



CMPS 350 Project Phase 1 – WebApp UI Design and Implementation (15% of the course grade).



The project phase 1 submission is due by **8am Sunday 6th March 2022**. Demos during the same week.

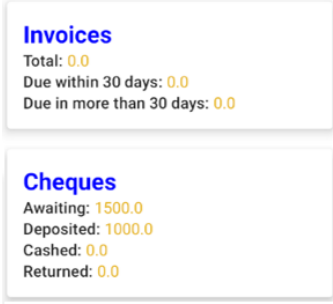
1. Requirements

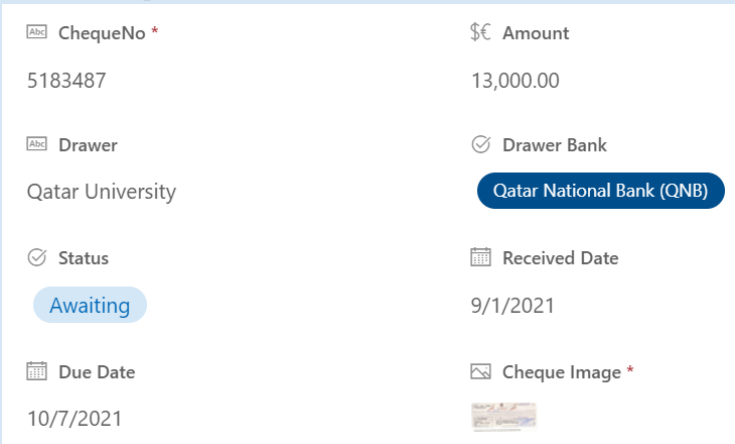
You are requested to design and implement **YalaPay** B2B Web app for managing the payments of invoices. Its purpose is to streamline the financial transactions between a company and its customers by speeding the collection of payments from customers, which has a positive impact on the cash flow of the company and its profitability. YalaPay also manages the cashing cheques and provides valuable financial reports to track pending payments.

Note that storing data in a local database and reading/writing data from remote Cloud Firestore will be done in phase 2 of the project. For phase 1, data should be kept in memory and if needed json files could be used.

The main YalaPay use cases are described Table 1.

Table 1. Use cases description

Use case	Brief description	
U1 - Login	<p>Allows the user to login. Login is prerequisite for all the YalaPay use cases described below.</p> <p>The list of users should be defined in <code>users.json</code> file. The minimum required user properties include: first name, last name, email, and password.</p> <p>No need to create a sign-up form.</p>	
U2 - YalaPay Dashboard		<p>The YalaPay Dashboard should display the summary of invoices and cheques by status as shown in the image. The dashboard should present a summary of data stored in the app database.</p>
U3 - List/Search/Add/ Update and Delete a Customer	<p>Provides the ability to List/Search/Add/Update and Delete a Customer.</p> <p>The minimum required customer properties include: customer Id, company name, address (street, city and country), contract details (first name, last name, mobile and email). When adding a new customer, the customer Id should be auto-assigned.</p> <p>The app should have an initial list of customers in <code>customers.json</code> file.</p>	
U4 - List/Search/Add/ Update and Delete an Invoice	<p>Provides the ability to List/Search/Add/Update and Delete an invoice.</p>	

	<p>The minimum required invoice properties include: Invoice number, customer Id, amount, invoice date and due date. When adding a new invoice, the invoice number should be auto-assigned.</p> <ul style="list-style-type: none"> - The app should have an initial list of invoices in <code>invoices.json</code> file. - When listing or viewing an invoice the app should display the invoice details including the balance pending payment. The invoice balance is equal to the invoice amount minus the total payments excluding the returned cheques. - From the list of invoices, the user can also select an invoice and enter the payments associated with the invoice.
U5 - List/Search/Add/ Update and Delete payments for an invoice.	<p>Provides the ability to List/Search/Add/Update and Delete payments for an invoice. An invoice can have one or many payments.</p> <p>The payment details include: the payment amount, payment date, mode of payment (cheque, bank transfer or credit card).</p> <p>For cheques more details should be entered as shown in Figure 1:</p> <ul style="list-style-type: none"> - Cheque No - Amount - Drawer - Drawer Bank (the user should be able to select a bank from the list of banks in <code>banks.json</code> that includes banks in Qatar listed here). - Status: should be set to <i>Awaiting</i>. Later the status could be changed to <i>Deposited</i>, <i>Cashed</i>, or <i>Returned</i>. - Received Date (should be auto-set to today's date) - Due Date - Cheque Image (this attribute will store an image of the received cheque)  <p>Figure 1. Example Cheque Details</p>
U6 - Manage Cashing Cheques: One of core use cases of YalaPay if to manage the cheques caching process as shown in Figure 2. This use case includes:	
6.1. Add Cheques Deposit:	<p>As shown in the illustrative image, the user can select:</p> <ul style="list-style-type: none"> - The Bank Account to deposit the cheques into.

