COM3504/COM6504 The Intelligent Web

Assignment 2023-2024

Deadline: Fri, 17 May 2024 3pm

Handin: zip file via Blackboard (see Section 8 - Submissions).

1. Introduction

This assignment will test your ability to create a web application using the methods and techniques taught in the module.

It will test your understanding of the various components of a client-server architecture, as well as your coding skills in Node.JS.

This is a group project and groups are assigned by the module leaders. The work you submit must be your own work and not plagiarised. You must **NOT** use any libraries that have not been explicitly recommended during the lectures, as we will regard this as **use of unfair means**.

2. Scenario

You will develop a "Plant Recognition" progressive web application that provides users with means to record and view plants and also to help with identification. You will need to develop this as a web application, with the supporting server infrastructure. Using the website, the users can add new plant sightings, view plants added by themselves or other users and also comment on the sightings.

The app should allow access to a sorted list of **plant sightings**. The system will allow:

- Viewing plants sorted (at least) by date/time seen, and whether identification is finished. Sorting by distance away is a 'stretch' goal (i.e. nice to have).
- The addition of a new plant sightings
    - Note that once added, they will not be modifiable
    - Each plant sighting contains (at least)
        - date and time seen
        - location (i.e. geolocation)
        - a (short) description of the plant
        - Plant Size (i.e. an estimate of the plant's size - height, and spread)
        - Plant Characteristics (i.e. Does the plant have flowers? Yes/No ; Does the plant have leaves? Yes/No ; Does the plant have fruits or seeds? Yes/No ; What is the sun exposure in that area? full sun/partial shade/full shade, what is the colour of the flowers?
        - an identification (which may be unknown or uncertain) - **this can be updated:**
            - name: the original user can supply a name for the plant and/or allow other users to make suggestions. If other users suggest a name and the original user is happy with the suggestion, they can approve the suggested name.
            - a status of the identification should be provided. The status can be *completed* or *in-progress*.

        - The identification should be linked to information obtained from the DBPedia knowledge graph. The information includes a common name/scientific name of the plant, an english language description and a URI (which should link to the DBPedia page describing the plant in detail).
      - a photo
      - the user's 'nickname'; as a string - there is no need to implement a login system
  - A chat/comment linked to each sighting which contains:
    - comments on the sighting, i.e. discussion and suggestions about what the plant is

When a new plant is added, 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 sightings).

3. Design

The design is left to you. A good approach would be for the plant detail interface to also work for adding new plants and updating the identification.

When clicking on a plant in the list, the individual details will be shown. For the original user who reported the plant, this will allow them to update the 'identification' detail. The other details are fixed after creation, but it must be possible to chat (in real time) about the plant. This chat is public to everyone who views the plant. The original user might also use the chat to add more detail/s.

4. Components

The following is a list of components that must be included.

4.1 The Web Application

- The web application **must:**
  - be progressive
  - support online and offline interaction
  - be implemented using EJS and JavaScript in a node.JS framework.
- The web application **can**
  - be multimodal (i.e. provide access via multiple devices, such as mobile phone, laptop etc).

FUNCTIONALITIES

- The web application must allow creating new plants with associated details - description, details, date/time, user nickname, location (which should come from geolocation or optionally selecting from a map) and an associated image (photo)
- The web application must allow viewing all plants with associated details - description, details, date/time, user nickname, location (which should come from geolocation or optionally selecting from a map) and an associated image (photo)

- Plants can be sorted by most recent and (stretch goal) by distance away
- Plants should be categorised and browse-able by types, i.e. with or without flowers
- A plant can be selected for more detail/chat by clicking on it.
- When **online**, the details must be sent to the database
    - Images likely need to be transformed to base 64. Images are uploaded to the server and stored in the MongoDB database. The images can be uploaded from the file system or can be referenced using a URL.
- When the user is **offline**, you must allow changes to be stored locally (for uploading later), including:
    - it must be possible to create (at least) one new plant in the browser (indexedDB)
    - It is strongly preferred for more than one (new) plant to be held offline in the browser
    - it must be possible to add chat messages to a plant the user has newly and previously added
- When the device is online again, you need to:
    - **firstly** upload local changes to the server - this should be a safe operation and should not need to consider any server changes
      Note that after this, you should be able to safely reload from the server
    - retrieve updates (i.e since last synced) to the list of plants (retrieving everything from scratch is a weaker solution). This should be new plants and (only) changes to the identifications.
    - you should also reload any chat messages related to the user's added plants
    - As a stretch goal, you should notify the user of new messages in their chats (i.e. for plants they added).

How might you do this?

- The plants addition can be implemented as a form where you provide text, a photo, etc.. The photo should be (at least) by uploading through an HTML5 form or (better) by taking a photo from the web camera - or phone camera (if you choose this option).

