COM3504/COM6504 Intelligent Web

Assignment 2021-2022

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. In providing a solution, you are required to use the methods and techniques taught in the module.
The whole assignment is intended to account for between 20 and 30 hours of each person's work towards the module as a whole.

Scenario

You are supporting a unit of secret agents who need to share relevant information about specific missions and their context. You are asked to implement a web site and the supporting server infrastructure. Using the website, the users can create reports, access them and comment on them.

The Interface should allow access to a list of missions, each of them composed by a sorted list of multimedia details. The system will allow:

- Visualisation of existing stories
- The creation of a new story.
    - Once inserted the story will not be modifiable
    - A story contains
        - a photo
        - a short text
        - the author name; this is just a string - no need to implement a login system
        - the date of issue
- Commenting/Annotating the stories

When a new story is created, it is accessible to all users (i.e. you do not need to implement a login system or a set of privacy rules – when the user enters the site they will see all the stories).
The stories are contained in a No-SQL database (Mongo DB).
In the next figure you can see how the list of stories could be displayed. The design is left to the students.

This is a live document. If you have specific questions please comment on it and then tag me (adding to your comment: +f.ciravegna@sheffield.ac.uk and assigning the task to me)

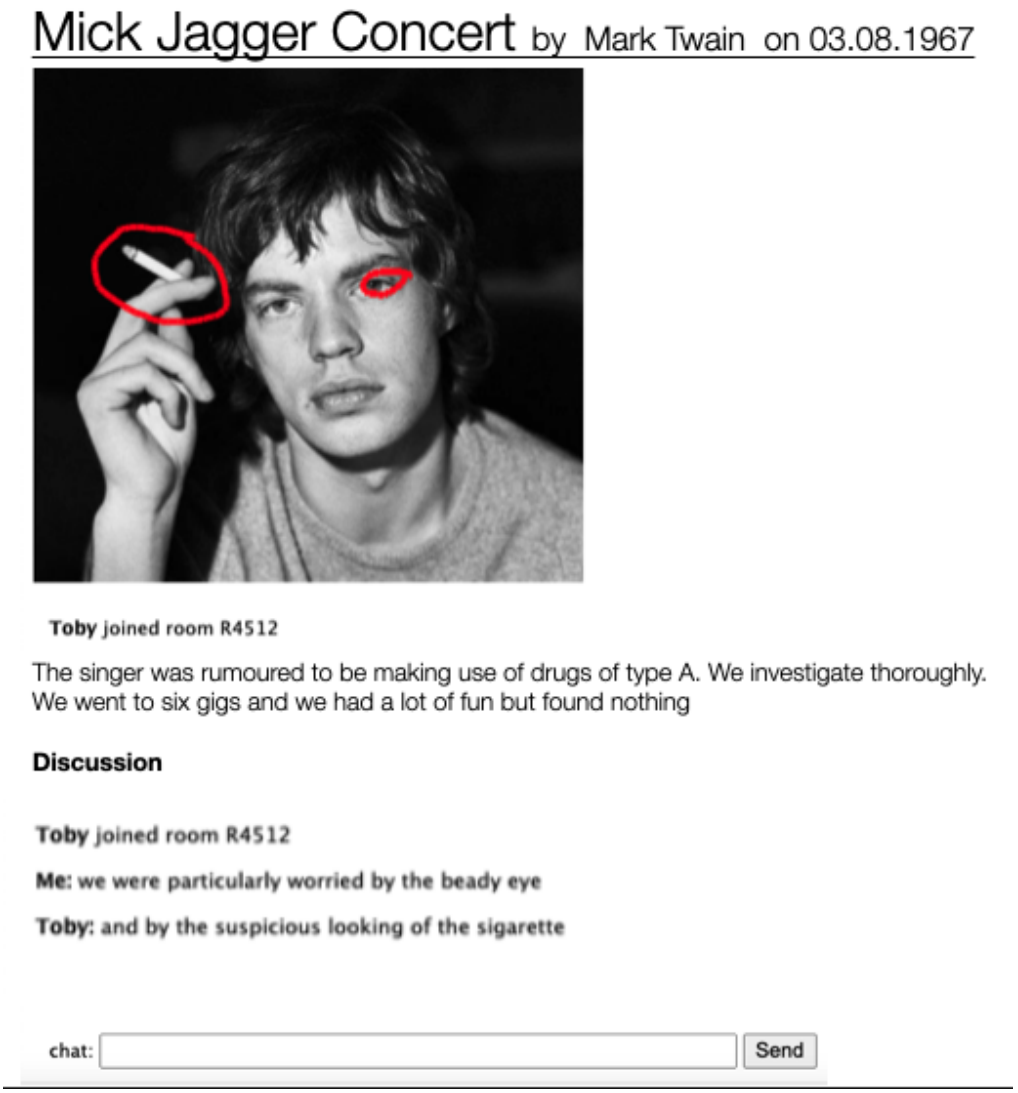
Don Valley Mission by Joanna Smith on 03.04.2022

