CAB230 Web Computing Assignment Server Side Specification

Release Date: May 2 2019
Submission Date: May 31 2019 11:59PM
The submission is due the Friday of Week 13
Weighting: 60% of Unit Assessment
Task: Individual Project

Introduction:

As noted in the earlier client-side specification, your task in this assignment is to develop a web application that makes crime statistics available to a user based on a range of queries. On the client side, you have developed an interface to query the API hosted at:

https://cab230.hackhouse.sh

Most students have also now made some attempt to present the resulting data in a form that is clear and readily understood by the user. This server-side work doesn't replace the earlier task, and you will need to strike a balance between the client and the server side components over the month of May. But the goal here is in principle very straightforward:

You must replicate the services that we have provided on the server side, and you must deploy them securely, connect your client-side app, and demonstrate that the whole application works and works well.

In the client-side spec we talked about the aims of the assignment, and these remain unchanged – our focus is on exposing and consuming REST APIs and full web application development. We will work with authentication and security, and we will guide you toward deployment.

As with the client side application, development of the server side component should be undertaken in stages. We prescribe some of the technologies – node.js and express are ***mandatory*** – but we are otherwise more flexible.

In what follows we will focus on the stages in development, and how they might correspond to the grade levels associated with the server component, which is worth 30% of the unit marks. We will not at any stage present an API specification in this document. The specification that you are required to implement is available in the Swagger docs at the application root linked above: https://cab230.hackhouse.sh. Other than to correct any errors which emerge, these will not change over the remainder of the semester, and this URL will remain live until after the assignment submission.

The Data:

As you know, the dataset is drawn from the Queensland Government open data initiative, showing criminal offences across the state over the past 20 years or so, organised according to offence type, local government area and type of offender. The URL for the original collection and some exploration of its contents was provided in the client side specification. These will *remain in the client side specification* and we will never touch them again. I want to emphasise that the database is provided for you, and you should find it very straightforward to upload the data into a MySQL database. Near to this specification in the assessment section on Blackboard, you will find a link to a MySQL dump of the database we have been using. You should use this to create your database and to populate its tables. This file will have the name:

`mysqldump.sql`

As discussed elsewhere, you may use another RDBMS if you wish, but the vast majority of the class will use MySQL. Above all else, do not go anywhere near the original dataset.

The REST API:

The REST API endpoints are as documented in the Swagger docs referenced above, and you are required to implement them as they appear there. We will not rehash them here in any detail, and in any conflicts between what is written in this document and the Swagger API docs, the Swagger docs have precedence.

Note that some of these routes (`/register and /login`) implement authentication, and that the `/search` endpoint requires that the user is authenticated before use. The information endpoints do not require authentication.

Staged Development:

The requirements for the server side of this assignment are ultimately pretty straightforward, but we strongly recommend that you follow the path laid out below.

Step 1: Create an Express App using express generator. You should initially serve these pages via HTTP, leaving consideration of HTTPS until later. You should expose the site and establish the routes needed for the application without spending too much time on the application functionality. You may organise these routes as you see fit, but we expect to see sensible application structure and use of routers to handle related tasks. Applications in which all of the routes are handled from app.js will not be viewed favourably by the markers.

Your work in step 1 should be based closely on the express prac exercises in which you develop a simple REST API to allow people to query the world cities database. You should generate a fresh application following the same approach, and use simple logging to ensure that the routes operate successfully. The database connection may initially be simple, but we will give more credit to approaches based on express middleware, as discussed in lecture 9 and in the second of the node and express prac sheets (released late in week 9).

Step 2: Having created an application and ensured that the routing is handled successfully, you should now replace these temporary logging statements with the requirements of each of the API endpoints. Most of the information endpoints correspond to simple SELECT * queries and these are a good point to begin. It is sensible to proceed as follows:

- Implement responses for the information endpoints – GET
- Implement responses for the authentication endpoints – POST
- Implement the basic /search endpoint without filtering or authentication
- Add filtering to the search endpoint

Step 3: Having created a working application deployed via HTTP, you should migrate the application to serve on HTTPS via localhost using a self-signed certificate. This will be discussed in the Security lecture and prac guide in week 10.

