CSE 3101: Internet Computing II

Lab Final Project

Paul Crawford (based on Mr. Girendra Persaud's proposal)

Semester 1 (August–December 2018)

1. Aims

1.1. Further understanding of Web applications, including client-side Web Forms (with server-side processing upon submission), server-side PHP functions, and service-oriented architectures.

1.2. Facility with the comprehensive set of concepts & techniques needed to develop a **prototype website** for the Department of Computer Science (CS), Faculty of Natural Sciences, University of Guyana (UG).

1.3. This completed Final Project's deadline has been extended but now revised again, and it is now due on **Saturday, 1st December**, 2018**, no later than 6 PM**. **This is a strict deadline, since all pre-Exams assessments must be concluded before the start of Exams Week.**

2. Specifications

[Note: The website's content will ***not*** be real data, but rather **simulated** data.]

2.1. Core objectives for the CS Department's prototype website:

2.1.1. Promote the work of the Department of Computer Science.
2.1.2. Provide information about the programmes and courses offered by the Department.
2.1.3. Provide a space for Alumni to be showcased.
2.1.4. Provide information about research undertaken by students and staff.
2.1.5. Provide profile information of staff.
2.1.6. Provide information on CS clubs.

2.2. Functionality (Both Maintenance and Display):

2.2.1. Staff Profiles.
2.2.2. Undergraduate Research Projects.
2.2.3. Programmes and Courses.
2.2.4. Department Activities.
2.2.5. Alumni Profile.
2.2.6. Information about CS Clubs.

2.2.7. Staff Research — Lists of papers and presentations, etc., that staff members are working on; could each be a separate page and/or part of profile page for staff.

2.3. Intended Audience:

2.3.1. Prospective and current students
2.3.2. Alumni
2.3.3. Researchers worldwide
2.3.4. Funding Agencies/Supporters/Donors

Website Maintenance & Display: Workflow

[NOTE: In the Figures below, dotted lines ('- -') indicate **navigational / redirectional flow**, whereas solid lines ('—') denote **data flow**.]