We checked the Don Valley stadium looking for signs of any wrongdoing related to mission XYX. The report details the finding and the suggested actions.

Mick Jagger Concert by Mark Twain on 03.08.1967

The singer was rumoured to be making use of drug type A and therefore we investigate thoroughly. We went to six gigs and we had a lot of fun but found nothing

When clicking on a story, the story is opened. The story per se is unmodifiable but it will be possible to discuss about the findings in a chat with another person online (when the two users are in the same room at the same time) or simply using the chat to take notes (if you are in a room alone).



In the picture above the two users have annotated parts of the photo and made comments.

Requirements:

- The comments are implemented using a chat system.
- Images can be annotated using the mouse (e.g. by drawing lines) during a chat session.
- Both the chat text and the hand drawn elements are considered annotations, as the text may complement each other (so in the rest of the document I will use the term annotations to refer to both).
- The annotations and chat are interactive, so if more than one person is present in the room, all participants will see them in real time while they are made. Different users can draw on the same image at the same time. See explanation video.

There are different parts in the application and they are discussed below.

The Web app

- It allows writing and uploading reports;
    - As mentioned, stories are associated with a title, an image and a description, as well as the author's name.
- Reports can be ordered by their author’s name or date.

- A story can be selected for discussion by clicking on it.

How would you do that?

- The report editor can be implemented as a form where you provide text, a photo, etc.. The photo can be taken using either an HTML5 form or (better) WebRTC.
- The report must be sent to the database using Axios. Data (e.g. title, images, description and author) must be exchanged using JSON.
    - Images need to be transformed to base 64. Images are uploaded to the server and stored into a MongoDB database. The images can be uploaded from the file system or can be referenced using a URL. Even in the latter case, the image must be uploaded to the server because the risk is that if the image is changed online, then the annotations will no longer make sense.
    - Annotations are NOT uploaded to the database. They are kept locally (see below)

The chat and annotation interface

- The user selects a report and enters a chat room
    - The report is displayed
    - The chatroom is available
    - It allows annotating the image using the mouse.

You are provided with a starting point project that implements the drawing capability and provides a basic version of the chat interface. This will help you focus away from basic HTML/CSS issues.
Use socket.io to send both the chat text and the graphical annotations to the participants in the room.
The starting point project provides some hints (@todo) on where to send the data. This should help you understand how the provided code works.
The annotation must be sent and received in real time (see video).

Local Persistence

The annotations must be persisted in the participants' browser using an IndexedDB database. The following information must be cached:

- The whole story inclusive of image, text, title, etc. – this part will also be available in the server but they must be cached to allow working offline (see below)
- The annotations (i.e. annotations on the image and the chat content). These are NEVER to be sent to the server, as they are considered confidential to the participants in the discussion

When the user comes back to a story they have annotated, all the information above must be shown from the cache.
Notes on caching:

- If the user has chatted with different people, the user must be presented with a choice of which chat session to see.
- If a user joins mid-way a chat, they will NOT see the annotations provided so far. They will just see the annotations that are created after they have joined.
- If a user has not joined a chat while it was taking place, they are not allowed to see the annotations (this is why the annotations are never stored on the server)

Working offline

The web app must be able to access stories, display annotations and allow annotations both online and offline. So the list of reports is to be cached and so is their content.

