The University Of Sheffield.

COM3504/COM6504 Intelligent Web

Assignment 2017-2018

This assignment is primarily concerned with applying the ideas that are being presented in the module on methods for accessing the Web and making sense of its content. Some of the basic algorithms needed for it have already been introduced during the module, and can be obtained from MOLE.

Organisation of the Assignment

The assignment is divided into two parts:

1. **Part 1**: Building a system to query twitter.com controlled via a Web interface.
    a. No marks: after feedback the revised Part 1 will have to be submitted as part of the Final Assignment. Penalty for not handing in a reasonable solution: up to 25 marks taken off the final marks. This will include the implementation of **at least parts 1.1, 1.5, 1.6 and associated documentation (and report).**
    b. Deadline: **Friday 31.3.2017 at 23:59:59**
2. **Final Assignment**:
    a. 100% of the marks
    b. Deadline: **Friday 12.05.2017 (Week 11) at 23:59:59**

All submissions are electronic via MOLE. No need to hand in anything at reception.

Workload

The whole assignment is intended to account for between 20 and 30 hours of each person's work towards the module as a whole. Experience has demonstrated that it is not really possible for one person to do all the work for the assignment in that time, unless they are an exceptionally competent programmer with previous knowledge of Web programming. Therefore, the assignment is organised on the basis that people should work on it in **groups**. Groups must be composed of a **maximum of 3 members.** Students are allowed to do the assignment in pairs or on their own.

Please note!
You can only have groups within the same module code, i.e. members must be either all in COM3504 or in COM6504.
PhD students should ask the lecturer before joining a group.
You must register your group at
https://drive.google.com/open?id=1hZ2VRFcGg55Z7icDs8Xqml5c48Bm9UIgNsV7R7D1Emg&authuser=0

Given a group, you will normally be expected to keep with that group for the whole of the assignment. However, if any problem arises that makes this difficult, you should notify the lecturer **immediately**, so that appropriate action can be taken. See lecture 1 slides on this topic.
It is important that you make clear what contribution each group member has made. It is required that each person contributes to each stage of the assignment, but it is up to each group to decide how to divide the work up between individuals. This must be explicitly stated in the report, as explained below.
Only one member of the group needs submitting on Mole. **However,** in case just one person submits, the entire group takes responsibility for that submission. If unsure, we suggest that all

members submit.

Deadlines

The deadline is **absolutely fixed**. You should therefore plan your work to aim at handing the report in at least a few days before the deadline – do not leave it until the deadline, just in case any minor thing goes wrong and you then find that you are late.
Note that the Computer Science department applies fairly severe penalties for handing coursework in late. If you want to look at the details you can find them on the University's website.

Material Provided

Lecture notes, lab class examples and websites/books to use.
NOTE: no third party code can be used in the assignment, except what has been **explicitly** provided in the lectures or lab classes. For example you are allowed to use code given in the lecture slides but you are not allowed to download any code from the Web or to use any other software that will perform a considerable part of the assignment. **Unauthorised re-use of third party software will be considered plagiarism.** In case of doubt ask the lecturer for permission before using any third party code. Libraries allowed, despite not being mentioned in the lecture notes are css/js libraries to improve the look and feel of any interface (e.g. Angular, Bootstrap). For other libraries, please ask before using.

1. Scenario

The goal of the assignment is to build a website and a progressive app for finding restaurants and for reviewing/rate them. A user can also create new pages for restaurants when they do not exist. Examples of these types of sites exists, for example TripAdvisor.com. You must build a client server architecture that enables searching for restaurants using a computer or a mobile (so you must use a Responsive Web Design on the client). The architecture will allow to:

- Search for restaurants by keywords and by location
- Create new restaurants descriptions inclusive of photos and description, address, etc.
- Rate and write reviews

1.1.1. Searching for restaurants and displaying results (15%)

The search for restaurants can be done via

- Keywords and/or schema.org tags: the user must be allowed to input a series of terms through a form (e.g. free keywords, type of cuisine, postcode, etc.). Conditions in the form are in AND. The system must be able to search those terms in a database and return a list of restaurants most relevant to the query
- Map: starting from the user location, the system should query the database and return the results on a map only for the restaurants in a user-defined radius (e.g. 1, 2, 5, 10 km)

1.1.2. The restaurant page (15%)

The results presented for the query (list of restaurants or map) should allow accessing the restaurant page containing all the relevant information, e.g. address, type of cuisine, and official photos as well as photos uploaded by customers, a global customer rating for the restaurant, reviews, a small map etc. A simple solution will provide a simple output while a sophisticated solution will allow further navigation from the page, e.g. the small map could link to a bigger map showing the user location (maybe sending to Google Maps for directions), the type of cuisine could allow finding other restaurants providing the same cuisine, etc.

1.1.3. Allowing restaurant reviews (5%)

The restaurant page must allow logged in users to add ratings, reviews and photos to the page. These must be uploaded to the server and be seen by other users as well on the restaurant page. Note that if the user is not online at the time of posting, the review must be saved for later sending.

