ISYS2120 Assignment 3 (sem1, 2024)

Due: Sunday 20 October, 11:59pm Sydney time
Value: 10% [5% for the group as a whole, and 5% for the individual component]
This assignment is done in **groups of up to 5 students**. 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 or 5 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.) The groups were formed in week 4, and typically combine two subgroups who worked separately on Asst2. You can ask to change group membership, by speaking to tutor in week 9 lab, or emailing coordinator before the end of week 9.
Provided for you: code skeleton for a web application (already provided in week 8 lab, though you should get a corrected version from Canvas)
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 which has a copy of all the code changes, and also describes the changes made to the code.
The group produces: a report evaluating the correctness and security of the various web applications written by the members, and describing how the members' code was evaluated.
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]

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 air travel. 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 Airports, Aircraft, Flights, Passengers, Tickets. (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 member's work is to allow users to show all rows of the corresponding table, show some row given the primary key value, add a row, update a row, remove 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 (but please, first make sure to fix any security errors in the provided code, which you were shown in lab08). *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 to the Airports table, should provide functionality so the end-user can (if they have the appropriate authority for that activity): show all the airports known to the database, show the airport with a given AirportID (which is entered by the end-user through the web page), add a new airport 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 airport, remove an airport that is in the database, display a summary showing the various countries, alongside with the number of airports that are located in that country.

- A member who is allocated the Aircraft table, should provide functionality so the end-user can (if they have the appropriate authority for that activity): show all the aircraft known to the database, show the aircraft with a given AircraftID (which is entered by the end-user through the web page), add a new aircraft 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 aircraft, remove an aircraft that is in the database, display a summary showing the various manufacturers, alongside with the number of aircraft that were manufactured by that manufacturer.

- A member who is allocated the Flights table, should provide functionality so the end-user can (if they have the appropriate authority for that activity): show all the flights known to the database, show the flight with a given FlightID (which is entered by the end-user through the web page), add a new flight 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 flight, remove a flight that is in the database, display a summary showing the various airports (by airportID), alongside with the number of flights that departed from that airport.

- A member who is allocated the Passengers table, should provide functionality so the end-user can (if they have the appropriate authority for that activity): show all the passengers known to the database, show the passengers with a given PassengerID (which is entered by the end-user through the web page), add a new passenger 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 passenger, remove a passenger that is in the database, display a summary showing the various nationalities, alongside with the number of passengers that are of that nationality.
- A member who is allocated the Tickets table, should provide functionality so the end-user can (if they have the appropriate authority for that activity): show all the tickets known to the database, show the ticket with a given TicketID (which is entered by the end-user through the web page), add a new ticket 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 ticket, remove a ticket that is in the database, display a summary showing the various classes, alongside with the number of tickets that are of that class.

Extra Coding Extensions: Doing the coding described above, for the six functionalities on your allocated table, can gain up to 3 points out of 5 for individual work. Any member who wants to earn a higher level of marks for this assignment, can also write some more code that improves the webapp. [However,

do not attempt any extra coding until you have got working the six functionalities for the table you were allocated!]
One direction of extra coding is for the individual 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.

If one doesn't choose to learn new features of Flask, you can instead provide more functionality beyond the six we described, through adapting the techniques already present in the provided code base. If you only do this, the most you can get for extensions is 1.25 out of 2. Examples of this type of extra functionality could be to provide other useful functionalities for the table they were allocated; or they may choose to extend the schema of data with some additional tables (that must be relevant to the domain), and then provide functionality involving the extra information;

As a group, you must arrange to evaluate everyone's code for both correctness and security, and write a report about this.

In evaluating for correctness, we expect each member's code will be tested by at least one other member in the group, and the code will be reviewed by at least one other member (that is, someone testing or reviewing their own code is not good enough). For full points, the testing will need to be done by interacting with the web application by opening pages in a browser, and must 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).

Evaluating for security should make the assumption that real data about cards, users, etc are in the flights, passengers etc is stored,, rather than the artificial data we supplied). This evaluation would typically involve both attempts to violate security, and code reviews, checks of configuration settings etc. The evaluation 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):

Individual submission
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 a cover page in the format we provide, followed by the following two parts

Part A
Show the code additions or changed parts, where you have altered what was provided in week 8 lab scaffold code. [This should be taken exactly from your code archive] For example, you can give the Python code of any methods you have written, and the HTML of any template pages. If you changed the code in a method that was already in the scaffold, make clear what has changed (eg by boldface for what you wrote). [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). If you wrote extensions beyond the required functionalities, include the code changes for those as well.

Part B
Describe any extensions you have made, beyond the required functionality for your allocated table. In particular, list any extra functionality you provided for the table you were allocated; 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 B can simply say that.

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

Group submission:
The *group* must produce a PDF report which contains a cover page in the format we provide, followed by the following three parts.

Part A:
This is divided into subparts, one for each group member (and the group members id should be clearly stated at the start of the subpart). That subpart should describe HOW the member's application was evaluated for correctness (that is, what tests were run, what code reviews were done, etc), and WHAT conclusions were made about correctness. If any tests involved changing the database contents, the report should state explicitly the content used for that test.

Part B:
A discussion of the security goals appropriate for the application, and the threats being considered. That is, you should identify what information should be available to whom, and what changes can be made by whom. It's important to consider parties in a variety of roles, not only end-users with accounts in the webapp, but also various admins on the systems involved, and those without accounts too.

Part C:
This is divided into subparts, one for each group member, and one extra subpart that considers issues beyond the webapp code (eg in the dbms, in the systems on

which the components are running, etc). The subpart a member should give the members student id, and then describe HOW the member's application was evaluated for security (that is, what attacks were attempted, what code reviews were done, etc), and WHAT conclusions were made about security (this includes statements about attacks that would not succeed, as well as those that would breach security). The extra subpart should describe how the security of other aspects was evaluated and what conclusions were made about security. If any security weakness is observed, please be explicit about how this could be attacked, and what damage could be done.

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 the code in Part A): 0.5 point [0.25 can be obtained as long as correct SQL query is written, even if this is not integrated correctly into the Python scaffold]
For extensions (as shown in Part B): 2 points [up to 1.25 points can be obtained with using code that is modelled on the scaffold; for higher scores you need to introduce additional features of Flask, beyond what is shown in the provided scaffold, and using these in *valuable* ways]

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.

For evaluating correctness (based on Part A): 2 points [max of 1 point can be gained from testing and reviewing the SQL queries directly, without running the web application and reviewing its code; max of 1.5 points can be gained without doing tests on tester-designed data]

For discussion of security goals and threats (based on Part B): 1 point
[up to 0.75 points can be gained from considering direct security attacks involving using the running webapp; higher scores require also considering threats to the whole data and IT ecosystem]

For evaluating the security (based on Part C): 2 points [max of 1.5 point can be gained from checks that only look at uses of the running webapp; higher scores require also sensibly evaluating threats to the whole data and IT ecosystem]