SurveyBuilder

Report/Documentation

The University of Edinburgh

IAESTE Summer Internship 2015

by Piotr Hajduk

Edinburgh, 14.08.2015

Contents

Introduction

This document is a technical documentation of the “SurveyBuilder – Survey Management System”. This is an application, which has taken inspiration from the https://www.surveymonkey.com/ and was implemented in Links programming language. The application has been developed by Piotr Hajduk during the summer internship in 2015 at the University of Edinburgh. The project has been supervised by: James Cheney, Sam Lindley and Garrett Morris.

Installation

To run the application you should have at first installed Links and PostgreSQL DBMS (or any other DBMS supported by Links). To do that you should follow the instruction from: https://github.com/links-lang/links/blob/sessions/INSTALL. Make sure that prelude.links file contains *textareacss* function. Afterwards you need to download “SurveyBuilder” source code from the repository: https://github.com/links-lang/links-examples/tree/master/surveybuilder. Then (supposing that you use Postgresql DBMS with installed pgAdmin III) you should run pgAdmin III, open a Query tool (“Ctrl + E” or “Tools -> Query tool”) and execute the query:

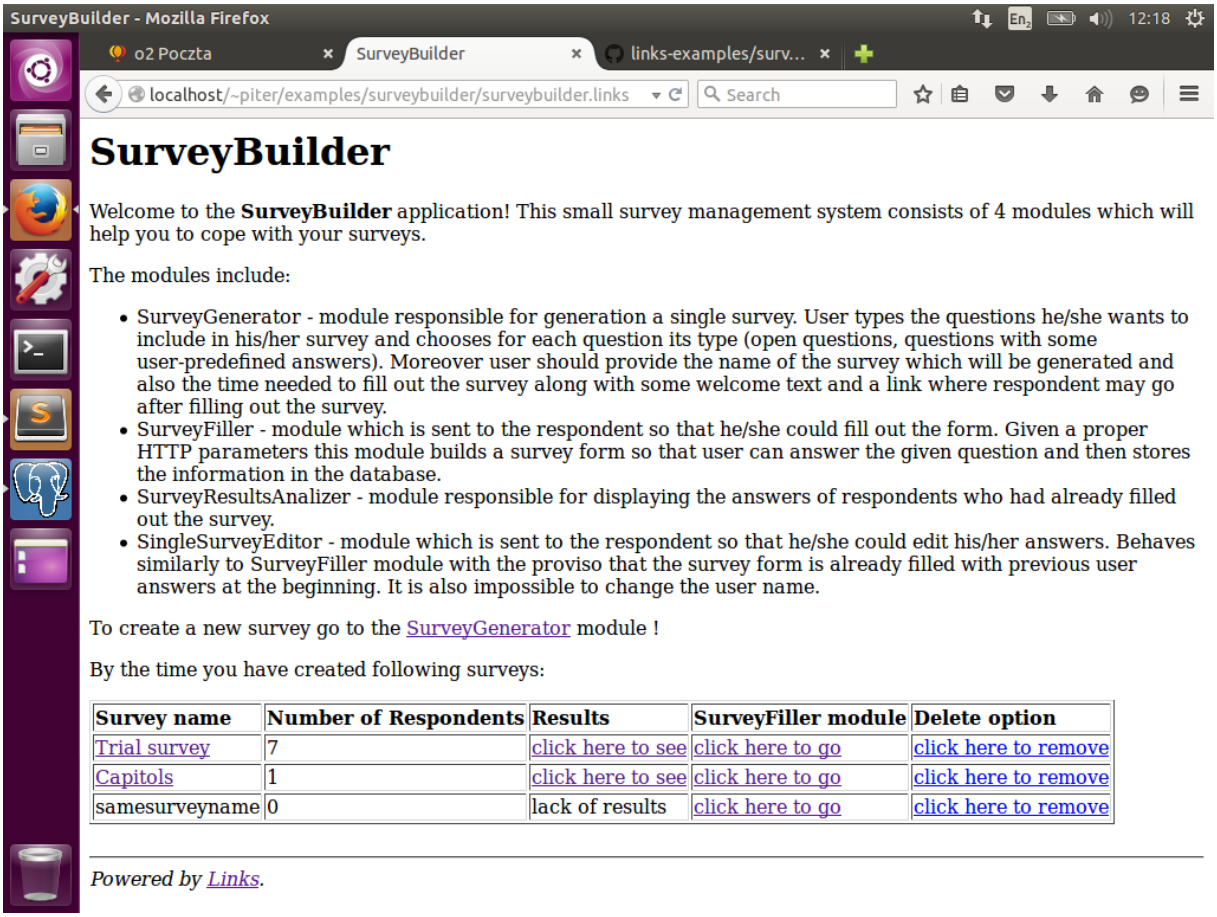
```
create database surveybuilder;
```

