CS5003 — Masters Programming Projects

Assignment: P3 – Project 3

Deadline: Wednesday 24th April (Week 11) 21:00
Credits: 45% of the overall module grade

NOTE: MMS IS THE DEFINITIVE SOURCE FOR DEADLINES AND CREDIT DETAILS

You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

Aims

The main aim of this project is to apply the experience gained during the first two practicals and develop an ambitious, open-ended application. It will require coming up with interesting ideas, researching independently, learning how to use new APIs and libraries, and implementing interesting functionality. This project will be done in groups of 5-6 people. Gaining experience in effective groupwork is one of the major goals of this practical.

Overview

You will create an online platform for cyclists. It's aim should be to provide everything a cyclist needs to plan, record and organise their daily rides, whether it's road races, mountain biking, BMX racing, ultra–distance cycle races, or simple daily commutes. At a minimum, your application should allow a user to keep a log of their training by recording information about their daily rides, such as distance and cycling time. It should also allow them to calculate useful statistics such as average speed, mileage per week, and anything else you may find useful.

The real challenge is to come up with useful functionality that goes beyond simply putting things into a database. Some examples include calculating route lengths given starting and end point (e.g. by interfacing with a mapping API), keeping track of upcoming events, a calendar interface representing the training plan (e.g. for an ultra–distance race), finding nearby cyclists with similar interests, plotting advanced statistics (personal best, current 25K time, average 25K time over the past year), plotting routes on a map, helping with a diet (e.g. calculating required calorie intake and suggesting nutrients), creating new routes on an interactive map (e.g. Google Maps or OpenStreetMap), weather forecasting, importing GPS traces in and plotting the route (e.g. using the GPX format), automatic grading of routes based on height profiles, recommending suitable routes that other users have cycled based on the statistics for both users, listing nearest towns so the users can plan arrival and departure. These are just suggestions, you are free to implement completely different ideas.

Requirements

Your application will have to fulfill each of these requirements:

- **User Interface** for interacting with the users (as in Practical 1).

- **RESTful API** written in Node.js/Express for exchanging data between the client and the server (as in Practical 2).
- **Database backend** written in Node.js/Express for persistent storage of data about users and possibly events.
- **An algorithm** for implementing additional functionality, such as calculating statistics, matching similar users based on experience, etc.
- **Use of an external API** such as mapping (e.g. Google Maps), distance calculation (e.g. MapZen), weather APIs, etc.
- **Use of a JavaScript library** such as D3 for graphics, chart.js for graphs, jplayer for media, etc.
- **Good Software Engineering practice** such as effective version control, testing, documentation, and modular design.

Deliverables

A single `.zip` file must be submitted electronically via MMS by the deadline. It should contain:

- The entire working copy of your repository, containing the source code and revision history
- A **joint** report (around 3000 words), in `PDF` format detailing the design of your solution, discussing any design decisions taken and how each of the requirements is met by your solution (including your development process, approach to testing, team work, etc.), and reflecting on the success of your application. Try to focus on the reasons for your decisions rather than just providing a description of what you did. **This will be joint work of the entire team.**
- An **individual** report (around 1000 words), in `PDF` format describing your own contribution and your own challenges.
- A short README file describing how to run your server and listing any node packages which need to be installed.

Submissions in any other formats may be rejected.

Marking Criteria

Marking will follow the guidelines given in the school student handbook:
`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptor`

Your submission will *not* be evaluated based on aesthetic appeal (this is not a visual design course), but use of CSS and DOM scripting which enhances the experience and interactivity will be rewarded.

There will be a group mark (based on the quality of the final application and joint report) and an individual mark (based on your individual contribution and individual report). The git repository will be used during marking to evaluate individual contribution.

Your final mark will be the average of the group mark and your individual mark.

Some examples of rough ranges for submissions in various bands are:

- A *basic implementation* **in the 11–13 grade band** should implement the user interface and a RESTful API and at least one further requirement. It should demonstrate good use of UI technology (e.g. HTML and CSS), Node.js and Express. It should allow a user to register and log in, and keep a basic log of distances and times. The site should be filled with enough initial data to make it possible to browse the webpage without a huge amount of typing. The code should be documented well enough to allow the marker to understand the logic.

