Bachelor Semester Project - Web application for Jobs Observer

Monday 16th December, 2019 - 03:02

Stetsenko Olexandr
University of Luxembourg
Email: olexandr.stetsenko.001.student@uni.lu

Nicolas Guelfi
University of Luxembourg
Email: nicolas.guelfi@uni.lu

***Abstract*—Abstract**

1. Introduction

Computer Science is one of the most important domains in the world and become indispensable in everyday life. As we live in a digital age, there are a lot of industries and companies which focus on improving and exploring new capabilities in this domain. Thus, this led to the grow of complexity and creation of the new subfields which request new skills from employees and create new places in the companies and industries.

So, a lot of new students who started to study the field of Computer Science may ask themselves, what are the possible fields of Computer Science and what are the possibilities in terms of career perspectives and the availability of jobs per field. For answering theses questions, a software was been developed for performing the actual computer science market analysis. It consists to fetch all available jobs from different countries and companies which belong to the Computer Science domain. Then, based on theses founded jobs, a lot of different statistics can be created such as tables and plots which can be used for observing the trends of the domain. Therefore, by using and observing theses statistics, students will have the possibility to choose the right field and seeing the perspectives for future career.

The Bachelor Semester Project will focus on deploying this recent developed software on a cloud server in order to make his futures available to all students, who would like to observe the actual Computer Science possibilities.
During this project, a small but essential part of the software will be deployed and will be considerate as the basis for future upgrades. Moreover, this part must be operational and include some of possible statistics of the existing software. Then, statistics delivered by the software will be discovered for understanding at best the results for correct implementation.
As the software deployment and development are in the context of software engineering, a small research in the field of software testing will be perform. This research will consist to prove the correctness of a program by looking from the scientific point of view.

2. Project description

2.1. Domains

The Bachelor Semester Project is done in different domains of computer science such as Software Engineering, Web development and Software testing.

2.1.1. Scientific domain. Software testing is a process which consists to verify and validate that a software is a bug free. The purpose of software testing is to find errors and bugs in the functionalities of a software that can lead from smaller failures to the malfunction of the overall system. Software testing is divided in verification and validation.
Verification[1] process allows to understand, if the development of the software is right. It consists to verify the specified requirements in order to understand if the development of the software is conformed to these requirements. Validation[1] is a process which consists to understand, if the developed software is the right software. This process happened after the development of a software and allow to answer, if the developed software meets the requirements and the needs of the customers.

2.1.2. Technical domains. Software Engineering is based on the principles, which are used in the field of engineering. In Software Engineering, developers develop different types of software which can be small, large, complex or easiest. During the development, they deal with the requirements specification, software design, testing and maintenance. Moreover, developers must apply a structured and disciplined approach for being able to navigate codes, during the development and maintenance.
In Software Engineering, developers must base on the customer's requirements, for providing a software that meets their needs. In addition, some other different factors have to be considerate as budget, time constraint or efficiency.

Web development consists to create web applications that are developed, deployed and accessed on the Internet.

Web development is divided in front-end and back-end development.
Front-end development consists to design the graphical interface of a web application that will allow the interaction between the system and the end users. Usually, the front-end is composed of several elements that are designed by CSS, HTML or JavaScript.
Back-end development is hidden part of a web application. Back-end is composed of three parts such as functionalities of software, databases and server which allows to access and perform operations on the system.

2.2. Targeted Deliverables

2.2.1. Scientific deliverables. The scientific deliverable is a presentation of program correctness proof using logical reasoning. The proof is based on natural deduction that allows to derive a conclusion based on the premises.
For proving the correctness of program, the proof is based on the principle of unit testing that shows if a program is correctly executed or not.Then, by using the developed proof, an explanation on how this proof can be applied for evaluating the correctness of Jobs Observer.

2.2.2. Technical deliverables. The technical deliverable is an operational software deployed on an AWS server. This technical deliverable can be divided into two parts, front-end and back-end.
Front-end deliverable is divided into two parts. First part is a mockup of the Graphical User Interface (GUI) for Human Computer Interaction in order to elicitate the requirements on the GUI. Second part is to create static web pages based on the designed mockup which will represent the GUI in web-server.
Back-end deliverable consists to develop a web-server by using Django python framework. Web-server is composed of static GUI web pages that can be interacted by the users. The web-server is deployed on an AWS server which allows to Job Observers to run continuously. Jobs Observer provide statistics on jobs in the field of Computer Science.

3. Pre-requisites

3.1. Scientific pre-requisites

3.1.1. Proof. Proof is a logical demonstration which shows the truth of a proposition based on assumptions or axioms. For writing a proof, there are no only one method which can be used but several methods s.t direct proof, mathematical proof, natural deduction proof and more. During this project, a proof is developed based on natural deduction.

3.1.2. Natural deduction. Natural deduction[2] is an intuitional method which allows to develop a proof. In Natural deduction all the developed proof is considerate as the proof by hypothesis.
The proof is developed based on smaller proofs which are then combine at the end based on the rules for providing a justification to an assumption or hypothesis. Usually a proof is written as sequence of lines where for each line of justification is based on the any previous line, also called premises.

