Assignment 2:
Implementing a workflow using RESTful and WS-* Services

This assignment is worth 27 marks

Due Date I (Submission of the Source Code): Week 12 Thursday 19th October 9am through WebCMS, group submission

Due Date II (Demo): Week 12 Thursday to Friday (19th October-20th October), to be scheduled separately.

The late penalty policy is the same as Assignment 1

Introduction: Online License Renewal System

In this assignment, you will be asked to put together the different types of services we have looked at in this course to build an application. The scenario below is fictional, and it does not necessarily represent the most effective/efficient way of implementing the actual process in question. The scenario is simply created to support the learning outcomes of this assignment, although we should employ some common sense whenever appropriate.

The state government wants to automate its existing driver's license renewal process. Also, as part of the Federal Open Government project, the statement government has been asked to make the data on driver licenses available online. The internal IT department will take up the challenge and will build a new Web-based workflow application to allow the staff members in the driving license office as well as the general public to process the license renewal online. The IT department will also look into options for making the car registration data accessible via simple APIs.

The IT department has formed two teams to deliver this project; Web service team, Web application team. The service team is responsible for developing REST-based APIs. The application team is responsible for developing the Web application that implements the license renewal workflow. The two teams would communicate, but the development processes are carried out separately. The application team will rely on the APIs the service team provides, as well as other external services.

There are three parts in the assignment. Part 1 - REST services, Part 2 - Client App, Part 3 - Data service.

- Part 1 and Part 3 are worth 45% together.
- Part 2 is worth 40%
- Group meetings, overall quality of the project outcome, project management aspects are worth 15%

The License Renewal Workflow

Overview

There are two main workflow processes to be considered: one for the drivers and the other for the license renewal officers. The overall process looks like the following:

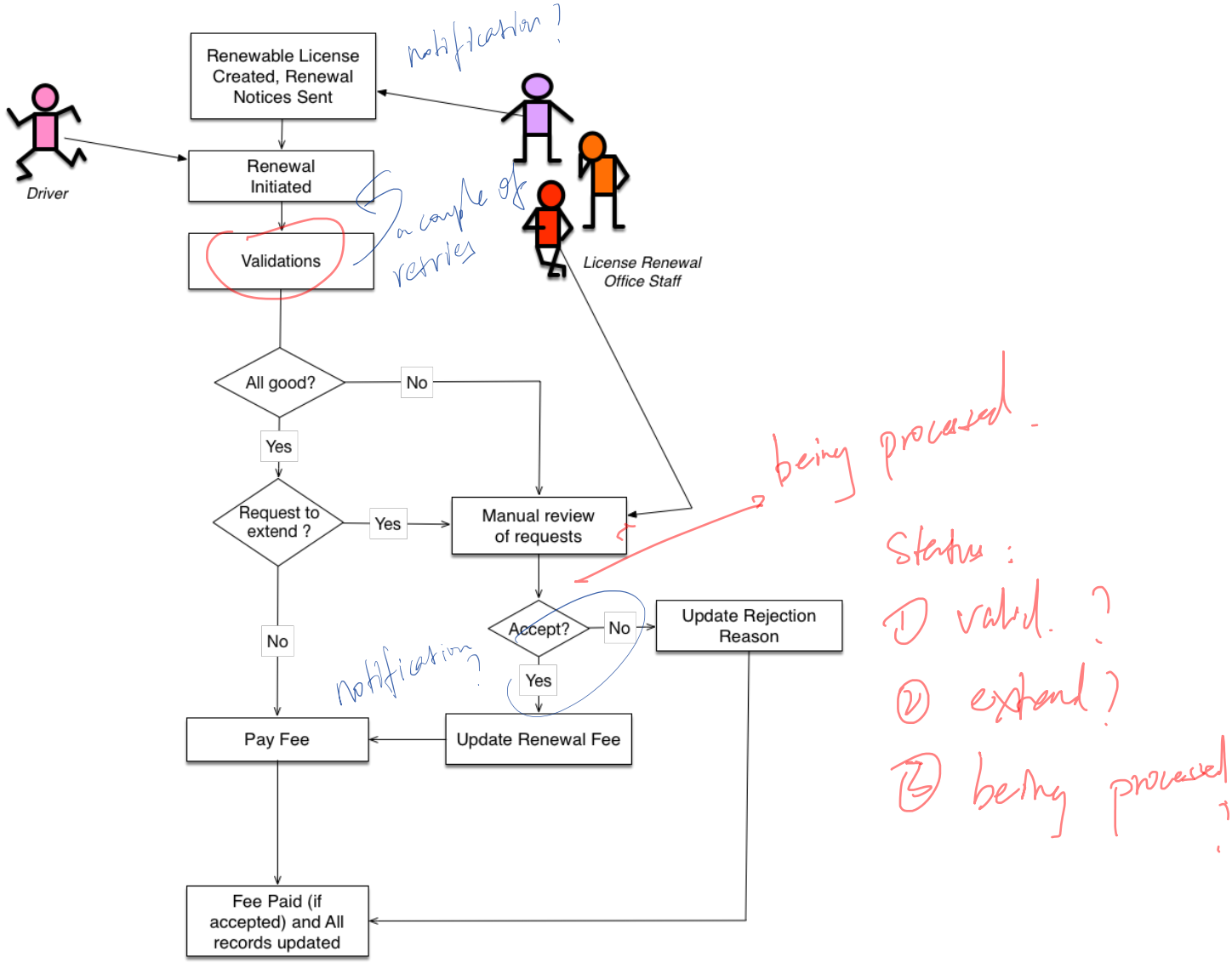

Figure 1: Overall Process

1. An officer runs a system function that generates and sends license renewal notices (e.g., those that are due for renewal in the next 60 days)
2. A driver follows a link in the notice to initiate the license renewal process.
3. The process involves validating (and updating, if necessary) the current address and contact email
    - If everything is OK, the driver pays the specified renewal fee and completes the process.
    - If the validation step fails, the renewal case is to be manually reviewed by the license officers and the driver is notified of the result later.
4. The process also allows the driver, before a payment is made, to request for an extension of the license period for extra 5 years

    - If the driver wants to apply, the renewal case is to be manually reviewed by the license officers and the driver is notified of the result later. The extra fee is also determined by the officer and added to the final amount to be paid.

Based on the overall workflow, let us look at the users (driver and office)'s view points.

The driver:

- The driver starts the renewal process by following the link provided by the system (i.e., a link in the email notice sent by the renewal system). The link serves as a unique ID of the driver throughout the process. The system does not require the driver to login (of course, the driver should be cautious not to pass the link/email to anybody else).
- At the start of the application, the system will pull relevant information for the driver in a form and the driver will need to either confirm the details (address and email) or update them. When everything is OK, a payment is made to complete the process.
- Using the given link, the driver can:
    - check the status of the renewal notice any time (e.g., during the manual reviewing step)
    - check the details of the current license
    - pay for the renewal after a manual review process is completed (e.g., if an extension is accepted, the system will detail how much the fee is set by the license office)
    - archive the renewal notice after payment (after this action, the link is no longer valid for the driver)

The license officer:

- At the start of the application, the officer can pull up all licenses that are due for renewal by certain date. Then, ask the system to generate the renewal notices and send email out to the relevant drivers.
- At any given moment, there may be cases to be reviewed manually. Such cases are presented as a list which can be ordered by their attributes (e.g., status, date)
- An officer can choose a particular case and process the case according to the relevant policy. Here, we assume that there is to be some background investigation to be done by the officer to make a decision, which is not captured in the system. The officer makes a decision (e.g., allow the driver to have an extension, or allow the driver to renew the license even after failing the validation step) which is recorded by the system so the rest of the process can complete:
    - if accepted, set the fee the driver should pay
    - if not accepted, set the rejection reason to inform the driver

Note: there can be more than one officer processing the manual cases, although a case is reviewed by a single officer. You should assume that a case may take a few hours/days to be processed, which means the system has to have a status flag so the officers know which cases are taken (i.e., being processed).

The validation step:

For this scenario, we will use a student solution from Assignment 1. The solution is provided as a docker image, so you can run it in your own docker to access the service. You will use the service as-is (i.e., no modification). Similar to what you have done in Assignment 1, the address and email validations are done by calling appropriate operations of this service.