This command will create you a database within which you will have to create all the tables connected with SurveyBuilder applications. You will find this command in surveybuilder.sql file together with all the other queries that will create for you the whole database schema. Make sure that you created successfully “surveybuilder” database and you are now operating within its scope and then just copy paste and execute remaining queries from surveybuilder.sql file that will create for you all the tables you need (all starting with CREATE TABLE).

Now all you need to start working with SurveyBuilder is running surveybuilder.links file (the best in the browser).

“SurveyBuilder” Survey Management System

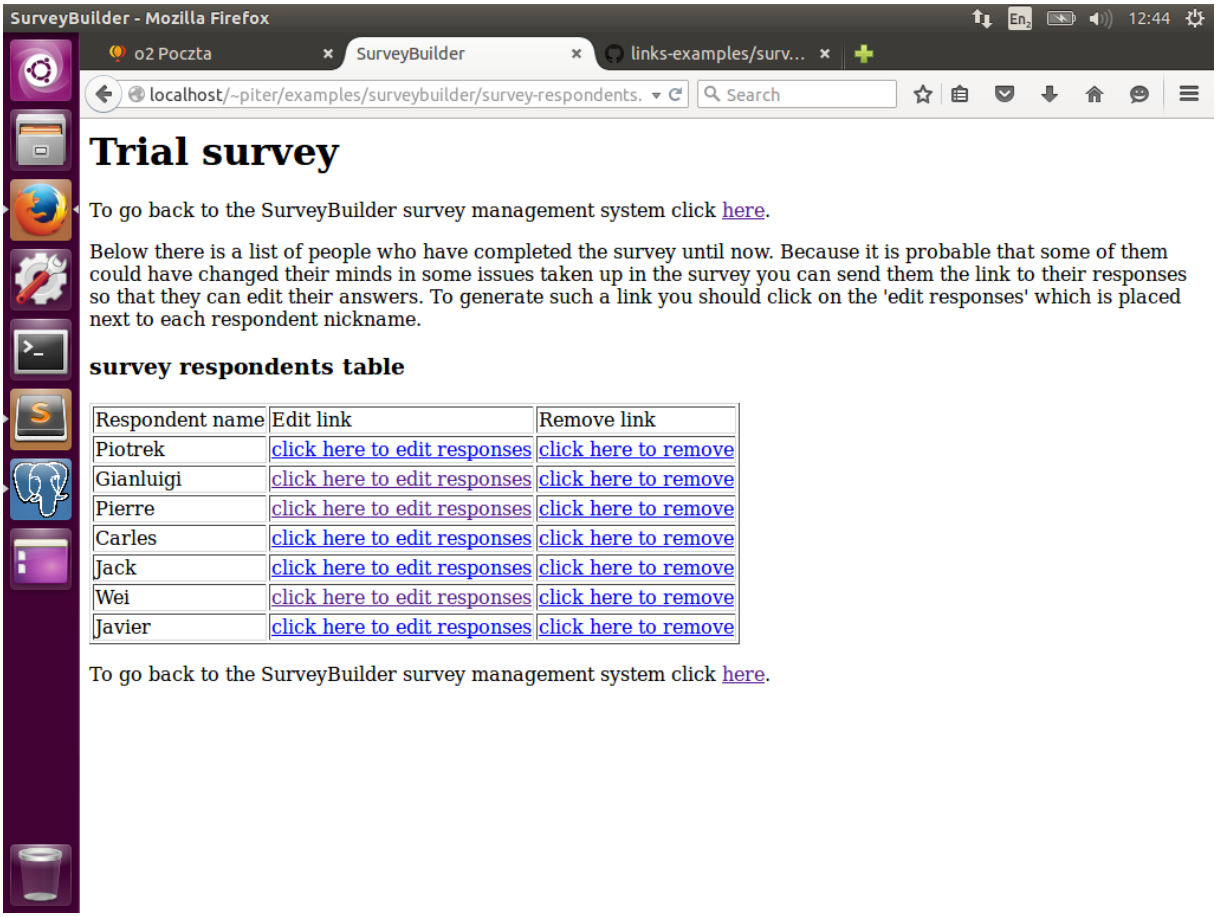
As I have already stated in the introduction this application is to a certain extent similar in presumptions to SurveyMonkey. So it enables the user to generate the survey, save it, then send the link with the survey to respondents so that they can fill it out. Furthermore author of the surveys can see the up-to-date summary of the results, the answers of the single respondent and in case some respondent would like to change his answers he can just send him a proper link with the survey and his current answers so that later his responses will be updated (with no need to create an entrance for the same user in the database twice). Obviously it is also possible to remove the survey when we do not need it anymore or the entries of a single respondent when we find it unnecessary. The main page of an application looks like that (the source code is stored in the surveybuilder.links file) :