Note the company may have 1 or many bank accounts stored in `accounts.json`. Just have a json document having the companies bank accounts no need to provide a UI to maintain them.

- The **Deposit Date** (defaulted to today's date)
- Select the cheques to deposit from the list of cheques **Awaiting** payment.

Next to the Due Date show between parenthesis the remaining days to the due date (i.e., due date – today). Make the remaining days green positive and red if negative e.g., 30/10/2020 **(+21)**, 01/10/2021 **(-11)**. Red means the cheque is late to be deposited.

Upon confirmation, the app should Create a **Cheques Deposit** having: the list of selected cheques, the *Deposit Date* and the *Bank Account* to deposit the cheques to, *Status* set to *Deposited*.

Also update all cheques included in the Deposit to set their *Deposit Date* and their status to *Deposited*.

6.2. List Cheques Deposits:

List the Cheques Deposits and provide the ability to **view**, **update** and **delete** each entry (see a sketch of the UI in the image below just to give you an idea).

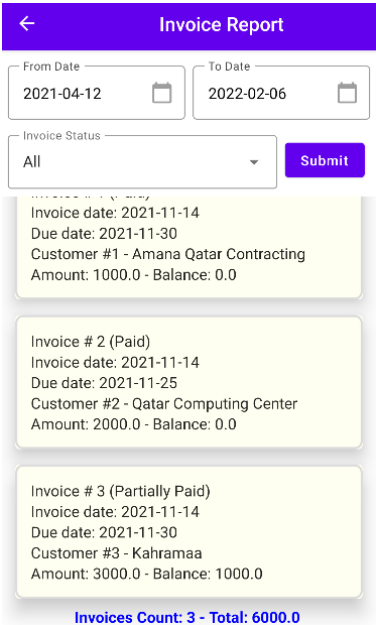
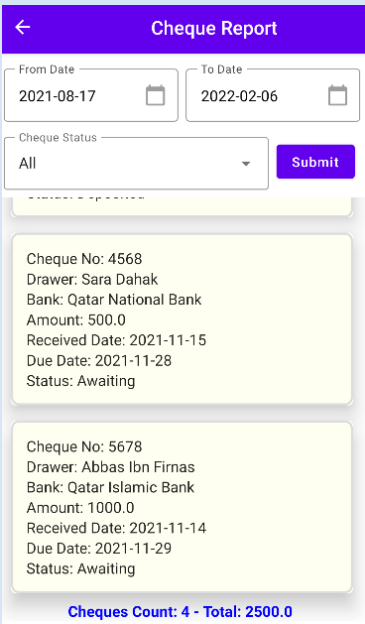
6.3. Update Cheques Deposit:

Once the Cheques Deposit has been processed by the bank (after few days) the app should allow Updating the status of the Cheque Deposit to either *Cashed* or *Cashed with Returns* and enter the **Cashed Date** (by default set it to today's date). See illustrative image below.

- If the Cheque Deposit status is set to *Cashed* then simply set the Cashed Date and the Cheques status to *Cashed* for all cheques included in the Deposit.