- An implementation **in the 14–16 range** should demonstrate good programming practices such as good comments, good modular design, evidence of testing, and it should deal with errors gracefully. It should implement at least one of each requirements outlined above, e.g. use one external API, one additional library, and at least one algorithmic solution for a problem such as searching for other cyclists or generating statistics. HTTP error codes should be used by the RESTful API. All of the JSON objects passed through the API must be checked and validated.
- To achieve a mark of **17 and higher** you should use multiple external web APIs, multiple libraries, or implement more challenging functionality such as graphics, map integration, and route plotting. Additional extensions that can boost the mark include unit testing (e.g. using Mocha/Chai) or end-to-end API testing (e.g. using curl) and API documentation (e.g. using JSDoc). This is intentionally left open so you can explore your own ideas and pick functionality that matches the interests and backgrounds of your team.

Some specific descriptors for the **individual component** of the assignment are given below:

- A *poor contribution* **in the 0–7 grade band** indicates a weak or missing individual report, demonstrating confusion and misunderstanding of the topic. Little or no contribution to the final software, adding little to no value and focussing on only one part of the program functionality.
- A *reasonable contribution* **in the 8–10 grade band** indicates an individual report lacking detail or clarity, with a small contribution to the final software, which added little value and focused on only one part of the program functionality.
- A *competent contribution* **in the 11–13 grade band** indicates
  - a good individual report, but with a small contribution to the final software, focussing on only one part of the program functionality.
  - OR an individual report lacking detail or clarity, but with regular contributions to the project limited to a few parts of the project.
- A *good contribution* **in the 14–16 range** indicates a good report, regular contributions to the project, but limited to a few parts of the project.
- An *excellent contribution* **in the 17 and higher range** indicates an excellent individual report, regular and sizeable code contributions throughout the project, and contribution to many important parts of the project.

Word Limit

An advisory word limit of approximately 3000 words applies to the joint report for this assignment and an advisory word limit of approximately 1000 words applies to the individual report for this assignment. Word limits exclude references and appendices. No automatic penalties will be applied based on report length but your mark may still be affected if the report is short and lacking in detail or long and lacking focus or clarity of expression.

A word count must be provided at the start of each report.

Lateness Penalty

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):
`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties`

Good Academic Practice

The University policy on Good Academic Practice applies:

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`

Hints and suggestions

You do not have to follow any of these suggestions, but you might find some of them useful.

Have the basic implementation working before starting to work with external APIs and libraries. A clean, fully functional and well–written basic implementation is better than a buggy submission that tries to do many things poorly. Try to get the basic implementation working, commented and tested well before the deadline.

Start by spending some time on design – what needs to be done to meet the basic implementation? What type of data needs to be represented and where (client or server, database or memory)? What is the best way to represent the data – objects, classes, JSON objects, etc? Write a basic outline of your object/class structure and RESTful API on paper and think whether it makes sense. I recommend going to a nice comfortable cafe with your team and spending a couple of hours hashing this out and getting to know the other members of the team.

Divide the work into small chunks and decide how to split the work. Agree on a rough work plan, and meet regularly to see how the work is progressing sometimes you will need to adjust because things turn out to be more difficult than planned.

Try to avoid splitting the work into parts that are mostly separate (like only client–side and only server–side) and working on them independently. Integrating such work at the very end is likely to be difficult. Try to split the work into sub-groups, with two or three people working on a part of the problem. Then integrate work at regular intervals and swap the groups occasionally based on skills and experience so you can look at each others code from time to time. It will help you understand your partners thinking, help spot problems, and help you pick up good habits and tricks from each other.

Finally

Don't forget to enjoy yourselves and use the opportunity to experiment and learn! If you have any questions or problems please let me know (`mailto:ruth.letham@st-andrews.ac.uk`) — don't suffer in silence!