3.1.3. Unit testing. Unit testing is a way of testing which purpose is to test the separate parts of a program and shows that each individual part is working correctly.
For showing the correctness of execution, some requirements are defined which include the inputs and the correct outputs. In unit testing, the developer knows in advance what is the implementation of the block of code to test. So, based on these knowledges, a test code will be produced for determine if a block to test behaves as expected.
For showing that a code behaves as expected, the block to test will accept some predefined inputs from the requirements and will produce an output. Then, the test code will check if the output of the tested code is correct with respect with the output from the requirements.

3.2. Technical pre-requisites

3.2.1. Python[2]. Python is an interpreted, object-oriented, high-level and multiplatform programming language dedicated for web and application development. It has a dynamic semantics [3] which define how and when the various constructions of a language should produce a program behavior. Python programming language allows a dynamic typing which gives the possibility to change type of value of a variable to another type without receiving an error. Additionally, it has a dynamic binding [4] option which allows to a computer program to wait until runtime for binding the name of a method for the actual subroutine.
The reason for using Python programming language in this project, is that it is easy to learn, to maintain and to develop. Moreover, it offers a possibility to divide a program in modules which allow to have a structure and improve organization. Additionally, some different packages can be imported from python library and they are used for facilitate the program development.
The advantage to use modules and packages, is that they can be imported anywhere in the program and their functionalities can be easily reused.

4. A Scientific Deliverable

4.1. Requirements

In this section, the requirements of scientific deliverable are identified in order to understand the goals that must be achieved at the end of production. The scientific deliverable is divided into two parts, proofs and explanation on how theses proofs can be applied.

4.1.1. Proofs.

- R1: The first proof has to use unit testing for showing that a program is correct executed and

provide the right results with respect to the results of the tests. Moreover, the program to test has to be considerate as a function which is composed of a single or multiple function.

- R2: The second proof has to reuse the provided result of program correctness from the previous proof and identify if a program is correct in terms of program requirements. In other world, is the program correct from the behavior point of view.

- R3: Both proofs have to demonstrate the correctness of a program by using logical reasoning [5]. So, the creation of the proofs has to be done in a rational, systematic series of steps based on mathematical principles for arriving to the conclusion. Additionally, the proofs have to use a structural approach such as division into sections, for allowing to visualize and understand better the steps performed for achieving the final conclusion and also to reduce redundancies between the both proofs.

- R4: The proofs should use Natural Deduction which is based on assumptions and hypothesis. The proofs have to be constructed of smaller proofs which are assembled together at the end to a bigger proof.

- R5: Some explanations of notation used in the proofs should be provide for improving the understandability and avoiding ambiguities.

4.1.2. Explanation of how the proofs can be applied.

- R1: Provide a textual description of how the proofs can be applied to Jobs Observer application in order to understand if the reasoning of the proofs is correct from the application point of view. Also, some explanations have to be provided for understanding why theses proofs are applicable or why not.

- R2: The explanation must be consistent with respect to the proofs. For achieving this consistency, the explanation must follow steps by steps approach used in the proof such that each step describe into the explanation can be referenced to a step into a proof.

4.2. Design

In this section, the explanations on how the proofs are created are provided. The explanations are composed of what is the correctness of a program for understanding the goals to achieve during the devopement of the proofs and for introducing different notations use in the proofs.

4.2.1. What is correctness of a program.

4.2.1.1. Introduction to correctness of a program

The first question to ask yourself is how it is possible to show that a program is correct? The answer is that for showing the correctness of a program some tests have to be performed and the outcomes of theses results have to be compared with the expected result.
It exists a lot of methods that allow to understand that a program is correct, as example white-box and black-box testing. Theses methods are composed of different technics where each technic allows to show that a program is correct from a certain point of view. For this project, a method of white-box testing is used.
The particularity of this method is that the tester knows the structure, implementation and design of the program in advance. A technic which belongs to white-box testing, called unit testing is used for understanding the correctness of a program.

4.2.1.2. Correctness based on unit testing

Unit testing allows to test the correctness of one or several components of a program. So, for testing a component one or several tests are defined. Each test based on this technic accept one or several inputs, a function which is basically the code of the component and one or several outputs which are compared with the outputs of the component. The component to test accept the same inputs and produced one or several outputs.
Then, for showing the correctness of a component for the accepted inputs, the outputs are compared.If the outputs are the same then the component is considerate correct for theses inputs.
Unfortunately to perform one test on a component is not sufficient for showing and proving that a component is correct. For proving that a component is correct, all the possible inputs have to be tested and validated by comparison of outputs. If, all the tests are considerate as successful then a component can be considerate as correct.
The problem is that it is not possible to test all the inputs because the number of possible inputs is sometimes too big and infinite. Consequently, the component will never be considerate as correct. As, for proving the correctness of the whole program, all the components have to be correct. If, one component is considerate as incorrect then the entire program is also considerate as incorrect.
So, how this problem with the inputs can be solve so that the whole program can be correct. Basically, the number of inputs have to be limited which can be realize by defining the requirements.

