COM3504/COM6504 Intelligent Web

Assignment 2019-2020

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.

Organisation of the Assignment

The assignment is divided into two parts:

1. **Part 1**: Building a fully-fledged PWA and associated remote nodejs server.
   a. No marks: after feedback the revised Part 1 will have to be submitted as part of the Final Assignment.
   b. Note: submission for this part is not mandatory. It is your opportunity to receive constructive feedback; you are not required to take it. However you will only receive feedback if you have completed all functionalities, as we will not be able to provide meaningful feedback on half-baked solutions.
   c. Deadline: **Wednesday 1st of April 2020 at 23:59:59** (week 8)
2. **Final Assignment**: add a MongoDB database and a recommender function to the revised part 1
   a. 100% of the marks
   b. Deadline: **Thursday 7th of May at 23:59:59** (week 12)

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

Feedback about part 1 will be given on Thursday 2.4.2020 and Friday 3.4.2019 (week 8). Note that that week is the last before the Easter break. If you are planning on going home, make sure to plan accordingly, as we will provide feedback until late on Friday afternoon.

Workload and group working

The whole assignment is intended to account for between 20 and 30 hours of each person's work towards the module as a whole. The assignment is organised on the basis that people should work on it in **groups**. Groups must be composed of **3 members.** Groups are left to self-organisation. Students have been advised since week 1 to form a group and have been given instruments to find suitable partners.

Please note!
You can only have groups within the same module code, i.e. all 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
The same link provide help in finding a suitable group

Given a group, you will 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 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 among individuals. This division must be explicitly stated in the self-assessment form where the expected distribution is described.
All members of each group are required to submit on Mole.

Learning from the module Vs developing a software solution

It is important for your solution to follow the patterns and material taught during the lectures. Solutions that adopt different approaches will attract low marks and in some cases zero marks. The goal of the assignment is to prove that you have learned from the module, rather than to prove you are able to write code that works.

Deadlines

The deadline is **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.
Please refer to the slides of the week 1 lecture for risks and considerations on working in groups.

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. Bootstrap). For other libraries, please ask before using.
List of allowed libraries:

- CSS/Javascript: bootstrap
- NPM Libraries: Passport, Multer
- Code used in the labs and lectures

Examples of NOT allowed libraries:

- Angular: reason: it implements Ajax and the communication with the server in proprietary way. This is considerable part of the assignment work
- Any language that builds on top of Javascript and e.g. requires compilation

1. Scenario

The goal of the assignment is to prove your knowledge of:

- client/server architectures
- nodeJs and express
- no-SQL databases (via the use of Mongo DB)
- Progressive Web Apps and Web Workers
- Client side data storage (localStorage, Indexed DB, etc.)
- Flexible client/server architecture (Socket.io, Ajax, etc.)
- Use of recommender techniques

In particular an excellent solution will be able to compose the topics above in harmonic way to provide an effective service and pleasant user experience.

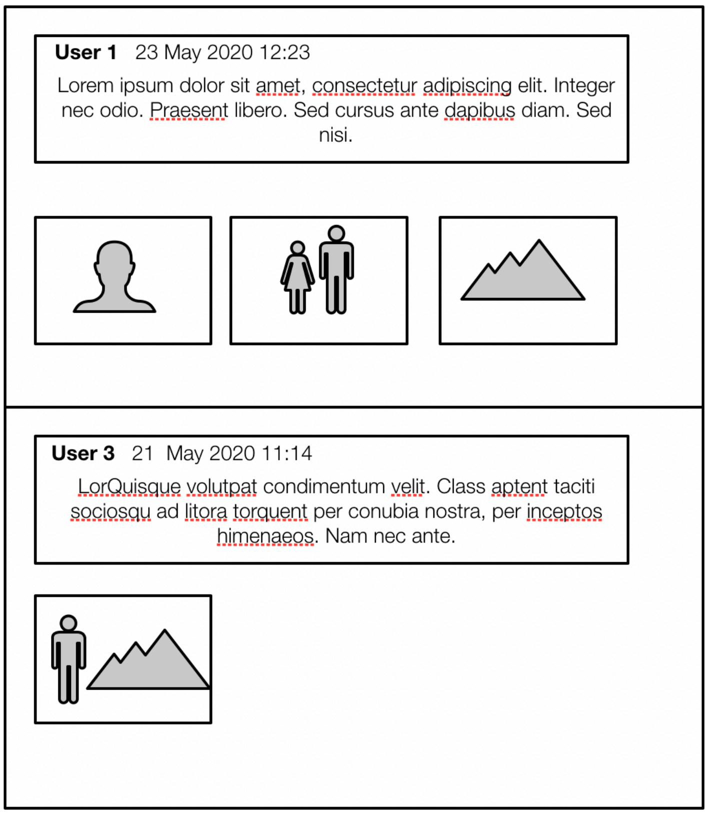
The assignment concerns building a social media platform. Users will be able to create stories (text and multiple images) and post them on the platform. They will also be able to access stories written by others. Users will be able to like or dislike stories using a 0-4 marking schema (0= dislike, 5=love). They will see the stories in an order tailored to their interests.

1.1. Building the PWA

Build a Progressive Web App which allows all users to:

  - Post stories including text and multiple images
  - Their stories are displayed on their *personal wall* where they can also see the likes left by others
- AND
  - Read the stories created by others which will be presented on a *stories wall*:
    - in chronological order
    - in order of preference as suggested by the recommender system (section 1.3)
  - Like or dislike a story written by others (range 0-4).

