ISYS2120 Assignment 3 (sem1, 2023)

Due: Sunday 22 October, 11:59pm Sydney time
Value: 10% [5% for the group as a whole, and 5% individual component]
This assignment is done in **groups of up to 5 students** (aim for exactly 4 members, but it may happen that sometimes a group is smaller or larger, eg if there are not enough students in a lab). We ask that all students in a group be attending the same lab session, so you can work together more easily, and show progress to the lab tutor in scheduled meeting.
Late work policy: (from unit outline) Late submissions for assignments will incur a penalty of 5% of the maximum awardable marks for each day, or part-day, past the due date, up to a maximum of 7 days (as after this time, feedback on on-time submissions will be available, resulting in an unfair advantage if submissions after this time were accepted). After 7 days late submissions will not be accepted. Where special consideration is granted for these assessments, extensions of a maximum of 7 days will be permitted. After 7 days, reweighting of other relevant tasks will be applied.

Summary (but details further down are authoritative)
Groups: usually with 4 people from a single lab session, who all get the same mark from the group submission (except that if some member is considered to have not contributed reasonably, then the unit coordinator may reduce their score appropriately.) In many cases, the group has the same members as the Asst2 group; however this is not required. Groups are initially set up with membership the same as in asst2; changes can be made during week 9, by contacting the lab tutor (in class) or unit coordinator (be email).
Provided for you: code skeleton for a web application (already provided in week 8 lab)
Each member produces: a revision of the code skeleton, with extra code (and perhaps some changes in what was provided), to offer six extra functionalities [listed in detail below] in connection with one table of the data. Also, the member produces a report describing the changes made to the code, and discussing strengths and limitations.
The group produces: a report evaluating the security of the various web applications, and describing how the members' code was tested.
Submit (as individual): a report with structure as described in detail below [submit on Canvas], and an archive with the code of your extended system [submit on Canvas]
Submit (as a group): a report with structure as described in detail below [submit on Canvas]

Group Membership Adjustment Procedure: In the second hour of week 9 lab, you should form a group. It is expected that most people will stay with the members of the group they were with in Asst2, but it is not necessary. There will be separate Canvas groups for this assessment; we will initialize them as copies of what were formed for asst2, but membership can change as described below. If any student wishes to not be with their asst2 team (or if they want extra members to increase the group size), they should speak to the lab tutor at the start of week 9 lab; alternatively, they can email the unit coordinator to make the request [but you should do so by Friday October 6 at the very latest]

If, during the course of the assignment work, there is a dispute among group members that you can't resolve, or that will impact your group's capacity to complete the task well, you need to inform the unit coordinator, alan.fekete@sydney.edu.au. Make sure that your email names the group, and is explicit about the difficulty; also make sure this email is copied to all the members of the group. We need to know about problems in time to help fix them, so set early deadlines for group members, and deal with non-performance promptly (don't wait till a few days before the work is due, to complain that someone is not doing their share). If necessary, the coordinator will split a group, and leave anyone who doesn't participate effectively in a group by themself.

The task:
The starting point for this assessment is the data-backed web-app that was used in the week 8 lab, accessing a (partially genuine, partially artificial) dataset about public transport. Every member should be familiar with the structure of this code, and especially with the relational schema that defines the data (found in the file 01_schema.sql).

As a group, during week 9, you should decide which one of the following tables each member will deal with: the choices are Trips, Stations, OpalCards, TravelTimes, and CardTypes (note that you may not work with Users table, for which the code is found in the scaffold given to you). Each member deals with one table only, so if there are fewer than 5 members in the group, one or more of these tables will not be handled.

Also, as a group, during week 9 lab, you need to choose one member to be the "data owner"; this member will be the one whose postgresql database each other member accesses, and therefore the data owner will need to grant appropriate permissions on their schema and certain of its tables, to other members. Note that the other individual members will need to alter their config file to access the data owners database rather than their own (however, it will need to be with their own login and password – the data owner must not reveal their password to other members). It is expected that each group member will tell the data owner what table access they need, and then the data owner can grant it. Similarly, if any member later wants to alter the data schema (for an extension), they will tell the data owner what DDL commands to perform, but the data owner will execute those commands on the shared dataset, through pgadmin.