4.2.1.3. Correctness of a program based on requirements

The requirements play a big role in testing because they

are used for understanding and setting the behavior to a program.
It exists two types of correctness, execution and behavior correctness. The difference between both is that the correctness of execution allows to understand if the program return the right result based on his implementation. The behavior correctness allows to understand if the program produced the right result from the requirement point of view.
Before, we mention that not all the component can be considerate as correct due to the incapability to test all the inputs.Now, this problem can be resolve with the requirements. So, basically, the requirements are used for setting the inputs to test and when all the defined inputs are tested, then a component can be considerate correct.
Thus, if all components will perform the test on their inputs defined by the requirements and will be considerate as correct then the entire program will be correct. But the program is correct from the execution point of view and not from the requirements point of view.
The problem is that just before the requirements are used as "limitation" for testing and do not represent the behavior of a program or a component. Basically, the requirements define, what a program or a component has to achieve as behavior at the end of execution. In order to guarantee the correctness of an entire program or a component from the behavior point of view and not only from execution point of view, all the requirements set before have to be satisfied.

4.2.2. Useful notations and explanations.

4.2.2.1. What is a program block

A program is composed of functions or some small part of codes which can provide an output based on inputs. A function accepts one or several inputs and produce an output. For a small part of codes, some adjustment has to be performed for allowing to start a test of a part of code from a certain state which is considerate as the inputs.

4.2.2.2. Functions

In the proof, each function is declared as: f: inputs $\rightarrow$ output. Each function accepts one or several inputs and will produce an output. The body of the function has to be considerate as a sequence of statements which represent the implementation of the program or a component to test.

4.2.2.2. Tuple and Triple

In the proof, some tuple and triple are defined for representing a variable, let's called it v. In case when a value from a tuple of a triple have to be retrieved, the notation of $[v]_{value}$ is used.

4.3. Production

In this section, the final deliverables which are the two proofs and the explanation of how theses proofs can be applied are described.

4.3.1. First Proof.

The first proof consists to show the correctness of a program from execution point of view. For understanding the entire proof, the explanation is divided into sections where firstly a small part of a proof is shown which is followed directly with a small description.

4.3.1.1. Declaration of a program block

$$Let\ inputs_{fpb} = \{v \mid v\ is\ a\ value\}$$

$$Let\ func_{fpb}: inputs_{fpb} \rightarrow output_{fpb}, where\ func_{fpb}\ is\ a\ function$$

$$Let\ pb_i = (func_{fpb}, inputs_{fpb}, output_{fpb})$$

Firstly, the inputs for a program block is declared. The inputs are considerate as a set of values which can represent any possible value of any type. Then, a function is declared, it accepts some inputs and return as result an output. This function has to be considerate as the implementation of a program block. At the end, a program block is created as triple composed of function, inputs and output.

4.3.1.2. Declaration of all program blocks

$$Let\ Blocks(P) = \{\, pb_i \mid pb_i\ is\ a\ program\ block\ of\ P\}$$

After the declaration of a program block, the entire program is defined by Blocks(P). The logic behind is that a program is composed of several program blocks that can be considerate as a set of program blocks.

4.3.1.3. Declaration of a test

$$Let\ inputs_{ft} = \{v \mid v\ is\ a\ value\}$$

$$Let\ func_{ft}: inputs_{ft} \rightarrow output_{ft}, where\ func_{ft}\ is\ a\ function$$

$$Let\ t_j = (func_{ft}, inputs_{ft}, output_{ft})$$

The declaration of a test is similar to a program block. The big difference is that the output of a test and of a program block can be different. For representing a test t_j ,as for a program block, a triple is used and it is composed of function which is the implementation of a test, inputs and outputs.

4.3.1.4. Declaration of all test sets for P

$$Let\ TA_{pb_i} = [t_j \mid t_j\ is\ a\ test]$$

$$Let\ TS_P = \bigcup_{pb_i \in Blocks(P)} TA_{pb_i}$$

Then, a declaration of all tests performed on the program P is done by firstly considerate an array of tests which is a simple matrix (1xn). Then a set of all tests is declared by performing the union on all arrays composed of several tests.

4.3.1.5. Show the validity and success of a test on a program block

$$success(t_j, pb_i) = \begin{cases} 1, if\ [t_j]_{output_{ft}} = [pb_i]_{output_{fpb}} \\ 0,\ otherwise \end{cases}$$

$$valid(t_j, pb_i) = \begin{cases} 1, if\ [t_j]_{func_{ft}} = [pb_i]_{func_{fpb}}\ and\ [t_j]_{inputs_{ft}} = [pb_i]_{inputs_{fpb}} \\ 0,\ otherwise \end{cases}$$

$$valid(t_j, pb_i) \rightarrow success(t_j, pb_i) = 1$$

Now, the correctness of a test performed on a program block is declared via the success(t_j,pb_i) variable. success(t_j,pb_i) accept two inputs, a test and a program block and provide as result 1 when the output of test is equal to output of program block, otherwise 0.If the result is 1 then a test is successful, otherwise 0 which is a failure. As, not all tests can be performed on a program block because some are dedicated to other program blocks, a test has to be validated in order to understand, if it is the right test to perform. This is realize by valid(t_j,pb_i) which accept a test and a program block as inputs and provide as result when a test can be performed on a program block. The result is 1 when the inputs and the implementation of program block and a test are similar, in other case 0.