Step 4: Finally, in step 4, we will provide instructions on final deployment of your system prior to assessment. These arrangements are not finalised at the time of the spec release and so we will leave this aspect of the assignment to a submission instructions sheet later in the semester.

Security and Logging:

We will require that you undertake an appropriate review of your application to limit its vulnerability – please see the lecture in week 10 for a list of the sorts of aspects to consider, but the most important in this application relate to the avoidance of SQL injection attacks.

At a code level, we require that you use the helmet middleware with the default components selected, but with the addition of the Content Security Policy component. These options may be found here at https://helmetjs.github.io/docs/, with additional discussion to take place in the week 10 lecture.

You should also use a standard logger component to preserve the interactions with your server. The standard approach would be to use the facilities of Morgan, which may be found here: https://github.com/expressjs/morgan

As noted above, migration to HTTPS will be undertaken later in the semester, and step by step instructions will be provided.

Other General Considerations:

In general, we will allow a number of alternative approaches to building the API server. As above, we mandate the use of node.js and Express, and you are strongly advised to base your approach on the pracs as discussed. However, there remain a number of other aspects to consider in building the application and in assessing its merit. These aspects include the nature of the database connectivity, the application architecture and role of application and router level middleware, and the use of appropriate security measures. In general, we will

allow a simplistic approach to some of these requirements, but we will give more credit when the approach taken reflects current standards in professional practice.

For example, the first version of the prac exercise introduced a very simple model of database connectivity based directly on the mysql package from the NPM registry. The model adopted was flawed from a professional standpoint in its failure to handle the connection properly, and it is a weaker approach than using the application middleware as you do in the second node and express worksheet. The second alternative provides a significantly cleaner architectural approach, and there is existing, well-established middleware available (see https://knexjs.org/) to do the job and to do it well.

Grade Standards:

Broadly speaking, grade standards for the server side of the assignment will correspond to the feature levels laid out below, and there is a reasonable alignment with the development steps outlined above. Mostly the lower grades correspond to the easy endpoints. The higher grades require that you successfully implement search and filtering, and that you follow a professional approach in the architecture and construction of the server. In particular, you should demonstrate that:

- The routes and the overall architecture are professional – logical and uncluttered with appropriate use of specific routers mounted on the application
- There is appropriate use of middleware for managing components such as database connectivity and security
- There is appropriate error handling and responses match those in the API spec
- The application is successfully deployed using HTTPS (instructions to follow later)
- There is an appropriate attention to application security (see below).
- The application successfully serves accurate Swagger docs on the home page route

All of these requirements must be met in order to achieve the highest grade levels.

Please note that we reserve the right during marking to connect a front end created by us to your API. So please ensure that your client and server components communicate only through the HTTP/S GET and POST methods described in the API docs. I recommend that you keep developing your front end against the API at https://cab230.hackhouse.sh until the very last stages of the assignment.

As before, you should understand that the grade levels discussed below assume that the feature levels required have been implemented competently. As always, if the implementation is substandard, then the marks will go down, sometimes substantially. It is more than possible to get a 4 level mark by doing a bad job of a 5 level requirement. Similarly, if you do a bad job of this 4 level requirement, then you are likely to get a 3 level mark. The

precise mark necessarily depends on the mix of features successfully implemented, and the grade levels below are intended as a guide.