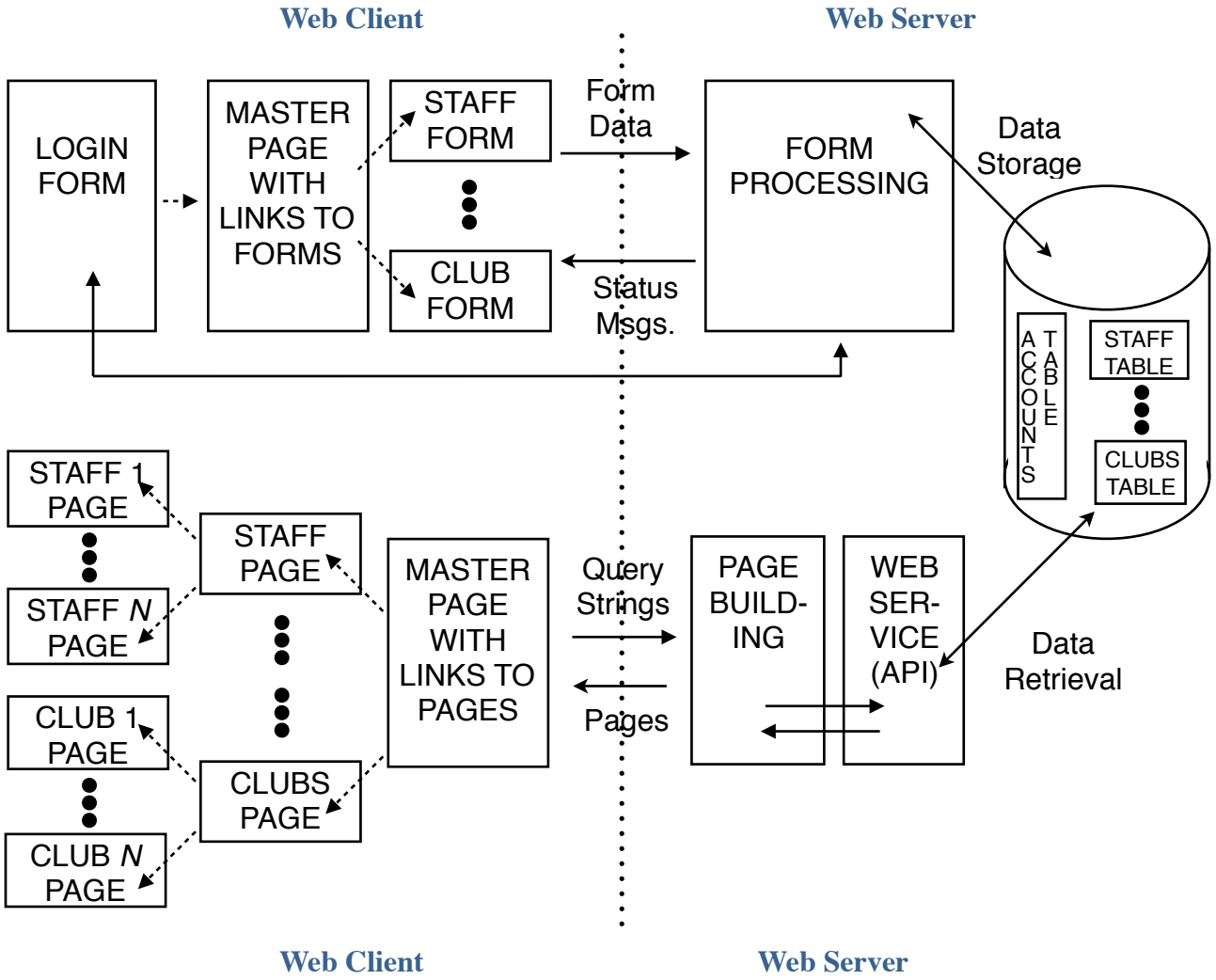

Figure 1

A). Website Maintenance Sub-System: More Details

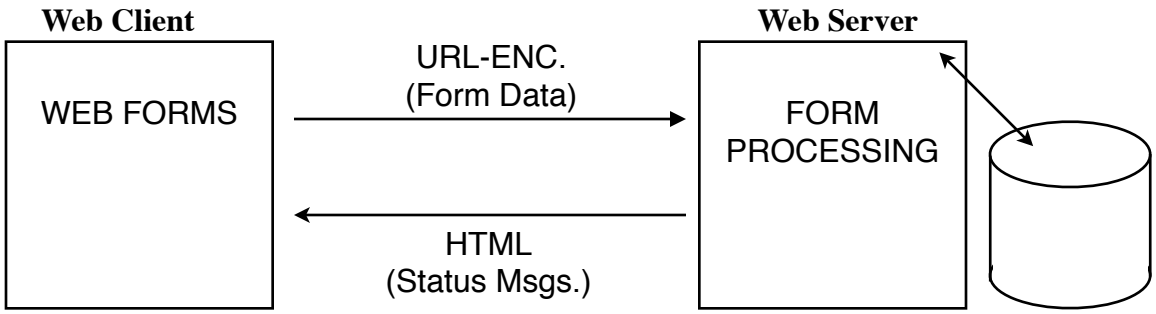

Figure 2

B). Website Display Sub-System: More Details

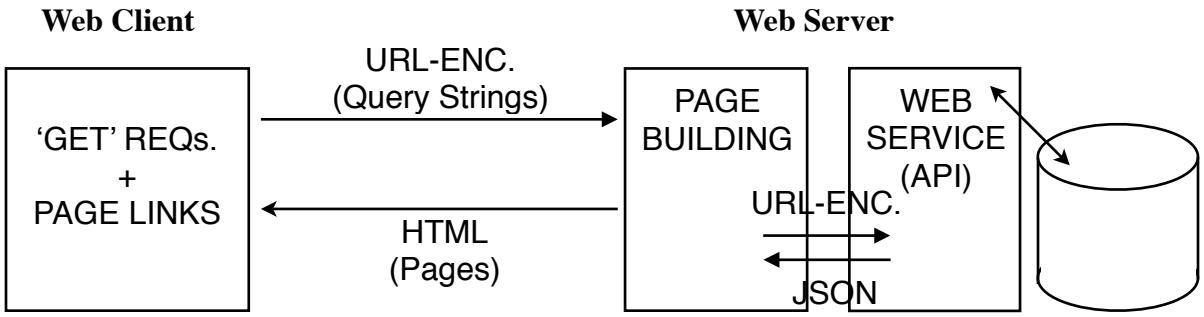

Figure 3

3. Tasks

3.1. Website Maintenance Sub-System

[Note: The earlier 'Address Book' web app Assigment is readily adaptable to this sub-system.]

3.1.1. Web Forms {Client-Side (received from, & submitted to, Server-Side)}

3.1.1.1. Display a Login Form for an Admin user.

3.1.1.2. Display a master Data Submission page with links to the following Web Forms:

- Staff Profile form
- Undergraduate Research Project form
- Programme and Courses form
- Department Activity form

• Alumn(us/a) Profile form
• CS Club Info form
• Staff Research form

3.1.1.3. For each of the above data-category forms, allow submitting the appropriate "dummy" data via a well-designed layout, with as much detail as suitable (at your discretion).

