). Does your implementation deviate significantly from your initial design? Write a **paragraph** comparing the two.

1.2. Habit tracker page

Finally, work on your habit tracker page and implement your design in CSS. The look of the habits page should be coherent with the splash screen. You will be able to reuse some of the CSS written for the splash screen - make sure that you are efficient and do not duplicate existing CSS code if possible.

Your design must visually distinguish between:

- habits that are **on track** and habits whose desired and actual frequency **differ substantially** (e.g. a positive habit with a daily goal of execution that is executed once a week could be flagged here). You can decide what "substantially" means to you.

- **positive and negative** habits (if you have those in your design), and

- habits that have been **executed** vs those **still due** in the desired time interval (e.g. a weekly habit that has not been completed this week should visually differ from one that has).

Once you have completed your CSS implementation, make a **screenshot** and add your implemented design to **your Brightspace discussion forum thread** (where you already posted your initial design). Does your implementation deviate significantly from your initial design? Write a **paragraph** comparing the two.

Note that designs can change and if your implemented design differs considerably from the envisioned one this is not a problem. It will not cause an issue during the assessment of your work - just be prepared to discus the reasons for your change in design.

Deliverable: updated project source code with CSS now included.

2. Compare the Schema of the `Habit` Database with your Design

Critically analyze the differences existing between the provided database schema and the in-memory data structure you created during the server-side JavaScript part of Lab Assignment 3.

1. If the database schema contains more concepts (tables), attributes, and relationships than your current implementation, then think about how you should modify your application accordingly.

2. If your application provides a richer data model, propose a modification of the database by extending the drawing of **deliverable 6** of Lab Assignment 2 (the visual representation of the schema)

Deliverable: An updated version of the application design (either as a mockup or as a modified HTML template) which includes parts of the data model not currently included. Also, an updated version of the database schema (as a drawing), if your design/application included additional data. In both cases, you should be able to clearly explain the motivations behind the changes in the design/database schema.

3. Database Interaction with node.js

Improve the `node.js` script you wrote in Lab Assignment 3. Instead of keeping a list of habits in the server's memory (which only works well if the server does not crash or runs out of memory), we will now use a database to store/retrieve/update our habits.

Change your `node.js` script so that the server now:

1. retrieves the habits from the database upon a client's request
2. adds a new habit to the database, upon a client's submission of a new habit
3. modifies an existing habit in the database, upon a client's submission of changes related to an existing habit
4. deletes an existing habit from the database, upon a client's request for deletion

In the basic version of this exercise, the User and the Habit_List can be fixed (i.e. they are constant selection conditions in the SQL query that retrieves Habits).

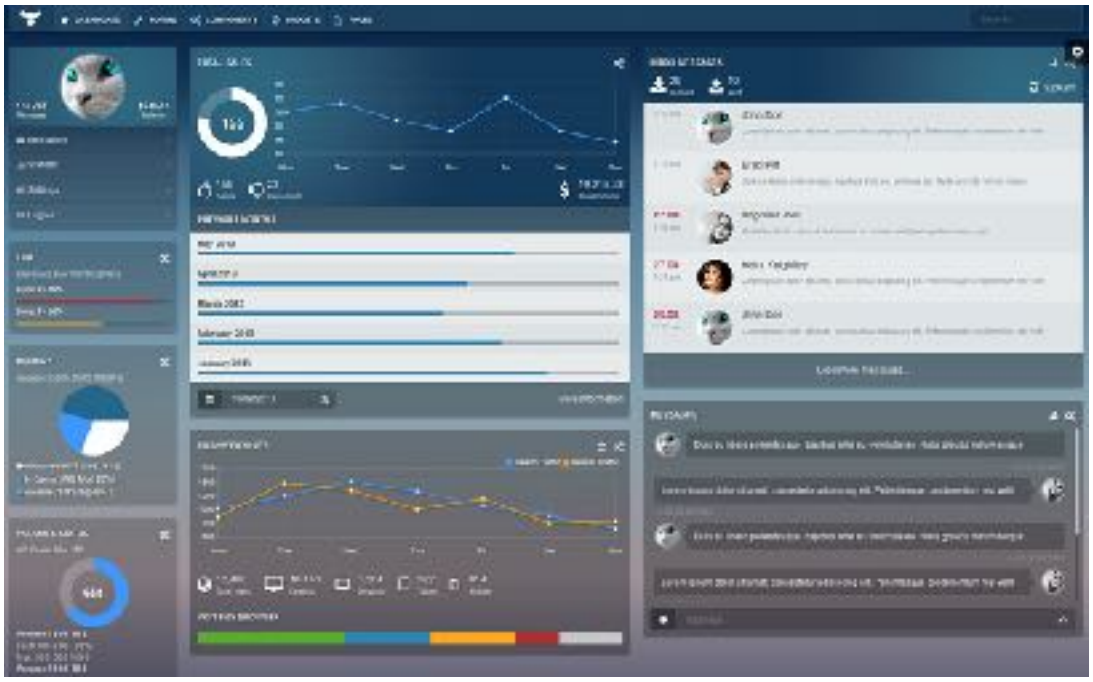
Deliverable: Updated server-side code of your Web application, running in your machine.

Additional optional exercise: expanding your code to support multiple users and multiple Habit_List.

4. Extension of Server Side Application With Analytics Functionalities

Improve your node.js application by adding a new new *route* (i.e `app.get("/$URL$", function (req, res) { ... })`) to serve a new analytics *Dashboard* HTML page. Here is an example of dashboard page you can use as an inspiration for your dashboard "look&feel". The page contains several "widgets", each devoted to the visualization of some aggregated data. An example widget is n histogram showing the average number of habit completions for a habit list (query 12 of **Assignment 3, Exercise 3**).

You will be asked to create the actual HTML page in exercise 5. In this exercise, the goal is to create the server-side functionality that will allow you to feed data to the dashboard page.



For each of the queries in **Assignment 3, Exercise 3**, create a new *route* (i.e `app.get("/$URL$", function (req, res) {…})`) that implements the logic required to execute the query on the database. Each new route must return a valid JSON response, to be later used by the *Dashboard* HTML page.

Deliverable: Updated server-side code of your Web application, running in your machine.

5. Extension of Client-Side Application with New Analytics

Create the new *Dashboard* HTML page. The page must make use of dynamic Javascript code to:

1. Asynchronously (i.e. using AJAX requests) invoke the new endpoints created in 4)
2. Update the content of the page with the results retrieved from the server.

Visualizing data in a textual form is an acceptable solution, although the usage of visual widgets is preferred.

Deliverable: Updated client-side code of your Web application, running in your machine.