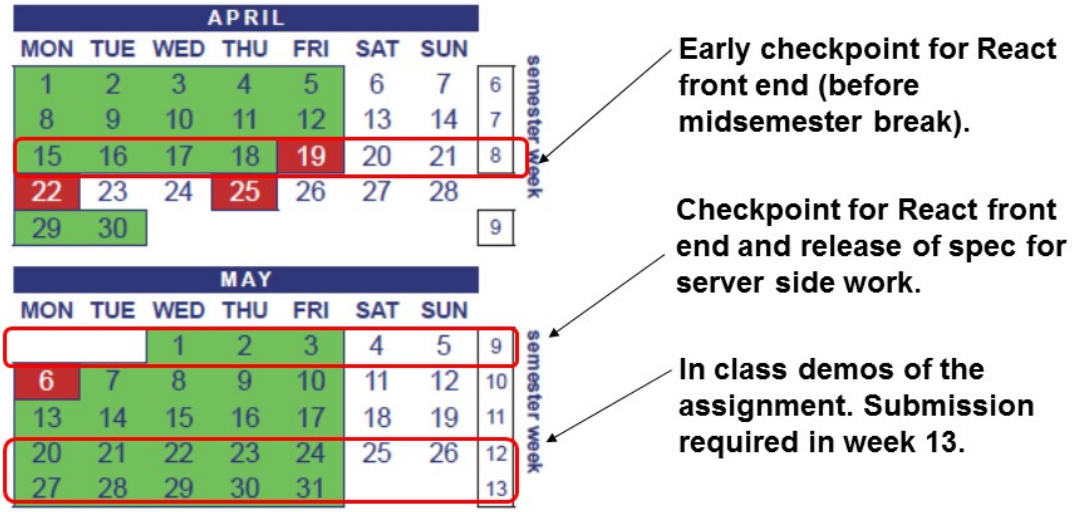
After taking note of all of these warnings and exceptions, the passing levels are as follows:

4. This level requires that you successfully deploy an Express based API which supports most of the endpoints of the API and interacts successfully with the database. The missing routes will vary between students, but our assumption is that people will successfully complete the information and basic search facilities first, and then have more difficulty with the authentication routes (registration and login) and their use in the authentication requirement for the search facilities. As discussed above, we expect that you will build the search facilities and *then* manage the authentication requirements.
5. The next level includes successful implementation of the endpoints discussed at the grade of 4 level, and successful implementation of authentication routes and authenticated search. At this level we would expect to see basic filtering on the search route, but the multiple term filtering may not be in place. We would also expect to see Swagger docs on the API root.
6. The 6 and 7 level grades require successful completion of the authenticated search route with full filtering capability as described in the API docs. The distinction between 6 and 7 level grades then relates to any errors in this basic functionality, and in the successful use of middleware security, database connectivity and deployment of the Swagger docs.
7. Please see the discussion at 6 above.

At the lower grade levels we expect basic attention to detail in security, and successful use of the database. At the higher grade levels – 6 and 7 standard work - we expect to see the database connection handled via middleware, appropriate use of the helmet security middleware (see above) and full implementation of the authenticated search endpoint as discussed earlier. There is no 'killer' requirement here that makes the difference between a 6 and a 7. These requirements are best seen as a set that together, and done very well, give you a 7 standard mark. If you miss some of them, but still do a good job of the search endpoint, then you are likely to get a 6 standard mark.

Submission

We will provide clear instructions on deployment and submission closer to the end of the semester. In the image below from week 1, you will see that we require you to demo your work at the end of the semester. This will once again take place in the prac classes and in special sessions that we will arrange to handle the overflow. We will provide further information on these requirements in due course. One important feature is that these demos will require that you show us your front end operating with queries to your back end system.



There is one final requirement on submission. As with the client side component, we will expect a short report and user guide, generally running to 5 pages or so, including screenshots. Your report must include the following sections:

1. Introduction – telling us what was implemented and what wasn't.
2. Technical description of the application. This section is to allow you to talk about the application architecture and middleware choices, and to discuss technical issues that caused you problems. This is especially important if something doesn't actually work.
3. Security – a brief discussion of security features of your application.
4. Testing and limitations – test plan, results, compromises.
5. References
6. Appendix: a brief installation guide.

The CRA rubric will be released as a separate file.