OlivApp - Olive Oil Mills Management App

CMPS 356 Project Phase 1 – Web UI Design and Web API Implementation (15% of the course grade).

The project phase 1 submission is due by midnight Sunday 31st March 2019. Demos Tuesday 2nd April during office hours.

1. Requirements

You are requested to design and implement an Olive Oil Mills Management App (named **OlivApp**) for managing the milling jobs processed by an Olive Oil Mill. If interested to know the milling process then watch https://www.youtube.com/watch?v=dqDE3aSV0Ek.

OlivApp allows the mill manager to enter milling jobs and manage processing and invoicing. It will also allow customers to follow-up their submitted jobs. The main OlivApp use cases are described below.

All use cases include **Login** to allow the user (i.e., *Manager* and Customer) to login to use the application.

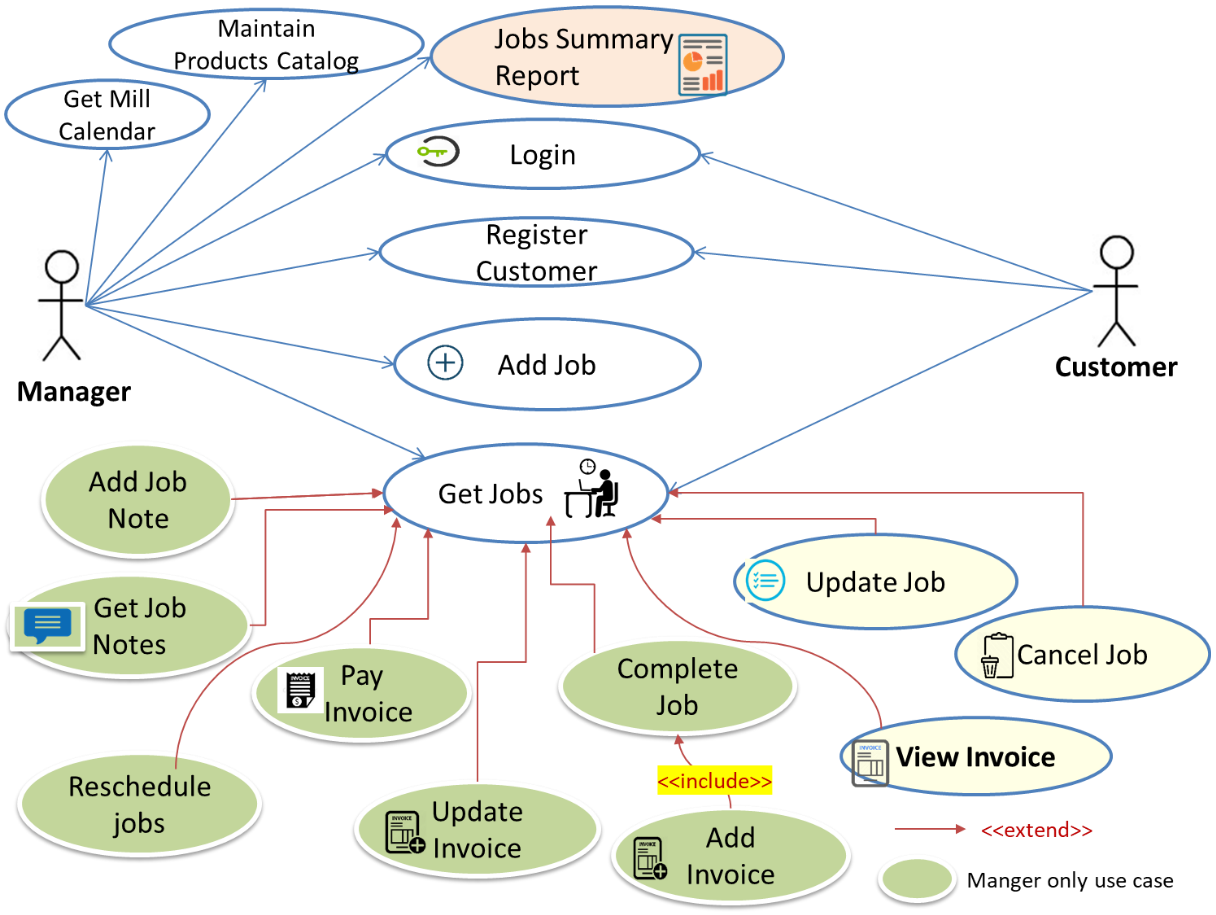

Figure 1. OlivApp Use cases

Table 1. Use cases description

Login	Allows the user (i.e., *Manager* and Customer) to login to use the application.
Register Customer	Customer or Manager can register a customer. The customer details include: firstName, lastName, street, city, mobile, email and password. The customerId should be auto assigned by the system.
Add Job	Manager can add milling job. For each job, the manager should be able to select a customer, enter the olives quantity, select olive type, harvest date, select source region, reception date time (default is the current date and time), scheduled milling date and time. - A job id should be auto-assigned to the entered job. - The job status should be auto set to **Pending**. - The manager should be able to access the ***Add Customer*** use case to make it more convenient by allowing the manager to add a customer without leaving the add job form. - Olive type and source region should be dropdowns populated from the lookup-data.json available @ https://cmps356s19.github.io/data/lookup-data.json
Get Jobs	- This should be the default use cases that the customer/manager gets upon login. - Manager can get jobs by Status (i.e., Pending, Completed, Paid and Collected, Cancelled, All) and by Customer (Select a particular customer or for all customers). By default, the manger should get Pending jobs for all customers. - Customer can get only their jobs by Status. By default, the customer should get their Pending jobs. The jobs list should allow access to other use cases associated with a job: - The manager/customer should be able to access ***update***, **complete** or ***cancel*** a job. - The manager can also update/pay the Invoice for a completed job. - The manager/customer can view the Invoice for a completed job. - The manager can Add Job note and Get notes associated with a job. - The manger can select some or all pending jobs and reschedule them (see the reschedule jobs use case for further details). The UI should allow selecting jobs (maybe a checkbox besides each job) to allow rescheduling them when the manage clicks the 'Reschedule' button.
Update Job	The customer/manager can update a job using the same UI for Add Job. - Only the manager can change the scheduled milling date and time. These two fields should be disabled for the customer. - The job status cannot be changed directly and should be displayed for both for the customer and the manager.