From the application point of view, the validation steps works as follows:

- The driver is first presented with the information (address, email and other license related details) stored in the system.
- The driver can either confirm, or update (e.g., recently moved to a new address, or changed the email address) the information.
- If the driver wants to update, any new data entered (address or email) will be validated. If validation fails, the system may allow a couple of re-tries to correct the info until validation passes, otherwise, the case will be referred to the manual review process. In which case, the driver will be asked to come back and check the status of the review.
- If validation passes, the rest of the process can complete as prescribed in the scenario above.

Part 1: RESTful Services for License Renewal Management

To support the the various workflow processes involved in license renewal processes, the Web service team in the IT department has to design and build REST-based services. Note that the service team is only responsible for designing and building the services, then exposing the services through API. The Web application team will use these services to build the renewal system (application) per their requirements.

The system overview

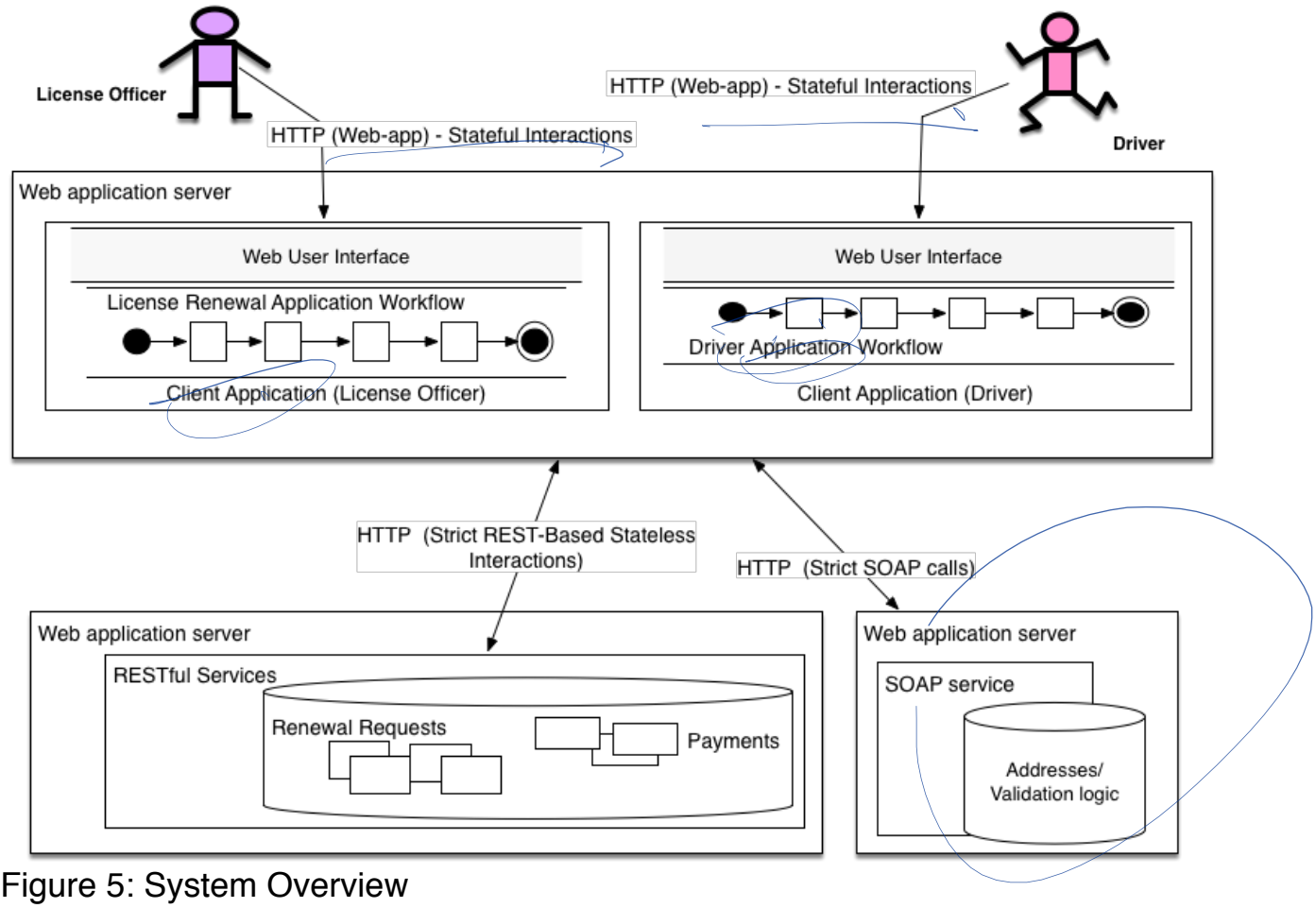

Figure 5: System Overview

The diagram above shows how the Web services API and the main client application are structured. There are three components: A main web application (for human users to use), and RESTFul and SOAP services that service the application. As mentioned before, the SOAP service is given to you and you are to use it as-is. In this section, we will look at what to do for the RESTful services part.