Basically there is some introduction text at the top, the reference to SurveyGenerator module which will let user start generating the new survey and the table with already generated surveys at the bottom.

In this table we see (starting from the left) the name of the survey, the number of people who had already filled it out, the link to the module where the results of the survey will be displayed, the link to the module which should be sent to the potential respondent so that he/she could fill it out and at the end to link which after clicking on will remove the survey (from the list and database). As someone can observe the survey name and results in case of survey filled out by nobody is unlinked.

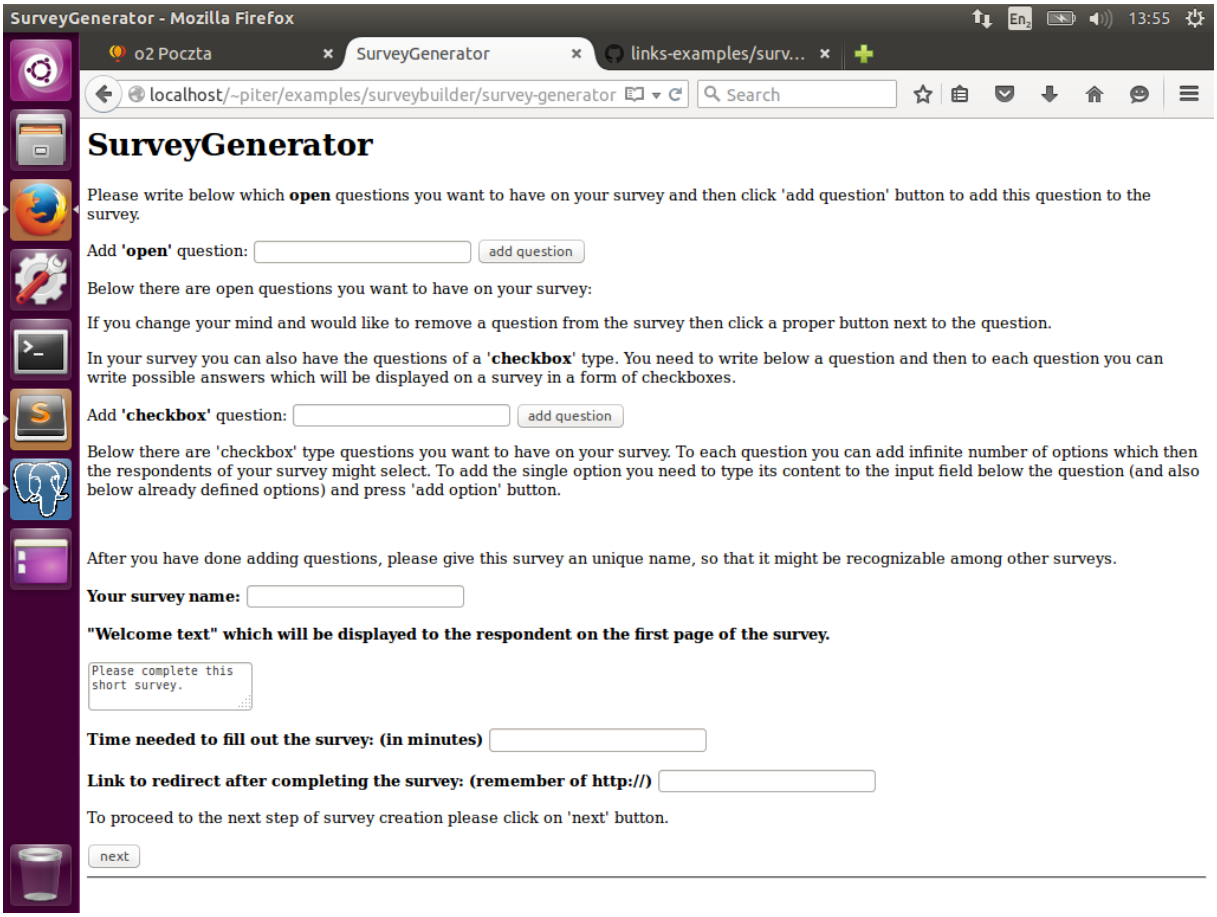
It was not explained before what happens when user will click on the link with the survey name. He/she will be redirected to the page where all the respondents of the survey will be listed and from there he/she could manage them and their answers. It looks like that:



There is the table listing the respondent name, link to his answers ("SingleSurveyEditor" module) which can be also sent him when he/she would like to introduce some changes and link to remove the respondent entries for this survey. The source code of this page is stored in survey-respondents.links file.

"SurveyGenerator" module

This is one of the most pivotal part of the system and the first one implemented (after defining SQL schema). In this subprogram all the new surveys emerge. The source code is located in survey-generator.links file. Creation of the survey consists of two steps. The first one, shown in pictures below, is actually the most important one.



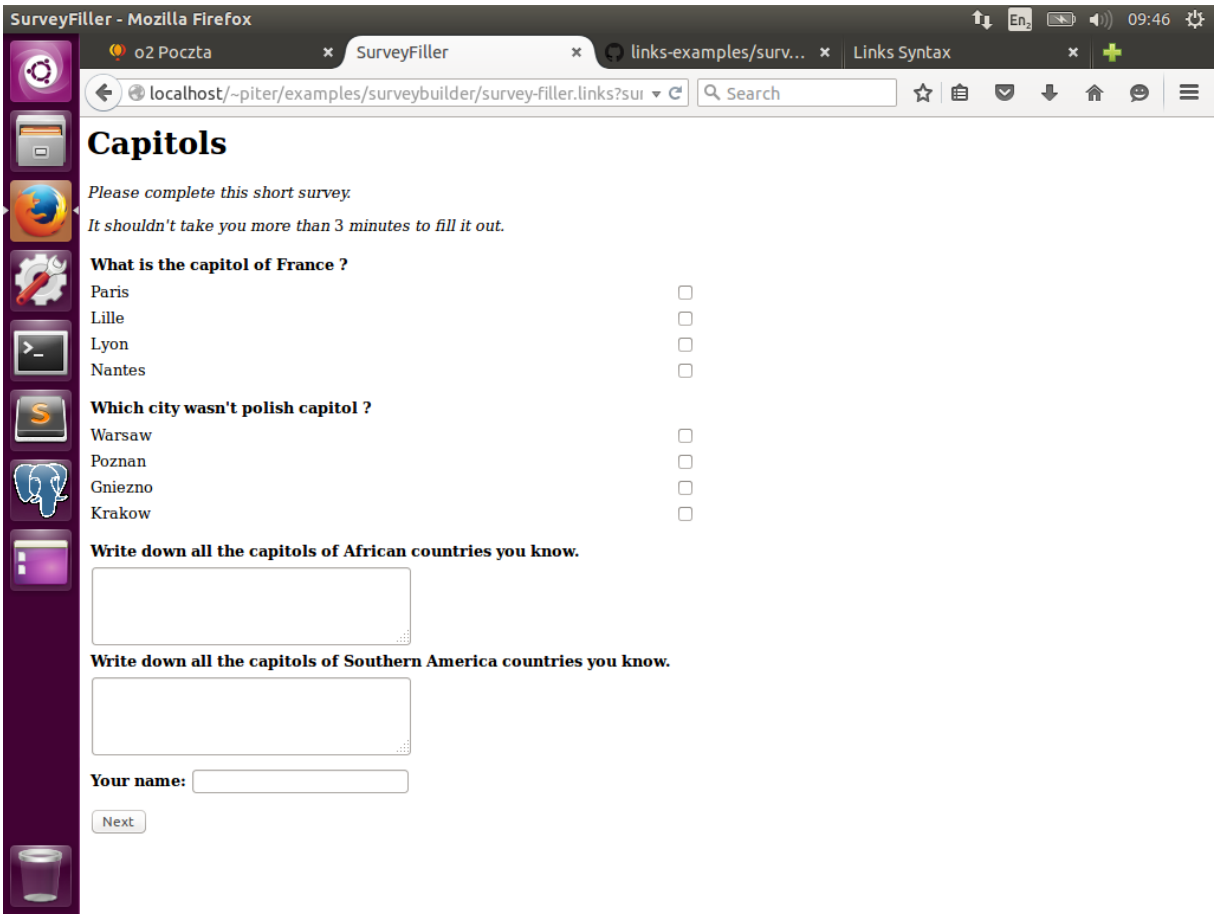
Here we are able to add open questions to our survey as long as questions of checkbox type. If we want to add a new open question we have to write it next to the “Add ‘open’ question:” label and press the add question button. This will cause a page to be reloaded (no AJAX implemented) and in the meantime the content of the question will be stored in `tmpquestions` database table. Then the added questions will be listed below the “Add ‘open’ question” form. It will be also possible to remove already added question from our survey prototype by clicking on “remove question” button which will appear after the question will be added. Adding the question of checkbox types happens analogically. At first we have to type in the question to input next to the “Add ‘checkbox’ question” label and then click on “add question” button. However for each checkbox question we might have more possible options out of which user can form his final answer. After a new question of a checkbox type is being added there is always a similar input for each question which enables to add an option for the question. The number of options per question is rather unlimited. And also if we want to remove a question or option it is possible by click on the “remove question” or “remove option” button which are next to