N.B. It is not a requirement to create a mobile web app, but you may wish to, since geolocation, camera and testing offline usage can be better on a mobile device. Note that Chrome allows you to choose to view web pages as if from a mobile device - choose developer tools then the icon highlighted here:

Also note that Chrome (by default) insists on HTTPS for geolocation except for localhost. You will be shown in the lab how to bypass this restriction for specific hosts, including for mobile Chrome usage.

4.2 The Chat messages

- The chat part of the application **must:**
    - be progressive
    - support online and offline interaction
    - be implemented using socket.io in a node.JS framework
    - be non blocking
- The chat application **can**

- be multimodal (i.e. provide access via multiple devices, such as mobile phone, laptop etc).

FUNCTIONALITIES

- When a user selects a plant, the (live) chat must appear (likely at the bottom
- Any new message must be visualised in real time, as it becomes available
- The user must be able to add new message/s which will appear in real time on other user's chats (when they have it open). As a stretch goal, users should be notified if someone has added a message to a plant they added.
- When online, these should update in real time.
- When offline, messages can be added to plants that the user added, and the chat will 'sync up' when online with new messages being shown and made visible to everyone viewing the plant detail
- Since messages are linked to a specific plant (eventually), you might choose to store the chat within the plant object in MongoDB
- Note that messages cannot currently be edited or deleted

How might you do this?

You will likely 'broadcast' (from the server) any new chat messages with a plant id', so the web app can choose which messages it needs to update - i.e. where the user has a plant open - and/or should be notified since they added the plant.

4.3 The server

- The server **must**:
  - be implemented using NodeJS+Express
  - must support the chat system
  - must connect to MongoDB.

4.4 Data Storage and retrieval

- Data storage **must** be implemented using
  - MongoDB as network DB
  - indexedDB as browser storage
    - Note: Using cookies or local storage is not appropriate for this module
- For the information linked from DBPedia
  - use fetch-sparql-endpoint module
  - retrieve the correct annotations from the DBPedia SPARQL endpoint
  - Do not store any of the information from DBPedia
    - the idea is to fetch the information in real-time so this feature should not work offline

6. 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, fetch, socket.io etc.
- All code and any libraries used in the labs and lectures (e.g. socket.io, Axios, etc.)

Examples of NOT allowed libraries:

- Angular, React, React Native, Vue - note that server/browser frameworks will not be allowed
- Any languages building on top of Javascript and e.g. that requires compilation

7. Marking schema

Marks for the different sections:

1. Web app and chat application: 40%
    - working both offline and online
    - implementing a web worker
    - implementing an indexedDB
2. Server: 15%
    - Correct organisation of server and routes
    - non-blocking organisation (use of promises and callbacks, multiple servers, etc.)
3. Server Data storage 15%:
    - MongoDB:
        - Correct use of the MongoDB including correct organisation into models, controllers, etc.
4. Asynchronous and Bi-directional client/server communication 15%
    - Correct use of HTTP requests and socket.io
5. Connection to the knowledge graph 15%
    - Correct use of SPARQL query to fetch the information from the DBPedia endpoint
    - Correctly parsing the JSON response from DBpedia and displaying it on the UI

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 typically 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
    - 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
    - Gitlab/Github history and source
- Code execution: 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 (WebStorm). It must contain proper use and handling of Exceptions. You should also include appropriate naming and code documentation, i.e. comments.

- Code organisation (e.g. readability, use of modules in node.js, Javascript files, etc.).

8. Submissions

Your solution must be contained in a self-contained directory **named after your group** 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/Javascript/CSS/Node.JS. 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 avoid submitting 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.

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.md file clarifying any installation/running instruction in the <MainDirectory>. N.B. Installation and running should still be easy/simple.
2. Code 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

9. Team Contribution

You will be using buddycheck to rate yourself and your team members on the following criteria:
1. Attendance and punctuality (to supervisor/your own team meetings, etc.).
2. Ability to work effectively with other team members.
3. Contribution to content and organisation of project deliverables.
4. Quality of contributions.
5. Timeliness of contributions.

Your final individual mark will be formed from a scaling factor that is applied to your overall team mark. The scaling factor is determined by your peer assessment and those returned by your team members. Note: The Department's Policy on Team-Based Assessments requires evidence to support an individual mark varying by more than +/-15% from the team mark; the evidence considered will include meeting attendance/notes as well as slack/github interactions.

10. Feedback

The deadlines are fixed. You should therefore plan your work to aim at handing the assignment in at least a few days before the deadline – do not leave it until the deadline, just in case something goes wrong and you then find that you are late.

Note that the Computer Science department applies fairly severe penalties for handing in coursework late. If you want to look at the details you can find them on the University's website.

If you would like formative feedback this can be given at all times during lab sessions. During week 8 and 9 we will provide you with appointment slots that you can book (1 per group) to demo the current version of the project and receive verbal feedback (no marks).

11. Queries about this assignment?

Should you have any queries about the assignment, feel free to contact us using the Discussion board or via email.