4.3.1.6. Proof that the whole program is correct

$$correctBlock(TA_{pb_i}, pb_i) \equiv \forall t_j \in TA_{pb_i} |\ valid(t_j, pb_i)$$

$$correctP\big(TS_P, Blocks(P)\big) \equiv$$

$$\forall pb_i \in Blocks(P), \forall TA_{pb_i} \in TS_P \mid correctBlock(TA_{pb_i}, pb_i)$$

The final step of this proof consists to demonstrate if the whole program is correct. For explaining this part, a bottom up approach is used. For testing the entire program all the tests have to be performed on each program block. So, basically a set of all subsets of tests dedicate to each program block is taken. Then, each test of theses subsets is applied to a program block which are first validate and then tested.

4.3.2. Second Proof.

4.3.2.1. Declaration of pre and post conditions

$$Let\ condition \equiv [v \mid v\ is\ a\ value]$$

$$Let\ conditionsPre \equiv \{condition \mid condition\ is\ a\ set\ of\ values\}$$

$$Let\ conditionsPost \equiv \{condition \mid condition\ is\ a\ set\ of\ values\}$$

$$Let\ pre-condition([t_j]_{inputs_{ft}}) =$$
$$\begin{cases} 1, \forall input \in [t_j]_{inputs_{ft}}, \exists\ condition \in conditionsPre \mid input \in condition \\ 0, otherwise \end{cases}$$

$$Let\ post-condition([t_j]_{output_{ft}}) =$$
$$\begin{cases} 1, \forall output \in [t_j]_{output_{ft}}, \exists\ condition \in conditionsPost \mid output \in condition \\ 0, otherwise \end{cases}$$

Firstly, define a "condition" as an array of values which can be of any values and any types. Then, the "conditionsPre" and "conditionsPost" which represent the pre-conditions and post-conditions are defined, each by a set of "condition". Next, "pre-condition($[t_j]_{inputs_{ft}}$)" and "post-condition($[t_j]_{outputs_{ft}}$)" functions are used for evaluating the inputs and outputs of a test with respect to "conditionsPre" and "conditionsPost". The "pre-condition($[t_j]_{inputs_{ft}}$)" return 1 as result when for each input in inputs, it exists at least one condition in conditionsPre where the input belongs to condition, 0 otherwise. Thee same is done with "post-condition($[t_j]_{outputs_{ft}}$)" but instead of inputs, outputs are used and are evaluated based on cconditionsPost.

4.3.2.2. Definition of all requirements

$$Let\ r_i = (pre-condition([t_j]_{inputs_{ft}}), post-condition([t_j]_{output_{ft}}))$$

$$Let\ RS_{pb_i} = [r_i \mid r_i\ is\ a\ requirement]$$

$$Let\ RS_P = \cup_{pb_i \in Blocks(P)} RS_{pb_i}$$

Each requirement is composed of two functions which allows to validate the inputs and the outputs of a test. Then, for each program block, it exists an array of requirements RS_{pb_i} that have to be satisfied for the correctness of a program block. At the end, all the requirements are defined as a set of all requirements RS_{pb_i}.

4.3.2.3. Test satisfaction of a requirement

$$inputsEquality(t_j, pb_i) = \begin{cases} 1, if\ [t_j]_{inputs_{ft}} = [pb_i]_{inputs_{fpb}} \\ 0, otherwise \end{cases}$$

$$outputsEquality(t_j, pb_i) = \begin{cases} 1, if\ [t_j]_{output_{ft}} = [pb_i]_{output_{fpb}} \\ 0, otherwise \end{cases}$$

$$conditionInput(t_j, pb_i, r_i) = \begin{cases} 1, if\ inputsEquality(t_j, pb_i) | [r_i]_{pre-condition([pb_i]_{inputs_{fpb}})} \\ 0, otherwise \end{cases}$$

$$conditionOutput\ (t_j, pb_i, r_i) = \begin{cases} 1, if\ outputsEquality(t_j, pb_i)\ | [r_i]_{post-condition([pb_i]_{output_{fpb}})} \\ 0, otherwise \end{cases}$$

$$satisfy(t_j, pb_i, r_i) = \begin{cases} 1, if\ conditionInput(t_j, pb_i, r_i)\ and\ conditionOutput\ (t_j, pb_i, r_i)\ and\ valid(t_j, pb_i) \\ 0, otherwise \end{cases}$$

In this section, the satisfaction of a requirement is described by firstly looking at the equality of inputs and outputs between a test and a program block. For this purpose, two functions are declared, "inputsEquality" and "outputsEquality". "inputsEquality" compare the inputs and "outputsEquality" compare the outputs.
Then, the inputs and outputs of a test and a program block have to be in requirement. For demonstrating that the inputs are in a requirement, the inputs of a test must be equal to inputs of a program block. Then, the inputs of a program block are evaluated based on "pre-condition" function from the requirement. The same strategy is applied to outputs but instead of "pre-condition", "post-condition" is used.
For testing the satisfaction of a requirement, "satisfy" function is used and accept as inputs a test, a program block and a requirement. So, for satisfying a requirement the inputs and outputs have to belong to requirement and the test must be valid and successful.