3.1.2. Page Processing / Handlers {Server-Side}

3.1.2.1. Process the submitted Login form, and upon successful login, send back a Master Data Submission page with links to the above-mentioned Web Forms.

3.1.2.2. For each of the submitted data-category forms, process the submitted "dummy" data as follows:
• Store the submitted data into the appropriate table(s) in the database.
• For any uploaded files such as PDFs, store only **pathnames** in the database, and place the actual PDFs into the server's file system.
• Send back a status message to reflect processing sucesss or failure (with meaningful human-readable errors).

3.2. Website Display Sub-System

3.2.1. Web Pages & Links {Client-Side (received from, & submitted to, Server-Side)}

3.2.1.1. Display a CS Department main web page, with suitably formatted links to each of the available data categories (stored by the Web Maintenance Sub-System).

3.2.2. Page Processing / Handlers {Server-Side}

[Note: This sub-system's Server side consists of **two logical components**: (1) a Web Page Builder; and (2) a Web Service/API. The former "calls" the latter to retrieve information from the database in JSON format, and then transforms that JSON data into formatted HTML pages.]

3.2.2.1. Process the request for the CS Department main web page by sending back a master Data Display page, nicely laid-out, with links to all the available data categories from above.

3.2.2.2. Process each data category's 'GET' request, as follows:

• Retrieve the appropriate information from the database, via a simple Web Service/API (RESTful flavour). I.e., use any suitable mechanism — such as the built-in I/O function '`file_get_contents()`' which also works with URLs — to submit a '**R**ead' ('GET') request to the Web Service's appropriate 'GET'-style URI (for Staff, or Clubs, etc.), which will perform the actual database query and then return the results as a JSON array (text).

Note that it would also be best to modify the API slightly so that it would allow suppressing the setting of HTTP **error** response codes, and merely return an error-message JSON array (text) as usual; this way, instead of getting interrupted by a web browser's typically cryptic error-handler page, the client would be free to decide for itself how best to display any errors.

• Decode the retrieved JSON array string, and then "translate" that into a nicely laid-out HTML page containing the formatted information along with appropriate links (e.g., back up to the Master Categories List page, or on towards an Item Details page, etc.).

Note that the same processing page can handle requests both for a given Category List page (e.g., a Staff List page, or Club List page, etc., with abbreviated items, each containing a link pointing to more detailed information) and for each Item Details page (e.g,., a particular Staff Member Details page, or Club Details page, etc.). For the former, ther would be **no** query-string appended to the URI (e.g., just '…/staff.php', or '…/club.php', etc.); whereas, for the latter, there would indeed be a query-string — specifying the particular record's **ID** — appended to the URI (e.g., '…/staff.php?**id=2**', or '…/club.php?**id=5**', etc.).

• Send back that HTML page, or an appropriate error message.

3.3. Requirements / Deliverables {Modelled upon the earlier 'Address Book' web app assignment}

3.3.1. Documentation of **complete functionality as described in § 2 'Specifications'**:- A PDF file with the following contents:

3.3.1.1. Use Case diagrams.

3.3.1.2. Entity Relationship (ER) diagrams.

3.3.1.3. Class diagrams (including the functionalities for each class).

3.3.1.4. Database schemas in the 3rd Normal Form.

3.3.1.5. Login credentials for (the Site Maintenance Subsystem's admin access to) the database.

3.3.2. Code covering **complete functionality as described in § 2 'Specifications'** (emailed to <pcrawford@mac.com> by midnight of the deadline given above in § 1.3)

3.3.2.1. An archive (Zip or RAR) entitled 'CSE3101_FinalLabProject_<*YourGroupName*>.zip' (or '.rar'}, containing the project code folder and your exported database '.sql' file.

3.3.2.2. The code must have comments describing the functionalities found within each file. ~~You must also include comments to indicate the names of the persons who worked on each of the different sections of the code.~~

3.3.2.3. You may also include a '.txt' README file containing any other information that you feel is necessary for the running of your web application.

3.4. ~~Presentations {Schedule Permitting} {Modelled upon the earlier 'Address Book' web app assignment}~~

3.4.1. ~~Maximum allotted time: 10 minutes per group.~~

3.4.2. ~~As a group, you will demonstrate that your web application successfully performs the functionalities as described in the requirements. You may also talk about your experiences during the process, and any issues that were overcome.~~

3.4.3. NOTE: Due to a lack of available remaining class time-slots, the **presentations have been cancelled**.

3.5. Overall Considerations

3.5.1. The use of Web Frameworks will **not** be allowed, other than UI layout ones such as Bootstrap.

3.5.2. Please remember to strive for high-quality code with clean design, decomposition into appropriate classes & functions, descriptive comments, etc.