Each group member, ***as an individual***, is asked to extend the provided web-app code with six extra functionalities as described in detail in one of the bullets below (this will involve some extra webpages in the website, supported by extra functions written into various files of the code base). Each members work is to allow users to show all rows of the corresponding table, show some rows whose content matches values provided by the end-users, add a row, delete a row, update a row, and produce a report of some kind that aggregates information from the table. The provided codebase has the same kinds of functionality for the Users table, so students should be able to adapt these parts of the code for their own needs. *Note that each member is expected to write the code to create these* *pages, in their own copy of the web-app (using their own login etc for the postgresql access) but all should access the same dataset, which is that of the data owner.*

- A member who is allocated the Trips table, should provide functionality so the end-user can (if they have the appropriate authority): show all the Trips known to the database, show the trips that occur on a particular date (which is entered by the end-user through the web page), add a new trip to the database (with values for the relevant fields being entered by the end-user through the web page), update other fields for a particular trip, remove a trip that is in the database, produce a report showing the various entrystations (by stationid) alongside with the number of trips that start at each of those stations.

- A member who is allocated the Stations table, should provide functionality so the end-user can (if they have the appropriate authority): show all the Stations known to the database, show the stations that occur within a particular range of latitude and longitude values (the lower and upper limits that define the range are to be entered by the end-user through the web page), add a new station to the database (with values for the relevant fields being entered by the end-user through the web page), update other fields for a particular station, remove a station that is in the database, produce a report showing the various stationtypes (by stationtypeid) alongside with the number of stations that belong to each of those stationtypes.

- A member who is allocated the OpalCards table, should provide functionality so the end-user can (if they have the appropriate authority): show all the cards known to the database, show the cards that expire on a particular date (which is entered by the end-user through the web page), , add a new card to the database (with values for the relevant fields being entered by the end-user through the web page), update other fields for a particular card, remove a card that is in the database, produce a report showing the various expiry dates alongside with the number of cards that expire on each of those dates.

- A member who is allocated the TravelTimes table, should provide functionality so the end-user can (if they have the appropriate authority): show all the path segments (which is what each row represents) known to the database, show the path segments that traverse a particular number of stops (this number is entered by the end-user through the web page), add a new path segment to the database (with values for the relevant fields being entered by the end-user through the web page), update other fields for a particular path segment, remove a path segment that is in the database, produce a report showing the various stations alongside with the longest expected traveltime among all the segments that start at each of those stations.

- A member who is allocated the CardTypes table, should provide functionality so the end-user can (if they have the appropriate authority): show all the card types known to the database, show the cardtypes whose name includes a given string (this string is entered by the end-user through the web page), add a new card type to the database (with values for the relevant fields being entered by the end-user through the web page), update other fields for a particular card type, remove a card type that is in the database, produce a report showing the various fare modifiers alongside with the number of card types to which that fare modifier applies.

Coding Extensions: Doing the coding described above can gain up to 3 points out of 5 for individual work; there is also 0.5 individual point available for discussing strengths and limitations of your coding work . Any member who wants to earn a higher level of marks for this assignment, can also write some more code; they may choose to simply extend the schema of data with some additional tables (that must be relevant to the domain), and then provides functionality involving the extra information; instead (or as well) they can choose to learn (and then utilize) some further aspects of the Flask environment that allows better user experience. For example, someone might choose to allow end-users to provide input values (such as an end station) through a drop-down that only accepts the valid values, or they may control the display so that if there are many rows of output, only the first few are shown initially, with more available on end-user request.

As a group, you must arrange to test everyone's code extensively, and also evaluate the security issues of everyone's application.

In testing, we expect each member's code will be tested by at least one other member in the group, who interacts with the web application by opening pages in a browser. For full points, the testing will need to cover edge-cases by trying a variety of situations with different data in the database (arranging the data may be done either by using the web app to make the changes, or by altering the database contents directly through pgadmin4, but the latter will likely require the tester to get extra permissions from the data owner)

The group is expected to discuss and jointly write about the security aspects of the system (for the situation where real data about cards, users, etc are in the data, rather than the partially artificial data we supplied). The discussion should indicate what security goals would be appropriate and what attacks need to be considered. Then, for each member's extended version of the web application, describe the mechanisms by which it tries to achieve security, and evaluate the effectiveness of those mechanisms (for example, it could indicate any realistic attacks that could succeed against the system to violate one or more security goal). The discussion should consider security of the whole system, covering the database as well as the web application, and the operating systems on which these are run.

