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 Monday 11th March 2019. Demos Tuesday 12th March 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.

The application will allow 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.

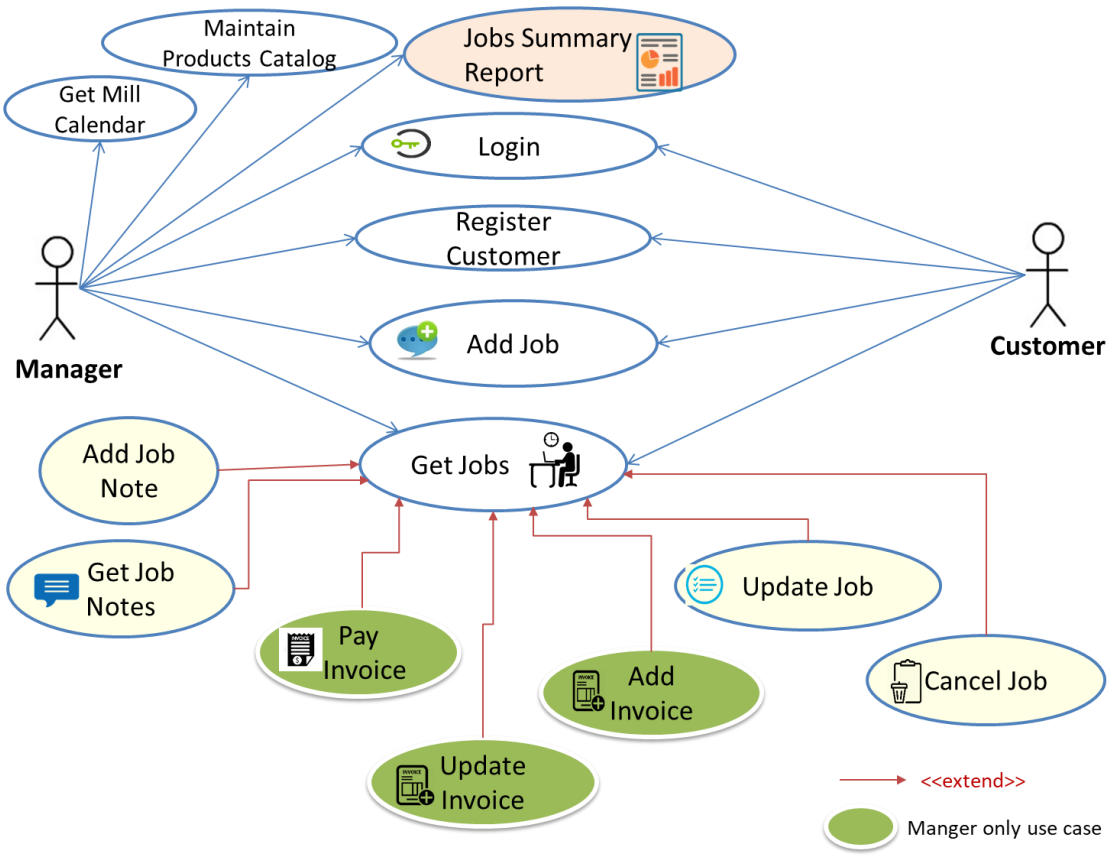

Figure 1. OlivApp Use cases

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 and email. The customer id 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, job year (e.g., 2019/2020), enter the quantity, select olive type, harvest date, select source region, reception date time (default is the current date and time), schedule date time, comment. Also, the manager needs to specify whether the customer will bring their own oil containers or purchase them from the Mill. If so the manager needs to select the container preferred by the customer (i.e., select among the available containers in the products catalog). - A job id should be auto-assigned to the entered job. - The job status should be auto set to **Pending**. - When a job is entered a notification should be sent to the customer by sms / email notifying them of their job id, quantity and scheduled data and time to process their milling job. - The user should be able to access the Add Customer use case to make it more convenient. - Olive types should not be hardcoded in the app. Example types include: Picholine, Haouzia, Menara, Arbequina, Arbosana, Picual, etc. - Customer can book a milling job and select a preferred milling date and time. The job booking should use the same UI as Add Job. The booked job status should be set auto-set to **Awaiting Confirmation**.
Get Jobs	- Manager can get jobs by Status (Awaiting Confirmation, Pending, In Progress, Completed, Awaiting Payment, Paid and Collected, All) and by Customer (Select a particular customer or for all customers). - Customer can get only their jobs by Status (Awaiting Confirmation, Pending, In Progress, Completed, Awaiting Payment, Paid and Collected, All). - From the jobs list the manager should be able to access the update or cancel a job. Completed or in-progress jobs cannot be cancelled. The manager can also update/view a Invoice for a completed job. - From the jobs list the customer should be able to access update or cancel a job. Completed or in-progress jobs cannot be cancelled. The customer can view the Invoice for a completed job. - 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.
Update Job	Update a job. - It should reuse the UI of Add Job. - Only the manager can change the job status.

	- If the job status is set to **Completed** then the manager should enter the number of oil liters produced for the job. The *completed date* should be auto set to the current date and time. Once the job is completed then the customer should be notified by email/sms.
Cancel Job	The manager or the customer can cancel a job.
Add Invoice	- Once a job is completed the Manager can add the Job Invoice: - Milling fee: This 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. The manager can enter a discount % or an amount to deduct from the line item total. - Containers: the manager can select the container type 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 manager can enter a discount % or an amount to deduct from the line item total. 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. The invoice number should be auto assigned by the system. Once the Invoice is added the job status should be auto-set to 'Awaiting Payment'.
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 (Cash, Cheque, Bank Card) and upon saving the system auto-sets: - The payment date to current date and time. - The job status to **Paid and Collected**.
 **Get Mill Calendar**	Manager can get the Mill jobs in a calendar format. You may use an open source JavaScript calendar component such as https://fullcalendar.io/ . The calendar entry should include the scheduled job ids and their customer names. The user should be able to access the job details for a calendar entry.
 **Add Job Note**	Manager or Customer can post Notes and associate them with a job. The manager/customer can respond to Notes. The system record who created the note and on which date.
 **Get Job Notes**	Users can get Notes associated with a job.
Maintain Products Catalog	Add/update the list of products and services that the Mill offers. The product details include product code, product name, description, size, available quantity, unit price. The product id should be auto-assigned by the system. 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 container title, description, size, unit price

	and available quantity. (The quantity should be auto-reduced when containers are sold to customers).
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, Invoiceed 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).

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:
   - Application Architecture Diagram.
   - 3 Class Diagrams showing Entities, Repositories and Services.
   - UI Design and navigation.
   - Discussion of design rationale (i.e., justification) of 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) 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'.

   Connected the Web UI with the Web API will be done in phase 2 of the project.

3) Create test data JSON files for each of the entities.
4) Implement services (i.e., Web API) using Node.js.
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	%	Functionality*	Quality of the implementation
1) Application Design: • Application Architecture Diagram. • 3 Class Diagrams showing Entities, Repositories and Services. • UI Design and navigation. • Discussion of design rationale of key design decisions.	18		
2) Implement **Web UI** using HTML and CSS.	36		
3) Create test data JSON files for each of entities.	5		
4) Implement the **Web API** using Node.js.	36		
5) Testing documentation using screen shots illustrating the results of UI and API testing. - Discussion of the project contribution of each team member.	5		
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.