- This means that when the user goes online, the first operation to perform is a refresh of the report database.
- The operation must be non-blocking, i.e. the browser must be working while the stories are refreshed
- As the stories cannot be modified or deleted, syncing with the Mongo DB means just downloading the new stories appeared since the last sync operation

When uploading a new story, if the device is offline, the story must be cached into the IndexedDB together with all the metadata (title, text, date, etc.). When the device goes online again, the story must be uploaded to the server's database via Axios.
Annotations from previous chats are saved in the indexedDB and displayed when the specific story is requested even when offline.

- It is possible to annotate while offline but the annotations are not shared with anybody.

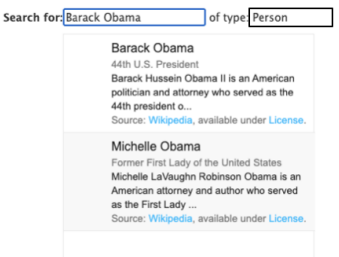
The server

The server must be implemented using NodeJS+Express and must support the chat system and the connection to MongoDb.

Working with the knowledge graph

Allow the user to annotate the images with information retrieved from Google's knowledge graph. For example, the Knowledge Graph Annotation could be used to provide information about a person in a picture or a location.
An example on how to annotate and present the information would be:

Annotation:



Presentation:

(you are free to choose how to actually present the information, for example you do not need to use a square image annotation, it is fine to use a hand drawn annotation)

As for all the other annotations, this information is to be cached and displayed when the user accesses the story again.
The Knowledge Graph returns JSON-LD. As it is not easy to read JSON, the information should be displayed using a human readable structure, e.g. via a table.

- The information should be stored into the IndexedDB in the original JSON-LD format.
- It is fine to never update the cached version of the information retrieved from the knowledge graph.

Documentation

The server documentation must be provided via Swagger.
The client JavaScript documentation must be provided in Javadoc-like style

Groups, division of work and associated documentation

The 3 team members should share the development equally. I expect the following parts to be developed by the following team members:

- Member 1: socket.io, nodeJS server, the chat/annotation interface
- Member 2: service worker, MongoDB
- Member 3: IndexedDb, Axios communication, Swagger documentation

Exceptionally, if your group is composed of only **two members (you must be explicitly allowed by the lecturer to form a group of two!!)**, the implementation of the MongoDb and the use of Axios are not required.

- Member 1: socket.io, service worker, the chat/annotation interface
- Member 2: nodeJS server (excluding socket.io), IndexedDb, Swagger documentation

If your group is composed of **only one member (again you must have an exceptional circumstance for that and be allowed explicitly by the lecturer)**, you are required to do the following:

- socket.io, service worker, the chat/annotation interface, nodeJS server, IndexedDb

Note that the above are just some of the functions required to develop a solution. The remaining functions must be shared fairly by the group.

Github Repository

The working of the group must be documented using a private **Github repository**. I expect the regular commits by each member to be clearly visible, so that it is clear who has developed what and when. I expect to see the programme build progressively over time in the repo.
With regular commits I mean very often – possibly daily.
The point of submitting the Github repository is to demonstrate that you have developed the code yourself over time. So its management, i.e. its commits are expected to show just that. Any large blocks of code suddenly appearing on the repository will be investigated for the use of unfair means.

Group Management

You must register your group on Google Drive. The same link provides help in finding a suitable group until week 2.
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.

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.
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 Blackboard.**

Deadlines; formative and summative assessment

The deadlines are 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.

There are two deadlines:

- **Monday of Week 9 at 12:00:00 for part 1**
- **Friday of Week 12 at 23:59:59 for the entire assignment**

Formative feedback will be provided in 1:1 meetings in week 9.