Cancel Job	When the manager or the customer request cancelling a job, the system should set the job status to **Cancelled**. Note that only pending jobs can be cancelled.
Complete Job	- The manager can set a job as completed by entering the completedDate and the completedTime, ***oilQuantity*** (liters of oil produced) and the Invoice details. In the UI the ***completed date*** default value should be the current date and the completedTime should be defaulted to the current time.

	- Once a job completion is submitted, the system should set the job status to "Completed" and store the submitted the details including the invoice with the job record. - The invoice should include multiple line items: - **Milling fee**: This invoice line item should be auto created. The Quantity should be auto-set to the job's olives quantity, Unit price should be auto-set to the default unit price retrieved from the products catalog. - Containers: the manager can optionally add one or many containers that the customer has purchased. For each container line item, the manager selects the container and the quantity the customer has purchased. The Unit price should be auto-set to the default unit price retrieved from the product catalog. The available quantity should be auto-reduced when containers are sold to customers. The *invoice date* should be auto set to the current date and time.
Update Invoice	Manager can update the Invoice line items for a particular job.
View Invoice	Manager or Customer can view the Invoice for a job.
Pay Invoice	Manager enters the payment method and upon saving the system auto-sets: - The payment date to current date and time. - The job status to **Paid and Collected**. The payment method should be a dropdown populated from the lookup-data.json available @ https://cmps356s19.github.io/data/lookup-data.json
Reschedule jobs	From the list returned by get jobs, the manager should be able to select some or all pending jobs and reschedule them forward or backward by certain number of days, hours and minutes e.g., reschedule all pending jobs 1h later. After the manager selects the jobs to reschedule the app should 2 things (1) Select the rescheduleMode (either reschedule *forward* or *backward)* (2) Enter the number **days**, **hours** and **minutes** Then the app should loop through the selected jobs then update the job scheduled date based on these inputs.
Get Mill Calendar	Manager can view the Mill Jobs in a **calendar format**. Each calendar entry should include the scheduled *job id* and the *customer name*. When a calendar entry is clicked, the manager should be able to access the full job details of the job id associated with the calendar entry. For the calendar implementation, you may use an open source JavaScript calendar component such as https://fullcalendar.io/ .
Add Job Note	Manager can add a Note to a job. A Note has a note text entered by the user. Additionally, the system records who created the note (i.e. createdBy attribute) and on which date (i.e. createdOn attribute).
Get Job Notes	Manager can get the notes associated with a job.
Maintain Products	Add/update the list of products and services that the Mill offers. The product details include product code, product name, category, size, available quantity, unit price. The product id should be auto-assigned by the system.