3.5.3. For additional practice and code samples, you could also explore any of various useful tutorials & references available online. Please refer to the Lab Sheets for details.

Appendix: Database Schema {Informal}

Database

Name: uog_cs_prototype_db

Tables

For Maintenance Login ONLY:

1. accounts
   - 1.1. id — int(11) {primary key; auto-increment from 1 by 1}
   - 1.2. user_name — varchar(255)
   - 1.3. password_hashed — varchar(255)
   - 1.4. created — datetime
   - 1.5. modified — timestamp {default: CURRENT_TIMESTAMP}

For Internal Housekeeping ONLY:

2. categories
   - 2.1. id — int(11) {primary key; auto-increment from 1 by 1}
   - 2.2. name — varchar(255)
   - 2.3. description — text
   - 2.4. created — datetime
   - 2.5. modified — timestamp {default: CURRENT_TIMESTAMP}

Pre-Filled Entries:

id	name	description
1	Staff	Data for Staff Members.
2	Undergrad Research	Data for Clubs.
3	Alumni	Data for Alumni.
4	Courses	Data for Courses.
5	Programmes	Data for Programmes.
6	Undergrad Research	Data for Undergraduate Research.
7	Activities	Data for Departmental Activities.

User-Accessible Categories:

3. activities
   - 3.1. id — int(11) {primary key; auto-increment from 1 by 1}
   - 3.2. name — varchar(60)
   - 3.3. description — text
   - 3.4. category_id — int(11) {default: 7 (*categories* ▸ "*Activities*")}
   - 3.5. created — datetime

3.6. modified timestamp {default: CURRENT_TIMESTAMP}

4. alumni
   - 4.1. id — int(11) {primary key; auto-increment from 1 by 1}
   - 4.2. first_name — varchar(30)
   - 4.3. last_name — varchar(30)
   - 4.4. age — int(11)
   - 4.5. photo_pathname — varchar(230)
   - 4.6. research_pathname — varchar(230)
   - 4.7. category_id — int(11) {default: 3 (*categories* ▸ "*Alumni*")}
   - 4.8. created — datetime
   - 4.9. modified — timestamp {default: CURRENT_TIMESTAMP}

5. clubs
   - 5.1. id — int(11) {primary key; auto-increment from 1 by 1}
   - 5.2. name — varchar(60)
   - 5.3. description — text
   - 5.4. category_id — int(11) {default: 2 (*categories* ▸ "*Clubs*")}
   - 5.5. created — datetime
   - 5.6. modified — timestamp {default: CURRENT_TIMESTAMP}

6. courses
   - 6.1. id — int(11) {primary key; auto-increment from 1 by 1}
   - 6.2. number — varchar(10)
   - 6.3. name — varchar(60)
   - 6.4. description — text
   - 6.5. programme_id — int(11) {default: 1 (*programmes* ▸ "*Undergraduate*")}
   - 6.6. category_id — int(11) {default: 4 (*categories* ▸ "*Courses*")}
   - 6.7. created — datetime
   - 6.8. modified — timestamp {default: CURRENT_TIMESTAMP}

7. programmes
   - 7.1. id — int(11) {primary key; auto-increment from 1 by 1}
   - 7.2. name — varchar(60)
   - 7.3. description — text
   - 7.4. category_id — int(11) {default: 5 (*categories* ▸ "*Programmes*")}
   - 7.5. created — datetime
   - 7.6. modified — timestamp {default: CURRENT_TIMESTAMP}

   Pre-Filled Entries:

id	name	description
1	Undergraduate	CS Undergraduate Programme.
2	Graduate	CS Graduate Programme.

8. staff
   - 8.1. id — int(11) {primary key; auto-increment from 1 by 1}

8.2. first_name varchar(30)
8.3. last_name varchar(30)
8.4. age int(11)
8.5. photo_pathname varchar(230)
8.6. research_pathname varchar(230)
8.7. category_id int(11) {default: 1 (*categories* ▸ "*Staff*")}
8.8. created datetime
8.9. modified timestamp {default: CURRENT_TIMESTAMP}

9. undergrad_research
   9.1. id int(11) {primary key; auto-increment from 1 by 1}
   9.2. researchers varchar(255)
   9.3. abstract text
   9.4. research_pathname varchar(230)
   9.5. category_id int(11) {default: 6 (*categories* ▸ "*Undergrad Research*")}
   9.6. created datetime
   9.7. modified timestamp {default: CURRENT_TIMESTAMP}