- If the Cheque Deposit status is set to *Cashed with Returns* then allow the user to specify the *Return Date* and the *Return Reason* for returned cheques. The user can select from a list of reasons from `return-reasons.json` file. Possible reasons of cheque return include (more info is available at this [link](#)):

- No funds/insufficient funds
- Drawer's signature differs
- Alteration in date/words/figures
- Drawer deceased/bankrupt
- Account closed

	<ul style="list-style-type: none"> ○ Cheque information and electronic data mismatch <p>Also, set the Cashed Date and the Cheques status to <i>Cashed</i> for non-returned cheques included in the Deposit.</p>
<p>U7 - Invoices Report</p> 	<p>As shown in the illustrative image, this use case should allow the user to get the invoices for a period and by status. The user can enter the from date and to date and select invoice status: <i>All</i> (this is the default value), <i>Pending</i>, <i>Partially Paid</i>, or <i>Paid</i>. Upon submission, the app should display the invoice details including the balance pending payment. The invoice balance is equal to the invoice amount minus the total payments excluding the returned cheques. Also, include a report summary including the total amount and count of invoices should be displayed.</p>
<p>U8 - Cheques Report</p> 	<p>As shown in the illustrative image, this use case should allow the user to get the cheques for a period and by status. The user can enter the from date and to date and select cheque status: <i>All</i> (this is the default value), <i>Awaiting</i>, <i>Deposited</i>, <i>Cashed</i>, or <i>Returned</i>. Upon submission, the app should display the cheque details. Also, include a report summary including the total amount and count of cheques should be displayed.</p>

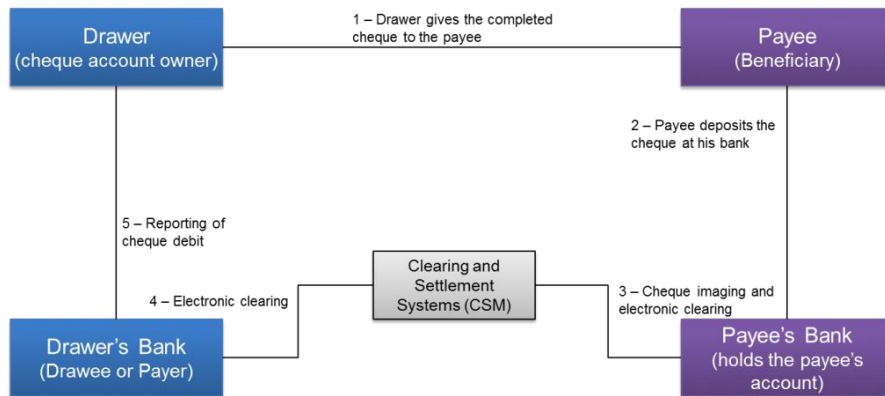


Figure 2. Cheque Cashing Process (source: <https://www.paiementor.com/the-four-corner-model-for-cheque-payment/>)

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) Design the App Web UI and navigation.

You may design the UI wireframe (sketch) to decide the UI components and the layout either on paper or use a design tool such as <https://balsamiq.com/>

During the weekly office hours, you are required to present and discuss your design with the instructor and get feedback.

- 2) For each use case, **implement the app Web UI and navigation using HTML, CSS and JavaScript**. The pages should comply with Web user interface design best practices. Also remember that 'there is elegance in simplicity'.

Design and implement the app navigation to allow the user to navigate from one page to another in intuitive and user-friendly way to achieve the app use cases.

- 3) For each use case, **implement the client-side data access repositories** using JavaScript to read/write the app data from/to IndexedDB.

Also, you should initialize the IndexedDB with data from JSON files (if the database is empty).

- 4) Application design documentation including the Entities Class Diagram and the Repositories Class diagram.
- 5) Document the app testing using screen shots illustrating the results of testing.
 - 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.

Note that this phase will be focused only **a fully working client-side implementation** using data stored in json files and local browser database. In phase 2 you will implement the server-side to move some of the computation and data management to the server-side.

3. Grading rubric

Criteria	%	Functionality*	Quality of the implementation
1) Implement the app Web UI and navigation using HTML, CSS and JavaScript. Including designing the App Web UI and navigation.	45		
2) Implement the client-side data access repositories to read/write the app data from/to IndexedDB. Also, initialize the IndexedDB with data from JSON files.	45		
3) Application Design: Entities Class Diagram and Repositories Class diagram.	5		
4) Testing documentation using screen shots illustrating the testing results. - 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 (according to Android naming conventions), no redundant code, simple and efficient design, clean implementation without unnecessary files/code, use of comments where necessary, proper code formatting and indentation.

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