4.3.2.4. A specific requirement is satisfied for a program block

$$Let\ satisfactionR(TS_{pb_i}, pb_i, r_i) =$$
$$\begin{cases} 1, if\ \forall t_j \in TS_{pb_i} | satsify(t_j, pb_i, r_i) \rightarrow (\exists t_j \in TS_{pb_i} | satisfy(t_j, pb_i, r_i) = 1) \\ 0, otherwise \end{cases}$$

For demonstrating the satisfaction of a concrete requirement, the "satisfy" function is performed on each test for showing if the requirement is satisfied or not. Then, between all the performed tests, it must exist one test where the "satisfy" function return 1 and satisfy the requirement.

4.3.2.5. All requirements are satisfied for a program block

$$Let\ satisfactionSetR(TS_{pb_i}, pb_i, RS_{pb_i}) =$$
$$\begin{cases} 1, if\ \forall t_j \in TS_{pb_i}, \forall r_i \in RS_{pb_i} | satisfactionR(TS_{pb_i}, pb_i, r_i) = 1 \\ 0, otherwise \end{cases}$$

Then, all requirements have to be satisfied for a program block. For achieving this objective, "satisfaction" function from previous section have to be reused on the whole array of tests and the whole array of requirements for a program block.At the end, all the requirements for a particular program block have to be satisfied.

4.3.2.6. All requirements are satisfied for a program

$$Let\ satisfactionALL = \forall pb_i \in Blocks(P), \forall TS_{pb_i} \in TS_P, RS_{pb_i} \in RS_P \mid satisfactionSetR(TS_{pb_i}, pb_i, RS_{pb_i}) = 1$$

At the end, for proving that the program is correct from the behavior point of view. All the requirements have to be satisfied. So, all the tests performed on each program block must be successful and all the requirements defined for each program block have to be satisfied.

4.3.3. How the Proofs can be applied to check the correctness of a program. For looking for the correctness of a program based on unit testing. A program must be already developed and which is correct from the syntactical point of view. Moreover, the created program must be composed of functions or of parts of codes that can be individually separated from the entire program.The reason is that for the proofs, a program block is considerate as a function which accept an input and produce an output.
The question is how the proofs can be applied to the Jobs Observer web application?
Firstly, the Jobs Observer web application is developed in a way that it is composed only with the functions or individual parts of codes that can be separately tested. For showing how the proofs can be applied, only views.py module will be used because the entire program is too big for the demonstration.

4.3.3.1. Application of first proof based on views.py module

Let's considerate this views.py as a subprogram of Jobs Observer which will be our program to test. At beginning, this program has to be divided into blocks which consist of functions that accept a request as input and provide a Http response as output.
Then, different test cases must be created for each program block to test. Each block can have one or more test cases which will be considerate an array of tests. All these array tests are considerate as a set for testing the entire program. When all tests are created, all arrays of tests are applied for testing all program blocks. All the tests in each array will be passed and check for the validity for showing if a test is the right test for testing the program block. If, the test is valid then it is applied to the program block and the result of outputs are compared. If, the outputs are the same, in our case is a Http response, then the test is considerate as success, otherwise it fails.
In the proof, if one test fails then the entire program will be considerate as incorrect. So, all the functions of views.py module must be correct. Thus, the entire program will be considerate correct. So, based on this proof, we showed how a part of a Jobs Observer can be tested for execution correctness.
For showing that the entire Jobs Observer is correct, all the subprograms(modules) must be correct executed.

4.3.3.2. Application of second proof So, for performing a second proof, the requirements have to be defined for views.py module, as an example each function return a HTTP response to the right HTML page.
When, all the requirements are created, the same tests can be performed on the views.py module as for the first proof but each time before the test the satisfaction of requirement have to be applied on inputs and outputs. Then, during the process of testing all the tests, the requirement is also evaluated for the satisfaction for each program block. It means that for each program block all the requirements have to be satisfied. If, for each program block all the requirements are satisfied and all the tests are successfully passed, as example all the tests on a program block return a HTTP response to the right HTML page then the program block behaves as expected. If, all program blocks satisfy all the requirements, then the behavior of the whole program is correct.

4.4. Assessment

In this section, the comparison of scientific deliverables with respect with the requirements previously defined is described.

4.4.1. Evaluation of proofs. The first requirement for the first proof is satisfied because the created proof demonstrate the correctness of a program by using the unit testing. Additionally, the proof demonstrates how the correctness of a program can be achieve if it is composed of several subprograms.
The second requirement is also satisfied because the second proof demonstrates the correctness of a program not only from the execution point of view but also from the requirements point of view.

The third requirement and the fourth requirements are also successfully achieved because the both profs use the logical reasoning. So, the two proofs are based on mathematical demonstration by using step by step approach for arriving to a conclusion. Moreover, they are based on Natural Deduction and use the premises for developing proofs and also reuse the sub proofs for creating a new one.
The fifth requirement is also achieved. Before starting to develop proofs, some notations are set and explained for improving the understandability.