Catalog	The main products/services the Mill offers are: - Milling service: the manager can update the milling fee per Kg. The available quantity and size are not applicable for this service. - Containers: the manager can set the product name, category, type, size, unit price, available quantity and imageUrl. (The quantity should be auto-reduced when containers are sold to customers). An initial list of products is available @ https://cmps356s19.github.io/data/products.json
Jobs Summary Report	This report allows the manager to get a summary of mill jobs by status for a date range. By default, the date parameters should be set to today's date. **Completed jobs:** Number of jobs, quantity of olives processed, quantity of oil produced, average oil liters per 100 kg, invoiced amount, average tons of olives processed per hour, average waiting time from reception to completion. **Awaiting Payment jobs:** Same details as completed jobs. **Paid and Collected jobs:** Same details as completed jobs. **Pending jobs:** Number of jobs, quantity of olives, average waiting time. **In Progress jobs:** Same details as pending jobs. **Awaiting Confirmation jobs:** Same details as pending jobs. From the summary report. The manager can drill down and get the details (e.g., from the Completed jobs summary report the manager can get the details of completed jobs). You need to design the UI and Web API of Jobs Summary Report but **you do not to implement it as this will be done in phase 2.**

2. Deliverables

Seek further clarification about the requirements/deliverables during the initial progress meeting with the instructor. Note that further important clarifications maybe modified/added to the project requirements.

1) Application design documentation that includes the following:
   - 3 Class Diagrams showing Entities, Repositories and Services.
   - Discussion of design rationale (i.e., justification) of 5 key design decisions.

During the weekly project meetings with the instructor, you are required to present and discuss your design with the instructor and get feedback. You should only start the implementation after addressing the feedback received about your design.

2) Create test data JSON file for each of the entities.
3) Implement Web UI using HTML and CSS. The pages should comply with Web user interface design best practices. Also remember that 'there is elegance in simplicity'.

   Connecting the Web UI with the Web API will be done in phase 2 of the project.
4) Implement the repositories and the services (i.e., Web API) using JavaScript.
5) Document the testing of Web UI and Web API using screen shots illustrating the results of testing.
6) Every team member should submit a description of their project contribution. Every team member should demo their work and answer questions during the demo.

Push your implementation and documentation to your group GitHub repository as you make progress.

3. Grading rubric

Criteria	%	Functio nality*	Quality of the implementation
1) Application Design: • 3 Class Diagrams for Entities, Repositories and Services [5 marks each]. • Discussion of **design rationale** of 5 key design decisions [5 marks]	20		
2) Create test data JSON file for each of the entities.	5		
3) Design and implement the **Web UI** using HTML and CSS.	30		
4) Implement the repositories and the services (i.e., **Web API**) using JavaScript.	40		
5) Testing documentation using screen shots illustrating the testing of Web UI and Web API.	5		
6) **Discussion of the project contribution** of each team member [-10pts if not done]			
Total	100		
Copying and/or plagiarism or not being able to explain or answer questions about the implementation	- 100		

* **Possible grading for functionality**: ***Working*** (get 70% of the assigned grade), ***Not working*** (lose 40% of assigned grade and ***Not done*** (get 0). The remaining grade is assigned to the quality of the implementation. In case your implementation is not working then 40% of the grade will be lost and the remaining 60% will be determined based on of the code quality and how close your solution to the working implementation. Solution quality also includes meaningful naming of identifiers, no redundant code, simple and efficient design, clean code without unnecessary files/code, use of notes where necessary, proper white space and indentation.

Marks will be reduced for code duplication, poor/inefficient coding practices, poor naming of identifiers, unclean/untidy submission and unnecessary complex/poor user interface design.