Submission of part 1 is non-mandatory. However if you submit, your solution must cover everything listed above except:

- MongoDB (just find a default solution to save the material sent to the server, e.g. keep it in memory in a list of objects)
- Swagger documentation
- The connection with the knowledge graph.

The final submission will include the entire system; you will have to resubmit also a revised version of part 1. This is why it is formative feedback: it is designed to help improve your solution.

Material Provided and external libraries

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 any code given in the lecture slides or in the lab classes 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, Express, node static, body parser, etc.
- All code and any libraries used in the labs and lectures (e.g. socket.io, Axio,JQuery, etc.)

Examples of NOT allowed libraries:

- Angular, React, React Native, Vue
- Any languages building on top of Javascript and e.g. that requires compilation

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.
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. Designing a beautiful website with lots of functionalities for an amazing user experience may be tempting but is pointless. The number of marks that will attract in terms of quality will 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.

The point of the assignment is to demonstrate that you master the techniques introduced by the module. 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.

Marking schema

Quality of the solution attracts 33% of the marks. Quality of the documentation attracts 33% of the marks, correctness of results attracts 33% of the marks. That means that having just a low quality solution providing correct results may be insufficient to get a pass mark.

Marks for the different sections:

1. Web app: 35%
    - working both offline and online
    - implementing a web worker
    - implementing an indexedDB
2. NodeJS server: 20%
    - Correct organisation of server and routes
    - Correct Swagger documentation
    - non-blocking organisation (use of promises and callbacks, multiple servers, etc.)
3. MongoDB: 15%
    - Correct use of the MongoDB including correct organisation into models, controllers, etc.
4. Client/server communication 15%
    - Correct use of Ajax and socket.io
5. Connection to the knowledge graph 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 and Swagger documentation: higher level comments on the code and its use. For more information on guidelines for code level documentation see http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html
    - The full history of the development via Github
- Code runnability: we must be able to run your code without any problems using normal computers.
- Code quality**:** your code should follow a consistent format regarding class and variable naming as well as indentation, this can be easily achieved with the use of an IDE (IntelliJ). It must contain proper use and handling of Exceptions.
- Code organisation (e.g. readability, use of modules in node.js, Javascript files, etc.).

Submitting

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 Blackboard.
All the code should be in the directory <MainDirectory>/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 machine. It should not need special installation of any external library other than running npm install.
(b) The external node modules must NOT be included in your submission so to avoid to submit very large zip files which will cause issues especially close to the deadline. 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).

Please note that the quality of the code carries a relevant portion of marks, so be sure to write it properly. **In particular we expect you to organise your code along the same lines used in the labs**. For example, the organisation of the Mongo DB code must follow the organisation provided in the lectures (controllers, modules, etc.)

Moreover you are expected to submit:

1. A README file clarifying any installation/running instruction. The task should be easy and self-explanatory but some instructions may help <MainDirectory>/README.
2. The Swagger documentation <MainDirectory>/Documentation
3. A few screenshots or a video showing the app so that we can understand what should happen when we run your solution <MainDirectory>/Screenshots
4. 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.

Finally you are required to share your Github repo with **fabcira**, **adeshtk** and **pegasus0642.**

How to submit

Everything must be submitted electronically via Blackboard. Store your solution in a .ZIP file that when unzipped will generate the directory organisation as described above. As an 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.

Do not worry about being a few minutes late on the date of the deadline – no need to contact the lecturer at 00:05 to say that your solution was a few minutes late due to problems uploading to Blackboard – you will not be penalised for a few minutes of delay.

If the division of roles in the group is balanced, all the members will get the same marks.

Anti-cheat measures

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

Queries about this assignment?

Should you have any queries about the assignment, feel free to contact
Fabio Ciravegna, Yunus Cogurcu, Jack Deadman
{f.ciravegna, yecogurcu1, t.adeosun}@ sheffield.ac.uk