RESTful Services

The resources and their URI patterns:

There are three types of obvious resources to be managed: Car Licenses, Renewal Notices and Fee Payments. The specification and your own understanding about the application domain will guide how these resources might be represented and managed. It is up to you to design these resources and their URI patterns.

Some suggested information for them:

- Car Licenses: _licid, driver name, current address, license number, license class, contact email, expiry date
- Renewal Notices: _nid, address, contact email, status (new, under-review, accepted, archived, ..., etc.)
- Payments: _pid, _nid, amount, paid_date

Something to note. Let's assume that the car license resource only updated after its relevant renewal notice is processed completely. So, the renewal notice itself will have to have the 'temporary' address and contact email data contained within it. Only after everything is done, the info from a renewal notice will be copied to its car license (ie, updated address if applicable).

For the exposed IDs (i.e., URL) of the resources, it is something you have to decide how you'd like to manage them. For example, an id for renewal notice can be associated with its corresponding payment resource by two ways:

- By URI patterns: e.g., host-details/app-root/renewal/1234, host-details/app-root/payment/renewal/1234
- By resource attributes - similar to having foreign keys

Managing resources:

It is important that all HTTP methods implemented by you satisfy the safety and idempotency properties of REST operations.

On Renewal Notices:

- A renewal notice is created by HTTP POST (initiated by an officer). The response of HTTP POST **must contain** the URI/Location of the new renewal notice created by the RESTful service.
- The details of a renewal notice can be retrieved any time through a HTTP GET operation. The response of HTTP GET must contain the latest representation of the renewal notice.
- The details of an existing renewal notice can be updated via HTTP PUT.
- After the processing of the request is finalised, the renewal notice resource can be archived by HTTP DELETE operation, when requested by the driver

On Payments:

- A new payment resource for the renewal fee is created by HTTP POST. When and who will create a payment is up to you to decide. The response of this HTTP POST **must contain** a URI of the new payment and its location.
- A payment is updated by HTTP PUT. When and who will update a payment is up to you to decide.
- A payment can be retrieved by HTTP GET at any time. HTTP GET on a payment should return the latest representation of the payment resource (including NOT FOUND if it doesn't exist).

On Car Licenses:

- Unlike other resources, Car Licenses are already present (stored) with the REST services (i.e., you can start with DriversDB.txt given with the specification). So, HTTP POST on Car Licenses is not supported.

- The details of a car license can be retrieved any time through a HTTP GET operation.
- HTTP PUT to update the car license once the renewal process is completed successfully. When and who will update this is up to you to decide.

Linking resources (HATEOAS principle):

- When the server returns a response to an operation, you must consider appropriate links to other resources the service can include (e.g., by looking at the given point in the workflow, or resource relationships).
- For example, a response of HTTP GET on a renewal notice resource may also contain a link to update a Payment resource if available, or a link to update the renewal notice resource, or a link to the current car license details. The point is, the client applications should be given as much as information about possible actions on a returned resource - and they should utilise the information as much as possible.
- As part of designing your URIs for resources and the representations for request/response, think about what kinds of links you might generate for the client applications on each request. Assume that the client application does not necessarily know what to do next and you as a service can tell the client application what can be done next.

Authentication and Authorisation:

The authentication and authorisation issues in this assignment are mostly relevant to restrict the access to the appropriate operations of each resource (e.g., POST on Car License is to be done only via officers)

- All requests from the client applications have to be authenticated and authorised. For example, A GET request to retrieve the details of a renewal notice, or a PUT request to update the status of a renewal notice have to be from the right client application.
- We will make a simple assumption for this. The main Web application will send appropriate access token to the service depending on which operation it wants to access. For this purpose, the main Web application has been pre-issued with two keys: one for driver role, the other for officer role. The client application will send the key to the service with every request. The key will be sent as part of the HTTP request headers. The key should be present in all requests. You can decide on the actual key values.
- The REST services will check the key to see if the request is pre-authenticated one and then, if the client application is authorised to perform the requested operation on the particular resource.
- You will have to think carefully about which operation should be allowed on the resource and when/by-whom.