4.4.2. Explanation on how the proofs can be use on Jobs Observer. The first requirement for this part is met because the proofs can be applied to the real-world problem which was been shown with the Jobs Observer views.py module. For satisfying the proofs some assumptions have to be define and the program to test must be conform to these assumptions in order to do the proofs work. The second requirement is partially satisfied. The second requirement consists to provide an explanation which is consistent with respect to proofs, so that each part explained can be find in the proofs. The reason why this requirement is partially satisfied is that some parts for the program have to be assumed, as example the requirements for the program or when the views.py module in Jobs Observer is correct, we considerate all other program blocks also correct for demonstrating that the whole Jobs observer is correct.

5. A Technical Deliverable 1

5.1. Requirements

In this Bachelor Semester Project, there are several requirements that must be satisfied. These requirements are divided into two parts: front-end and back-end. For each part, functional and non-functional requirements will be introduced.

5.1.1. Front-end.

5.1.1.1. Front-end functional requirements

In this section, we will see the most important functionalities that the front-end must satisfies. The front-end consists to design a GUI that allows the interaction between web-server and end-users.

- FR1: Graphical User Interface should be created by using static pages in order to obtain a first visual representation of the web application. These static pages must have some basic elements of a webpage allowing simple interactions. Additionally, some buttons should be used for displaying statistics based on the data used from the database
- FR2: The GUI static pages must have the possibility to display statistics by using buttons in order to allow the execution of back-end python script.

5.1.1.2. Front-end non-functional requirements

- NFR1: Friendly interface. The GUI should be easy to use without spending a lot of time for learning. Moreover, the user interface has to be intuitive so that the user can expect himself in advance the result produced by taking an action on an element of web page. Moreover, the basic elements of a webpage must be found in the right places.

- NFR2: Readability: The font-size of components of GUI must be easily visualize so that the information transmit by a GUI element can be easily understand by the user. Moreover, each static page must be not overload with the data in order to avoid the misunderstanding and confusing users.

5.1.2. Back-end.

5.1.2.1. Back-end functional requirements

In this section, the most important functionalities that the back-end must satisfies. The back-end part consists to develop a web-application which is deployed on an AWS server.

- FR1: The web-sever development must be realized by using Django framework which is written in Python and the compatibility between Django web-server and AWS server has to be ensured.

- FR2: The old database from the existed software has to be used in the Jobs Observer. As the database file is to big, it should be imported from an external source such Dropbox from where it will be downloaded and reused. As, the old database is designed by using peewee library, which is a local database, Jobs Observer has to reuse the codes from the existed software for making the database working.

- FR3: Operations. All the operations for statistics creation have to be imported to the Jobs Observer in order to apply them on GUI buttons. When a statistic is requested by clicking on front-end button, some of operations have to be used in the right order for producing a statistic. Moreover, the operations to use must be operational and provide the right results. Additionally, the creation of all statistics must be made in Django views.py module which allows to display the HTML pages.

- FR4: The back-end must be connected in a certain way to the front-end in order to execute the opera-

tions requested from the front-end and display the result at front-end.

5.1.2.2. Back-end non-functional requirements

- NFR1: Maintainability. The development of web-server must be done in a way that it can be maintainable after the end of the project. For ensuring this requirement, a structural approach has to be used, some comments have to be written for the understandability of the program. The right naming must be used for operations and variables.
- NFR2: Operationality. The web-sever must be operational after the development for allowing the execution of different statistics and to be able to provide some firsts results.
- NFR3: Compatibility. The created web-server must be compatible with AWS for allowing Jobs Observer to run continuously and non-locally.

5.2. Design

5.2.1. Project creation. The first step for designing the project, is the creation of Jobs Observer on AWS. The creation is realized by using a development tool called CodeStar which allows the creation of simple project and also more complex project such as a web-application. For creating the project, a Django framework is used, based on the Python language. Additionally, the project source code of the project can be found on github which allows to perform an incremental development.

5.2.2. Graphical User Interface. Before passing to the development part of web application, we will focus on the Graphical User Interface design. The first part consists to create a mockup by using an external online tool for creation of web sites. The creation of mockup is started by defining the structure of each page such as the locations of buttons, images and content. Then, the order in which these pages can be accessed was been imposed in order to ensure the logical passage between one page to another. There are in total four pages that are designed such home, about software, plots and statistic which can be considerate as the basis for the future development.
Then, the designed mockup was been used for created HTML pages for representing the content and a structure. CSS language is used for improving the visual representation of HTML pages.

5.2.3. Django structure. Django framework has a structure which allows the creation of web application. This structure is divided into two folders which allows the development and configuration of a web application. The first folder, called “ec2django” is used for the configuration. The second folder, called “helloworld” is used for the development of application.

5.2.3.1. Configuration

The “ec2django” folder is used for the configuration and contains four python modules such init, settings, urls and wsgi. The settings.py module is the most important and is the only one module in which some modification are performed. It allows to setup application parameters such as path to HTML static pages, type and location of database and the middleware. During the development, this module was been used for indicating the location of CSS files that are reused by HTML files for design. Moreover, it is used for performing some changes in middleware classes for allowing a local execution.

5.2.3.2. Development

The “helloworld” folder is used for the development of a web application (Notice: the “helloworld” name is the base name of application). As previously, this folder contains some modules such as init, admin, apps, models, views and urls. For the explanations, we will focus only on the last three modules.