the questions or options. Each time we click an add button a proper function is called and temporar results are stored in a database tables. For checkbox questions these tables are: `tmpcbquestions` and `tmpcboptions` . All these operations in backend are wrapped into forms and clicking on a add button means submitting this form and runs a triggers some action (function). These functions are `add(name)`, `addcbquestion(name)`, `addcboption(name, qid)`. Similarly when clicking on “remove group buttons” functions: `remove(name)`, `removecbquestion(name)`, `deletecboption(name, qid)` are called.

Each survey has to have some basic properties as the name, some welcome text which will be displayed to the respondent, approximate time needed to fill it out and in case of our system the link (since we program in Links) which will be proposed respondent to go after he/she fills out the survey.

All the logic of this first step happens in the *main()* function. After clicking on the next button the execution flow is being forwarded to `summarizesurvey(surveydata)` function. This is the place where all the defined by the user information are displayed so that the user can once again see what is going to be on the survey, and if the user is sure of it, he/she can click on the "submit" button to store the information in a database. It is done in a following order. At first we create a new entry in database in a table `surveys` where we provide all the properties of the survey the user has entered. DBMS generates an unique ID for the survey which is then read by us. When we know the survey ID we copy the content from the so called temporary tables (`tmpquestions, tmpcbquestions, tmpcboptions`) to the target tables (`textareas, checkboxquestions, checkboxoptions`) which store information about the questions of different types which have been given by an user for this survey. At the end we clean the temporary tables deleting all their hitherto content and display a success message on the webpage.