1.1.4. Creating a new restaurant page (15%)

Logged in users must be able to add new restaurant pages to the system. The information required will be at least address, type of cuisine, reviews, and one official photo. A basic solution will provide just a form while a sophisticated solution could help the user with filling in the information e.g. via address searching.
The official photos can be uploaded from the file system (e.g. a phone's library) or taken directly from within the page (e.g. if on mobile but also on a computer). The photo taking must be implemented using WebRTC (as opposed to a simple HTML5 input/camera field).
Note that if the user is not online at the time of posting, the new page must be saved for later sending.

1.1.5. The Server Architecture (20%)

The server must be implemented in Express/NodeJS and must use both Ajax <u>AND</u> socket.io for the different functions. This means that a simple direct POST from an HTML form is acceptable; some functions will be implemented using Ajax, others will have to be implemented in socket.io. The goal here is for the students to show that they master both techniques. Data communication must be implemented using JSON.

1.1.6. The Database (5%)

The database of restaurants must be implemented using MongoDB and contain all the information relevant to each restaurant; it must allow the searches above.

1.1.7. A Progressive App (10%)

The website must be built as a progressive app, able to cache pages and to let the user access the most recent pages retrieved (i.e. the main pages, the last restaurant visited, etc.). The search will not be allowed when offline but the user must be told that they are offline (i.e. the app must present a clear message and of course must not crash).

1.1.8. Quality of the solution/ keeping your solution manageable (15%)

The assignment per se is rather simple, as it follows the module's lab class exercises. However, we require a quality solution. While a simple solution will attract some good marks, a first class solution will have to provide sophisticated features. Some have been mentioned above.
Please note that the quality is not necessarily related to a stylish design (although that helps) or a high number of functionalities (ditto).
The quality must be intended as related to the module's learning objectives, which are to learn the use of sophisticated client/server architectures, using a range of methods such as login management, Ajax and socket.io, where the client is specifically designed for users on the go (hence the progressive app requirement), uses WebRTC, etc. The server must be scalable to potentially thousands of users, so your solution must be non-blocking on the server side.
Having said that, please note that the assignment is open ended in some respects. Implementing a perfect solution would be far beyond the scope of this module. Make sure to keep the solution manageable in the time allocated to the module. Do not overdo it.

2. Marking schema

Each part described in subsections 1.1-1.6 will carry marks divided as follows:

36% for the documentation in the project report. More details on the detailed marking schema can be found in appendix A.
39% for the implementation of your solution (quality of code and Javadoc-like documentation). See appendix B for further details on the specific marking schema.
25% for the correctness of results.

Please note

- the direct consequence of the marking schema is that providing a program returning the correct solution is not enough to get a pass mark. You will need to implement the correct strategies and document/discuss them properly! Quality of documentation and code are very important issues for computer scientists.
- the weights makes so that getting marks over 75% requires to be creative and go beyond what strictly required by the assignment: we expect that only exceptional solutions will get marks above that threshold.

3. Handing in

Your solution must be contained in a self-contained directory called *IntelligentWeb* (`<MainDirectory>` in the following) compressed into a zip file submitted through MOLE.
The directory must contain:

1. The code of the solution (please note that we will both inspect and run the code).
   (a) All code must be **developed in HTML5/EJS /Javascript/CSS**. We must be able to run your solution without problems on a standard lab machine. It should not need any application to be installed in order to run. If in doubt about using a library or not, be sure to ask the lecturer for instructions.
   (b) You are required to use node.js and to make sure that your solution runs on the lab computers.
   (c) All the external libraries must be included in your solution. Some css and js libraries can be provided with external links that are uploaded automatically when pages are loaded (e.g. you do not need to include JQuery, just link it in your HTML/EJS files). The node module libraries do not need to be included. Just make sure to create a complete package.json file so that we can install all the modules running *npm install*.
   (d) Source code; please provide:
      1. All the code should be in the directory **`<MainDirectory>/solution.`** Please note that the quality of the code carries a relevant portion of marks, so be sure to write it properly.
2. A report documenting your work. The report must be contained in the directory **`<MainDirectory>/report/.`** An outline and set of requirements for the report are provided in Appendix A.
3. The documentation in the Javascript/HTML file must be of very high quality. We expect the same type of quality that would enable generating a complete Javadoc documentation from a Java file. Please note that this documentation carries a relevant portion of marks, so be sure to write it properly. For more information on guidelines for this type of documentation see http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html
4. Screenshots or video of the app so that we can understand what should happen when we run your solution **`<MainDirectory>/screenshots`**
5. The filled self assessment form.

3.1. *How to submit*

Everything must be submitted electronically. Nothing is to be handed in at the reception! **Use MOLE**. Store your solution in a .ZIP file that when unzipped will generate the directory organisation as described above. As emergency measure (and only in that case!), if any last minute issue should arise in handing in electronically, please send your solution by email to the lecturer (cc to demonstrators) in a self contained .ZIP file.
Make sure never to leave your solution in the public_html (or any other public directory) as it could be copied by others. Solutions not working on the departmental computers (e.g. working only on personal computers) will not be considered.