5.2.3.2.1. models.py

This module is used for creation of database and declaration of database structure. As, we have already an existed database from the existed local software, this old database is reused for fetched the available jobs. Moreover, the oldest database was been created by using peewee package and the structure of database was been previously defined. So, the code for existed structure is reused for retrieving the data.
Unfortunately, the existed database is too big (around 1.2GB) and can not be used directly from the repository. So, this database is manually stored on a cloud server, called dropbox. When this module is executed, the stored database file is download to an empty database file which is then used as the principal database.

5.2.3.2.2. views.py

This module is used for processing the requests from the GUI and allowing the interaction between the GUI and users. This module is composed of several functions, called “views”, which accept a request and return a HTTP response with the request and HTML page to display. As, we have design previously different HTML pages for our web application, these functions are used for indicating the right HTML page to display. Additionally, it is possible to perform in between some additional operations that allow the creation of different statistics. Theses created statistics

are themselves the HTML pages and can be returned in HTTP response.

5.2.3.2.3. urls.py

This module is used for indicating the right urls for each view function in views.py module. This module is composed of a list of urls which are associated to each view function in views.py module. So, after the execution of a view function, an url is requested for displaying a HTML page.

5.2.3.2.4. stats.py

This module is one of the most important modules because it is used for creating different statistics based on available jobs in Computer Science. This module contains many functions that can be used for creating statistics. Usually for creating a statistic, a dataframe is firstly created and is used for creation of a plot or a table which represent a statistic. For creating the statistics, this module uses previously imported database for retrieving the data.

5.3. Production

In this section, the description of the deliverable concrete production will be explained. For this section, the deliverable is divided into two parts, front-end and back-end deliverable. Notice that this deliverable is only a prototype and represent the basis for future development.

5.3.1. Front-end deliverable. Front-end deliverable is a Graphical User Interface view which is composed of several HTML pages that are used for the interaction with the web-application. In total the front-end deliverable contains four HTML pages such as "Home", "About Software","Plots" and "Statistic".

5.3.1.1 Home page

The "Home" page is the first HTML page which show to the user when he accesses the web application. The purpose of this page is to show what kind of statistics are offered by the web application and to redirect a user to other pages where he will be able to find the descriptions of possible statistics or some general information about the software.
The "Home" page is separate into three parts such top, middle and down. The top of the page is a menu bar which is composed on a "BiCS" image on the left and some buttons to the right. There are six buttons to the right such "Home", "About Software", "Plots","Tables","Text" and "About us" which are use for redirect the user to other HTML pages. At the moment, there are only three first buttons that are operational.
The middle of the page is composed of four circles which are the clickable buttons. The biggest button contains the name and a logo and is used for redirect the user to an "About Software" page. The other three small buttons include images which show what can of statistics are possible and based on the title inside of images redirect the user to the right pages. The down of the page is reserved for additional information such as contact, Help&FAQ and Social Media but not yet completely finished.
For the next pages is top and down of each page is the same as in the "Home" page.

5.3.1.2 About Software page

The purpose of this page is to provide a user with the general information about the software. This page provides three types of information. The first is an overview about the software where the purpose of web application is described. The second describe what are the possible statistics and the third describe how the software works. Each description is place into a white box label.

5.3.1.3 Plots page

The purpose of this page is to provide a description what kind of statistics are possible and to allow a user to choose a statistic to display. The description of each statistics can be found in a white box composed of a title (the name of statistic) followed by a small text of expected result. At the bottom of each white box, there are a button "DISPLAY" which is used for displaying a statistic on another page.
On this page, there are right now four description of statistics among which two are operational, "Jobs Per Country" and "Jobs Per Company". These four descriptions are structured as follow, there are two statistic description per row and two per column.

5.3.1.4 Statistic page

This page appears when a user clicks on a "DISPLAY" button from the "Plots" page and show to user an interactive statistic. At the moment, only the statistic is displayed without any additional information about it. From the mockup point of view, the idea of this page is to display a window for the statistic and to add an additional information around.

5.3.2. Back-end deliverable. Back-end deliverable is a web application deployed on an AWS server. The web application is developed in Django and is composed of several modules. In this section, the concrete implementation of modules will be described such models.py, views.py, urls.py and stats.py and their relations.

5.3.2.1 models.py module

This module is used for the database creation and declaration of database structure. At beginning, an existed database file, called "jobs.db" is downloaded to the system from the Dropbox by using dropbox python package. This download is possible only, if it exists inside the system another database file. So, the "jobs.db" database file from Dropbox is downloaded into an empty database file also called "jobs.db" and this new file will be used as the new database. The reason to perform this download operation is that the existed database file from the previous software is too big and can't be store on github after that the project is already modified from his initial state.
Then, a structure for the database is declared for accessing the data and different tables of database. The database is composed of three types of table such Job, SearchQuery and Token. Job table is composed of several attributes such id, county, company, etc. SearchQuery table is composed of a term and a place which indicate the location. Token is composed of jobs, values and order which is used for indicating the number of available jobs.