"SurveyFiller" module

This module performs a significant role in the "SurveyBuilder" Survey Management System. It is the part which is being sent to the potential respondent (in a form of link) so that he/she could fill out the survey and the author would get interesting from his/her point of view answers. The source code of this module is placed within the survey-filler.links file and the respondents of the surveys will obtain a page similar to the one below.



It consists of the survey title, some welcome text, information about approximate time needed to fill out the survey at the top, checkbox type above the open questions defined by an author of the survey. We also assume that each respondent has to give his/her name. From the point of view of back-end logic the most critical part is recursive formlet method which is responsible for gathering

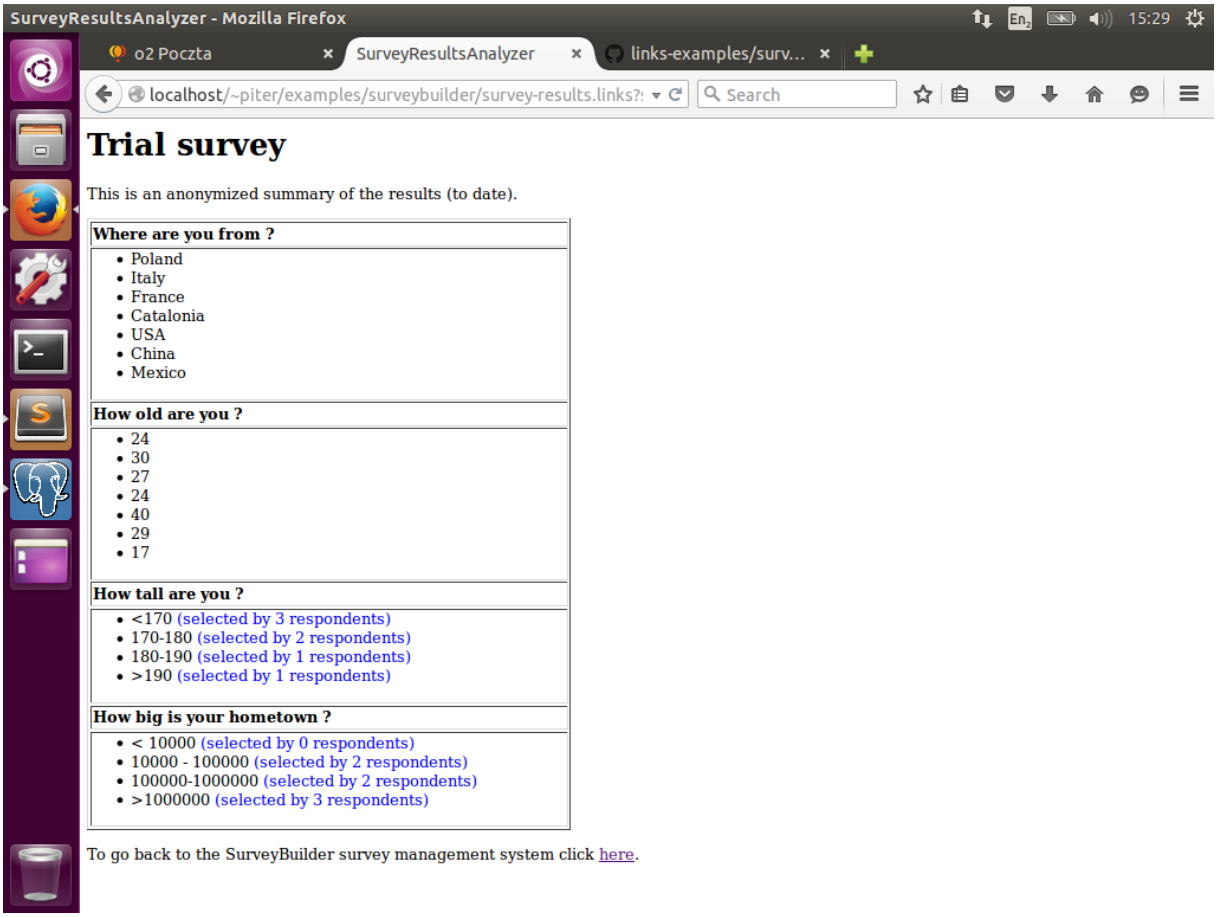
user information and creation of a proper data structure from them. It is all done in a *nSurveyFormlet(copt, textareaquestions, checkboxesqo)* function which produces or rather yields a following record as an output: (name:String, listofvalues:[(`id:Int, answer:String, question:String)`], cbvalues:[(`questiontxt:String, answers = [Int]`)]), where name is the nickname of respondent, listofvalues are anwers for openquestions and cbvalues are answers for checkbox type questions. In the matter of *nSurveyFormlet* function parameters textareaquestions are the information about open questions to be displayed and checkboxesqo are the information about checkbox type questions to be displayed. The parameter copt decides whether the next question to display should be of a checkbox type (value 1) or an open question (value 2). What is also worth mentioning is the fact that there is embedded recursion because recursive function *nSurveyFormlet(copt, textareaquestions, checkboxesqo)* calls inside its body another recursive function `checkboxesFormlet(cboptions)` which is responsible for creation and gathering data from the checkboxes. It may appear to be complicated but after being cautious with stop conditions it works pretty well.