What to submit (and how):
Each member *individually* must produce a file that is a code archive of the whole web application, based on the provided scaffold and incorporating their individual changes. Also, the member must produce a PDF report, which contains the following four parts
Part A
Give the studentid of the member, and the name of the group to which they belong.

Part B
Show the code additions or changed parts, where you have altered what was provided in week 8 lab scaffold code. For example, you can give the Python code of any methods you have written, and the HTML of any template pages. [If you have not been able to write working Python, you can at least give here SQL queries that would produce the expected effect when run directly on the database (and this will gain partial points, if the SQL is correct).

Part C
Describe any extensions you have made, beyond the required functionality for your allocated table. In particular, describe any extra tables you had added to the schema, and how (and why) you used these; describe any extra aspects of Flask that you learned and used (and state where each was used). If you did not do any extensions, Part C can simply say that.

Part D
Write a discussion of strengths and limitations of your system, in regard to the functionality provided and its convenience for users.

The member must upload the individual report and the code archive, to the appropriate Canvas assessments (called Asst3 Individual Report, and Asst3 Individual Code)

The *group* must produce a PDF report with the following structure, in five parts.
Part A:
A list of SIDs of the group members who participated in the work, and for each, a description of what contribution they made (including which table they were allocated for individual code, whether they were the data owner, what testing they did, etc). Note: ***do not*** include student names or unikeys; only refer to members by SID, to allow marking to be anonymous.

Part B:
A description of the testing that was done (including for each test, what the purpose of the test was, and what result was obtained). If any tests involved changing the database contents, the report should state explicitly the content used for that test.

Part C:
A discussion of the security goals appropriate for the application, and the attacks being considered

Part D:
For each member's system, a description of the security mechanisms that were used. [Any mechanisms that are shared between members, for example, in the database itself, can be described once, and then simply referenced when describing each member's system separately.]

Part E:
For each member's system, an evaluation of the success or limitations of the security mechanisms that were used. [Any discussion of the success or inadequacy of the mechanisms that are shared between members, for example, in the database itself, can be stated once, and then simply referenced when evaluating each member's system separately.]

One member of the group should submit the PDF of the report, to the Canvas link for Asst3 Group Report.

Weekly progress steps
Week 10: In the meeting with the tutor, each student should show the SQL commands they will use, at the heart of their code, for each of the required functionalities. The group can also use some time to describe what they have considered about security goals and possible attacks to consider, for the application. The tutor can provide feedback and suggestions.

Week 11: In the meeting with the tutor, each student should show the code which they have written in database.py, for each of the required functionalities. The tutor can provide feedback and suggestions.

Marking scheme:
For the individual component (out of 5):
The score is based mainly on the submitted individual report (which includes a copy of code changes to the provided scaffold, written by the individual); the code archive may be consulted if the marker wishes to check any aspect.

For each of the 6 required functionalities (as shown in Part B): 0.5 point [0.25 can be obtained as long as correct SQL code is written, even if the web application does not run correct]
For extensions (as shown in Part C): 1.5 points [up to 0.75 points for adding extra relevant tables to the schema, and using them in valuable ways; up to 0.75 points for introducing additional features of Flask, beyond what is shown in the provided scaffold, and using these in valuable ways]
For discussion of strengths and limitations of the system (as shown in Part D): 0.5 point

For the group component (out of 5):
The score is based on information in the submitted group report; the members' individual code archives may be consulted if the marker wishes to check any aspect.
Part A is not marked. However, if this Part shows that some student did not make a reasonable contribution to their group's submission, then the unit coordinator may reduce that member's mark from what the group was awarded.

For testing (based on Part B): 2 points [up to 1 point can be gained from testing the SQL queries directly, rather than from running the web application; up to 1.5 points can be gained from tests that use provided data]

For discussion of security: 3 points
0.5 points for description of security goals, and attacks being considered (based on Part C)
1.0 points for description of security mechanisms (based on Part D)
1.5 points for evaluation of effectiveness of security (based on Part E)