5.3.2.2 stats.py module

This module can be found in "modules" folder. This module is completely imported from the existed software and is used for creating different types of statistics. It provides three types of statistics: tables, plots and text which should be only created as HTML pages. In this module some changes are performed compare to initial version in order to allow the correct execution on the server. The first change consists to change all path to more common paths. Another, change consists to remove the read Excel sheet with all keywords because AWS server doesn't allow to load the Excel sheet as a dataframe locally and externally. There exists a solution for this problem which can be to create a new database externally and load the data from the Excel sheet into it.

5.3.2.3 views.py module

This module is used for interacting between the front-end and back-end and it can be accessed from the "helloworld" folder. It is composed of different functions called "views" that are accept a request and return a HTTP response to an HTML page. In this module there are three functions: get, change_view and statistics_view.
The get function is used for showing the "Home" page when a user accesses the Jobs Observer. The change_view function is used for returning a HTTP response for "About Software" and "Plots" page. This function is executed when the request method is GET request. This GET request is represent as a dictionary with one value inside which indicates the name of the button in the HTML page. So, for this function the names are "aboutSoftware" and "plots" which represent the names of buttons in HTML.
The statistics_view function is used for displaying statistics requested by the user from plots page. The principle of execution is the same as for the previous function but instead directly to return a HTTP response for an HTML page, some operations have to be performed in between for creating a statistic HTML page. At the end a HTTP response for the creating HTML page is returned. The operations performed are the operation from the stats.py module. Let's take an example. A user request for a statistic, called "Jobs Per Country". This request will be sent to views.py module and will pass this request through all the functions. The value of GET request is "displayPlot1" and if this value is true for a certain "if" method such if request.method == 'GET' and "displayPlot1" in request.GET then some operations from stats.py will create a statistic which is used for returning a HTTP response.

5.3.2.4 urls.py module

This module is the last module before displaying a HTML page. It is composed of a list of urls where each name of url is associated to a function from views.py module. As example, the url called "AboutSoftware" is associated to change_view function. Notice that the url "AboutSoftware" is not a full url but a part of url which is considerate as the continuation of the basic url s.t basic url + "AboutSoftware".

5.3.2.4 Project on AWS

Jobs Observer web-application is already deployed on AWS. The deployment is realized by putting the project on github from where the AWS take the source code and performed a build and deployment itself.

5.4. Assessment

In this section, the satisfaction of technical deliverable with respect to the requirements are compared and evaluated. As previously, the front-end deliverable and back-end deliverable evaluated based on requirements.

5.4.1. Front-end deliverable. The front-end deliverable which is the GUI interface of our web application satisfies all the functional requirements. The accomplishment of the first requirement can be seen on the figure(number) which shows that the GUI contains some basic elements of a web page. Additionally, the created page is a static page and contains some buttons that allow the creation of statistics.
The second requirement is also satisfied because the buttons for displaying statistics are connected to the python script and are operational. The front-end deliverable also satisfies all the non-functional requirements which consist to have a friendly interface and a good readability. For the first requirement, a simple interface is created for avoiding to

user to spend a lot of time for learning because based on the names of buttons and titles, he would be able to predict the result in advance.
For the second requirement, the GUI interface doesn't overload the user with a lot of information which increase the readability. Additionally, the font-size is adequate because it allows to see all the names of elements and to read all the text sections.

5.4.2. Back-end deliverable. In this section, a comparison between the back-end technical deliverable produced and the back-end requirements is realized. The objectives are to identify, if all the functional and non-functional requirements are satisfied.

5.4.2.1 Satisfaction of functional requirements

The first functional requirement is satisfied. It consists to use Django for creating the Jobs Observer and to ensure the compatibility of Jobs Observer application with the AWS server. The technical deliverable is based on Django framework and is already deployed on AWS server. Additionally, all the operations and user interactions are available and can be performed.
The second functional requirement is also satisfied. It consists to import the database and to reuse the same structure in Jobs Observer from the previous software. Jobs Observer use the same structure for the imported database. The imported database isn't included directly to Jobs Observer but it is downloaded from the external cloud server to the empty database file.
The third functional requirement is partially satisfied because the Jobs Observer contains all operations from previous software but not all of them are operational and some more changes have to be performed for reaching the satisfaction of the whole requirement. Nevertheless, some operations are available and provide as results the created statistics.
The fourth functional requirement is satisfied. Some operations at back-end can be call for execution from the GUI. As the result for each operation, a single statistic is displayed.

5.4.2.2 Satisfaction of non-functional requirement

The first non-functional requirement is partially satisfied. Indeed, it is difficult to evaluate if the Jobs Observer is maintainable. The Jobs Observer is not yet a final product and can be improve. For simplifying further development, a structural approach is and a meaningful naming for operations and variables are used.
The second and the third non-functional requirement are satisfied. From the back-end functional requirements, it is possible to conclude that the back-end of Jobs Observer is operational and compatible with the AWS server.

Acknowledgment

The authors would like to thank the BiCS management and education team for the amazing work done.

6. Conclusion

The conclusion goes here.

References

[BiCS(2018a)] BiCS Bachelor Semester Project Report Template. https://github.com/nicolasguelfi/lu.uni.course.bics.global University of Luxembourg, BiCS - Bachelor in Computer Science (2017).

7. Appendix

All images and additional material go there.