Stories will be presented on both walls as a list of items e.g.:


User 1 23 May 2020 12:23
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi.
User 3 21 May 2020 11:14
LorQuisque volutpat condimentum velit. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nam nec ante.

Photos
The PWA will allow to add photos to the stories either by taking them with the camera or by uploading existing photos from the file system or a phone's library.

Stories
When displayed, stories will show:

- pictures – you may impose an upper limit to the number of pictures that can be attached to a story, e.g. 3 pictures
- text – you can impose an upper limit to the text length – e.g. 150 characters or 30 words
- date and time
- author

The PWA must have the following characteristics:

- It must use a web worker to allow working while off-line (a common mobile experience). The offline mode will have to work for both querying and storing stories/likes
- It must store data locally into an IndexedDB structure
- It must sync with the remote MongoDB server when online
- It must never leave the user waiting while data is retrieved from the server; i.e. the web worker supporting the PWA will have to use a strategy like stale-while-fetching or race-between-network-and-cache.

1.2. Building a nodeJS server collecting data from the PWA

Build a server collecting data captured by the PWA. It will allow syncing with the PWA – for example if user A adds a story to an event, this story should be made available to all users of the PWA when they access the site.
The server must be backed by a MongoDB database to store and search the information.
The server must be implemented in Express/NodeJS and must use both Ajax AND socket.io to

communicate with the client. This means that simple direct POSTs from an HTML form are not acceptable; some functions will be implemented using Ajax, others will 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.
The server must be scalable to potentially thousands of users, so your solution must be non-blocking on the server side.
The organisation of the MongoDB data is left to you. However please note the suggested organisation in section 1.3.

1.3. Recommending Stories

When the user is online, the wall will display new stories (if available). When offline it will just show the stories already cached. The user will be able to choose whether to see the stories in chronological order or in an order suggested by a recommendation algorithm. The algorithm will use the data of all users and their likes, as well as the likes of the specific user to provide recommendation. It will adopt an item-based recommendation strategy (i.e. stories will be suggested on the basis of the stories liked so far).
You are being provided with a library of *n* users and their stories, as well as the likes given to the stories. You must develop a recommender algorithm able to suggest the correct suggestions for a user given their own personal history. You are required to use the methods described in the lectures.
The library of users and stories will be contain:

- a json array of stories {storyId, userId, text}
- a json array of users including their likes {userId, [{storyId, vote]}

I suggest that you use this organisation in your MongoDb so that you can upload the files directly.

1.4 Quality of the solution/ keeping your solution manageable

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 mentioned above.
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.
The point of the assignment is to demonstrate that you master the techniques introduced by the module. Designing a beautiful web site with lots of functionalities for an amazing user experience may be tempting but is pointless.
The number of marks that that will attract in terms of quality will be not be worth the amount of time that will cost you.
I suggest instead that you spend your time working on the quality of the core solution and its documentation.

2. Marking schema

Each part described in the subsections above will carry marks divided as mentioned above.

Marks for the different sections:

1. PWA: 45%
   - o general organisation: 5%
   - o posting and displaying stories: 5%

- correct use of a web worker: 10%
    - caching data on the client (IndexedDB): 10%
    - working both offline and online: 5%
    - non blocking interface when displaying stories: 5%
2. NodeJS server:
    - Correct organisation of server and routes: 15%
    - non-blocking organisation, multiple dedicated servers to free up the routes' thread: 10%
3. MongoDB:
    - Correct use of the MongoDB including correct organisation into models, controllers, etc. 10%
4. Client/server communication
    - Correct use of Ajax and socket.io: 10%
5. Recommending stories:
    - Appropriate and correct recommending strategy: 15%

The quality of the code of your submission for each part of the assignment will account for quite a large part of the marks. We will check:

- Code functionality: how the code meets the requirements set in the assignment description.
- 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!
- Code runnability: we must be able to run your code without any problems using the lab computers.
- Code quality**:** 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.). It must contain proper use and handling of Exceptions.
- Code organisation (e.g. readability, use of modules in node.js, Javascript files, etc.).

3. Handing in

Your solution must be contained in a self-contained directory named after your group (this must be the same name you have registered with on the Google form (***`<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 special installation of any external library other than running *npm install*.
    (b) You are required to use node.js/express and to make sure that your solution runs on the lab computers.
    (c) The external node modules must NOT be included in your submission. Just make sure to create a complete *package.json* file so that we can install all the modules running *npm install*. Some css and js libraries can be provided with external links in HTML/EJS files (e.g. you do not need to include JQuery, just link it in your HTML/EJS files).
    (d) Source code:

   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 read me file clarifying any installation/running instruction. The task should be easy and self-explanatory but some instructions may help ***<MainDirectory>*/README.**
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 project. 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 stating what functionalities you have implemented and what is your confidence of having done an excellent piece of work. It is important that you make clear what contribution each group member has made to the solution.

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.
Solutions not working on the departmental computers (e.g. working only on personal computers) will lose considerable marks.

Please note: only one member of each groups is required to submit the solution to the assignment. However: as submission is via MOLE, we will only be able to return the marks to those who have submitted. If the division of roles in the group is balanced, all the members will get the same marks.

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, Matthew Barber, Jack Deadman
{f.ciravegna, m.g.barber, jdeadman1}@ shef.ac.uk

T