After the respondent submits his/her answers clicking on the “Next” button he/she will have shown his/her answers. When he/she submits them again, the process of data saving into database will start. The first information we insert is a new respondent. DBMS automatically generates for him/her a new ID, and then with this ID we can start inserting his/her answers. Of course we are also using the ID of the survey as long as IDs of the questions. So as we can see everything is very associated.

After the data are successfully stored in the database there will appear a thank words, a recommendation of a webpage to visit (link) and some image.

“SurveyResults” module

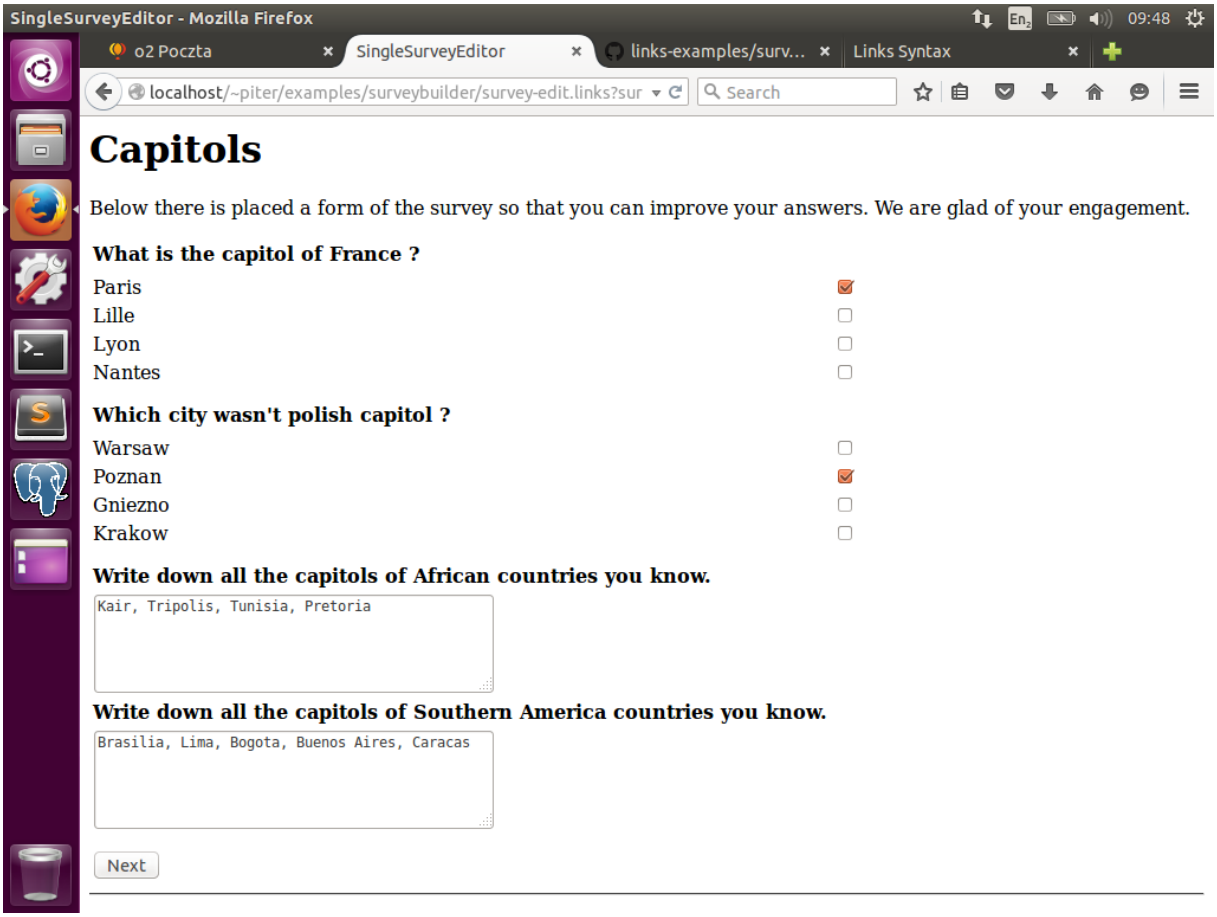
The implementation of this module is kept in survey-results.links file. This module can be accessed when the user clicks on the “click here to see” link which is in the “Results” column of the table with surveys. There are displayed all the results in the form of table. One row is a question and the row below are the answers already given to this question by all the users. In case of open questions the answers are listed one by one, so that it is very easy to set together the answers of a particular person. In case of checkbox type questions there is a list of options and the information how many times each option was selected by different users in total. In the logic of this module there is lot of select queries which download all the necessary information from the database. The print screen of this module is shown below:



"SingleSurveyEditor" module

This module works mostly in the same way as "SurveyFiller" module. However there are some differences and extras comparing to "SurveyFiller" module. Starting from the beginning besides the questions belonging to one survey we need to download from database also information about the answers to these questions that the respondent has already given and pass them to the formlet responsible for gathering data so that the respondent know what answers had he/she already given. Another change relative to "SurveyFiller" module is use of *checkboxDefault* formlet instead of ordinary *checkbox* formlet. It enables creation of checkboxes that can be already selected in the beginning. And the final difference is the behaviour after respondent submits editing his/her answers. Instead of insert operations there are plenty of update queries executed.

And below is how does this module presents at the webpage:



The implementation of this module is kept in survey-edit.links file.

Survey removal functionality

This is the simple functionality which enables survey author to remove the chosen survey from the list of his/her surveys after he/she does not need it anymore. The code of this functionality is placed in the surveydestroyer.links file. It deletes from the database all the information connected with the survey. All the questions, answers and also respondents who filled it out as the database schema consists only of kind of one-to-many relationships (there is kind of because foreign keys were not defined as an official constraint).

Respondent removal functionality

This functionality enables the author of the survey to remove unwanted or duplicated respondent entries for a given survey. The code of this functionality is placed in the respondentdestroyer.links file. It deletes from the database all the information connected with the given respondent. So it deletes all the database entries (in tables responsible for storing answers like: `textareaentries` and `checkboxentries`) where the ID of the respondent (in both tables it is the field `respondendid`, written with orthographical mistake) is the one given.

What is missing and perspectives for further development

Because the works upon this system last less than 6 weeks, within which the first couple of days the developer had to learn the syntax of Links and the specifics of functional programming language there are lot of features and functionalities which could be added to extend this application.

First of all, the survey can consist of only two types of question now (open and checkboxes), but Links supports also such user input reading means as: radio buttons, popup menus or multi-select scroll boxes which can be added to increase the range of types of questions on the survey.

The next thing which could be added (very critical one) is authentication panel. Now we just start an application with no need to log in. To do that the database schema should be enlarged, some tables would be altered but it would be a very useful functionality.

The last but for sure not the least important improvement which comes into my mind is layout (and UX) of the whole system. Now it is written in almost pure HTML, with almost no use of CSS or JavaScript and looks like the pages from the early 90ies of the previous century. However in the current stage of Links playing with styles is very challenging and would require a really good and experienced front-end programmer.