Part 2: Client Application Implementation

You will implement the client application based on the APIs provided and the workflow process specified.

The following list shows some suggestions - it is more an indication of the required functionality of the application than a complete description. You do not have to have all-out fancy highly interactive user interface. However, you are expected to create a simple, coherent, well-put-together UI for your applications. This is not a test client. We expect this to be a fully functional application that meets the specification and good usability. All the required functionality elements should be reflected in your client app design.

On Driver Applicants:

- A way to initiate the process of renewing license (and update it when appropriate).
- A way to see current/relevant details
- A way to see the detail of the renewal notice in process
- A way to make a payment for the required fee
- A way to archive renewal notices

On Officer Application:

- A way to create, and list all renewal notices, filter the list by status
- A way to see current/relevant details
- A way to 'process' a renewal notice (i.e., make a decision and then to decide on a fee payment)
- A way to get the latest status of a renewal notice or a payment

and so on ... You get the idea !!

Logging Requirements

You will be asked to show us the HTTP request/response traffic between the REST services and your client applications. That is, we will look at the pure RESTful interactions traces between the service and your client, not just the presentations shown on your client app.

For this, you should implement logging on the REST services. Each HTTP request/response interaction will be logged from the REST services so that it clearly shows the content of HTTP request and response - either in a separate log file, or out to console.

Part 3: NSW Car License Data Service (as a RESTful service):

The RMS department of the NSW statement government exposes various data sets all in HTML format: http://www.rms.nsw.gov.au/cgi-bin/index.cgi?fuseaction=statstables.show&cat=Licensing

Obviously, HTML is not really an appropriate medium for exposing data. We would prefer the data to be in XML so that we can apply XQuery or XSLT to build an data access layer over the data set.

For this part of the assignment, we will only consider the following two data sets

- Licence class by postcode as at (quarter/year)
  - http://www.rms.nsw.gov.au/about/corporate-publications/statistics/registrationandlicensing/tables/table215_2005q1.html
  - Note: the last path param responds correctly to YEAR and QUARTER. For example, 'table215_2017q1.html' returns the data for the first quarter of 2017. You only need to pull 2016 data.
- Primary licence type by postcode as at (quarter/year)
  - http://www.rms.nsw.gov.au/about/corporate-publications/statistics/registrationandlicensing/tables/table225_2005q1.html
  - Note: the last path param response correctly to YEAR and QUARTER. For example, 'table225_2017q1.html' returns the data for the first quarter of 2017. You only need to pull 2016 data.

Based on these data sets, you are to design and implement a new data service, implemented using REST service techniques. First, you need to examine the data sets and design a single XML file to represent both data sets (i.e., combine the primary license type and licence class type by postcode for all quarters of year 2016).

Implement REST-based Data Access Service serving the following queries (borrowing from OData syntax).

- URL: http:// .../data_service_root/class-and-type?&filter=quarter eq 1
  - License class and Primary license type by postcode as at first quarter, 2016 ('quarter' being the filter parameter)
- URL: http:// .../data_service_root/class-and-type?&filter=quarter eq 1 and postcode eq 2000
  - License class and Primary license type as at first quarter 2016 by postcode 2000 ('postcode' and 'quarter' being the filter parameters)
- URL: http:// .../data_service_root/class-and-type?&select=Class C,Class LR, Learner, Unrestricted&orderby=Class C&filter=quarter eq 2l
  - License class Class C, Class LR and Primary license type Learner and Unrestricted as at second quarter 2016, ordered by Class C (each license class name and primary license type being the selector parameters, order by will only accept the names included in the selector parameter)

You can use Java libraries available to parse the HTML pages. You must incorporate XSLT or XQuery in the implementation of the data service. It is up to you to decide

when you would use them and how (e.g., HTTP GET operations may run an appropriate piece of XQuery or XSLT code for generating the expected results in XML.

Group Meetings

- First meeting - present your detailed design of the REST and Data services (e.g., resources, operations, error codes, etc.)
- Second meeting - present your detailed design of the client app (e.g., UI page design to represent the flows/interactions)