3.2. *Anti-cheat measures*

Please note that measures are in place for detecting plagiarism and in general cheating.

4. Queries about this assignment?

Should you have any queries about the assignment, feel free to contact

Fabio Ciravegna, Temitope Adeosun, Abiel Tomas Parra Fernandez:
{fabio, tope.adeosun, atparrahernandez1 }@ shef.ac.uk

Appendix A: Report Outline and Marking Schema

It is important that you produce a high quality report. Be sure not to leave the report at the very last minute. It is importance that you do not ignore techniques and examples provided to you during the lectures and lab classes. Referring back to them during your planning/implementation stages will give you the base from where to start, as well as a point of comparison/discussion where your ideas differ from what already presented to you (i.e., do not reinvent the wheel – if it has been done before, reuse it and complement it where it lacks functionality).
You may use diagrams to aid your explanation in these sections, but please note that a diagram alone is not acceptable and must be accompanied by an explanation/discussion.

Your report must be organised according to the following outline, which is designed to help you ensuring that all the required information is provided. **Length: maximum 5 pages (3 pages for the first part).**

Index Table

[No marks – OPTIONAL] - You may or may not want to include an index table on your report. This is completely up to you. This does not count against the page limit.

Introduction

[no marks] – This should give the marker a guide as to what to expect from your report. Please make the marker's work easier by being honest.

Diagram

Include a diagram explaining how your solution is organised, showing interconnection (which page leads to what pages, which connections require logging in and which ones do not, etc.) Also show where the information is stored (e.g. the database), where it is cached, etc.

For each task in section 1

Create a subsection in your report **for each of the requirements** in sections 1. It is very important that they are documented individually by explicitly following the organisation below.
Solution:
Design and its motivations: [55% of the report marks for this section] explain how your solution works and explain why you chose to design your solution in this particular way (e.g. to optimise number of twitter queries). Does it have advantages/disadvantages over other design choices?
Issues
[10% of the report marks for this section] **-** Introduce the task this section refers to and the challenges that you were faced with.
Requirements: [20% of the report marks for this section] how does this design comply with the requirements specified in the original assignment sheet? Are you meeting all requirements?
Limitations: [15% of the report marks for this section] have you thought about exceptional situations that may limit your solution? Is your solution extensible? Can it be easily adapted for other requirements? Remember that no design is flawless!

Conclusions

[no marks] - You should include here any relevant conclusions you've collectively arrived at regarding the process of designing the solution for this part of the assignment as well as any lessons learned.

Division of Work

[No marks – MANDATORY] – Have all the group members shared the workload in a balanced way? In what way? E.g. what has each member provided – be precise **and especially make sure that this is completely agreed by all group members**!
For example (note - the sections listed in the declaration below are randomly chosen and may not correspond to an equal development effort):

All the members of the group contributed equally to the assignment solution. The solution was designed jointly and then each member lead the implementation of one specific part of the code, its associated documentation and contributed to the writing of the final report.
In particular:

- *Member X lead the implementation of 1.1, 1.2 and 1.3 and wrote the corresponding documentation*
- *Member Y lead the implementation of 1.4, 1.5 and associated documentation*
- *Member Z lead the implementation of 1.5, 1.6, 1.7 and associated documentation*

The final document was jointly edited.

Make sure to list clearly what each member has done.
If the group developed the entire project together and nobody contributed particularly to a subset of sections, feel free to write so.

Extra Information

[No marks – MANDATORY] – This section should include any extra details needed to run your code. If no extra configuration is needed, please explicitly say so in this section.

Bibliography

[no marks] **-** Do not forget to cite any sources and reference these within the text where appropriate (e.g., "(...) *we have used the techniques as per [1]."*).Make sure to use a standard format style. The lecture notes must indicate the lecture number (e.g. week 1)

Appendix B: Marking schema for the Code

The quality of the code of your submission for each part of the assignment will account for quite a large part of the marks. It will be marked according to the following schema:

[65% of the code marks] – Code functionality: how the code meets the requirements set in the assignment description.

[5% of the code marks] - Code documentation. This includes

- in-line commenting to make your code intentions clearer to someone reading
- Javadoc-like documentation: higher level comments on the code and its use. It is important that you note the different level of detail generally included in Javadoc-like comments as compared to the kind of comments that are included within your code!

[15% of the code marks] - We must be able to run your code without any problems using the lab computers.

[5% of the code marks] - Your code should follow a consistent format regarding class and variable naming as well as indentation, this is can be easy achieved with the use of an IDE (Eclipse, IntelliJ, Netbeans, etc.).

[5% of the code marks] - Proper use and handling of Exceptions in your code.

[5% of the code marks] - Code organisation (e.g. readability, use of modules in node.js, Javascript files, etc.).