A Better Laundry System for
HKU Dorms

Final Report (fyp22023)

Atiab Bin Zakaria (3035718230)

Supervised by Dr. Jia Pan

18 April 2023
Abstract

A most prominent issue faced by the dormitory residents at the University of Hong Kong (HKU) is the uncertainty mixed into the laundry process since they do not know about the availability of the machines before physically going to the laundry room. This report presents a project to modernize the process by introducing an application for users to be able to check exactly how soon they can truly begin their laundry. The method proposed is to use a camera and code to detect the time-remaining information displayed on the machine’s screen since the ageing machines at the dorms do not have ways of accessing this information programmatically. Further quality-of-life improvement features implemented include machine reservations, accepting e-payments and automated email notifications for laundry completion.
Acknowledgements

I would like to thank Dr. Jia Pan for his guidance as the group’s supervisor, the Department of Computer Science for facilitating the hardware acquisition, and Miss Mable Choi for her instructions in the written components.

My groupmate Joshua Ho deserves an especially heartfelt thank-you because this project benefitted tremendously from his drive and passion.
# Table of Contents

Abstract i

Acknowledgements ii

Table of Contents iii

List of Figures v

List of Tables vii

Abbreviations and Acronyms viii

1 Introduction 1

1.1 Background: Outdated laundry systems at HKU 1

1.2 Current solutions 2

1.3 Motivation and objectives 2

1.4 Outline 3

2 Methodology 4

2.1 Overview 4

2.2 Hardware 4

2.3 Backend 6

2.4 OCR 7

2.5 Frontend 9

2.6 Summary 9

3 Results 10

3.1 Image pre-processing 10

3.1.1 Monochromatization 10

3.1.2 Deskewing 12

3.1.3 Noise removal 14
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.</td>
<td>Raspberry Pi 3 Model B+</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2.</td>
<td>Pi Camera V2.1</td>
<td>5</td>
</tr>
<tr>
<td>Figure 3.</td>
<td>Tuya WHD09 smart plug</td>
<td>6</td>
</tr>
<tr>
<td>Figure 4.</td>
<td>Seven-segment font structure</td>
<td>7</td>
</tr>
<tr>
<td>Figure 5.</td>
<td>SSOCR algorithm visualisation</td>
<td>8</td>
</tr>
<tr>
<td>Figure 6.</td>
<td>Example before monochomatization</td>
<td>11</td>
</tr>
<tr>
<td>Figure 7.</td>
<td>Example after monochromatization</td>
<td>12</td>
</tr>
<tr>
<td>Figure 8.</td>
<td>Example before deskewing</td>
<td>13</td>
</tr>
<tr>
<td>Figure 9.</td>
<td>Example after deskewing</td>
<td>13</td>
</tr>
<tr>
<td>Figure 10.</td>
<td>Example of vertical edge case</td>
<td>14</td>
</tr>
<tr>
<td>Figure 11.</td>
<td>Example before noise removal</td>
<td>15</td>
</tr>
<tr>
<td>Figure 12.</td>
<td>Example after noise removal</td>
<td>15</td>
</tr>
<tr>
<td>Figure 13.</td>
<td>Example before dilation</td>
<td>16</td>
</tr>
<tr>
<td>Figure 14.</td>
<td>Example after dilation</td>
<td>16</td>
</tr>
<tr>
<td>Figure 15.</td>
<td>Sign in screen</td>
<td>21</td>
</tr>
<tr>
<td>Figure 16.</td>
<td>Google authentication screen</td>
<td>22</td>
</tr>
<tr>
<td>Figure 17.</td>
<td>Home screen dashboard screen</td>
<td>23</td>
</tr>
<tr>
<td>Figure 18.</td>
<td>New reservation screen</td>
<td>24</td>
</tr>
<tr>
<td>Figure 19.</td>
<td>Current reservations screen</td>
<td>25</td>
</tr>
<tr>
<td>Figure 20.</td>
<td>QR code screen</td>
<td>26</td>
</tr>
<tr>
<td>Figure 21.</td>
<td>Time remaining screen</td>
<td>26</td>
</tr>
<tr>
<td>Figure 22.</td>
<td>Add balance screen</td>
<td>27</td>
</tr>
<tr>
<td>Figure 23.</td>
<td>Stripe secure payment screen</td>
<td>28</td>
</tr>
</tbody>
</table>
Figure 24. Energy consumption screen 29

Figure 25. Example automated email notification 30
List of Tables

Table 1. API endpoints for the backend

19
## Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>HKU</td>
<td>The University of Hong Kong</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>OCR</td>
<td>Optical Character Recognition</td>
</tr>
<tr>
<td>QR</td>
<td>Quick Response</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background: Outdated laundry systems at HKU

With the rise of the adoption of internet-connected home appliance devices such as smart laundry machines and dryers in modern times, the features expected by the users of such devices have changed from simple utility to being able to conveniently access comprehensive operational information about the devices such as the status (in use or not) and time remaining for becoming available on personal devices such as smartphones and laptops. Laundry systems in the dormitories provided by The University of Hong Kong (HKU) to house its students have not kept up with the advances in technologies in this aspect of residents' living experience at the facility. In comparison to the features provided by modern machines, the laundry workflow at these residences is rather outdated. Users are always having to guess with low certainty about the availability of machines since they do not have any way of knowing the states in advance of actually going to the laundry room (likely with a heavy load of clothes). Furthermore, they are required to ensure having enough balance in a proprietary dorm-provided debit card to pay for their machine use. The only method of topping up this card is using cash at a kiosk located in a common area separated by numerous floors from the laundry room. Once these initial hurdles are cleared by a user and they have successfully started laundering their laundry items, they must ensure to note the time remaining till completion reported on the machine's display to be able to collect the load promptly. This is important to avoid unwanted handling of the clothes by others during busy times.
1.2 Current solutions

Indeed, a simple way of solving many of the issues mentioned above is to replace the currently deployed machines with newer internet-connected ones. However, it would be a significantly expensive endeavour and at the same time, would raise questions about the fate of the machines to be replaced since discarding working devices is quite irresponsible from an environmental sustainability standpoint. As noted by Shakya et al. (2021), the research in the space of modernizing existing laundry facilities is sparse. The authors of that paper highlight another group of researchers who were able to achieve the detection of whether a machine is in use or not by monitoring the power drawn by the machine in its different operational states. However, the approach taken by Shakya et al. (2021), which is based on monitoring the vibrations detectable on the machine's body, avoids having to modify the electrical power delivery mechanism to the machines which might be desirable in the context of a power source connected to numerous machines such as in a dorm environment. There are also third-party commercial services like LaundryView, which is used by over a thousand academic institutions. LaundryView installs hardware and software components on washers to create a network accessible remotely. Machines transmit data to a secure cloud-based backend, where insights are analysed and sent to user applications.

1.3 Motivation and objectives

While both of these approaches, i.e., based on power monitoring or vibration monitoring, can detect the state of the machines, neither can report how long the machine will remain in the in-use state. We believe implementing a method for users to be able to know when they can find a machine available will be a
welcome quality-of-life improvement among many others in their already busy lives as HKU students.

As described in Section 1.2, it would be convenient to be able to implement modernization projects without altering the existing infrastructure such as the electrical power source. Hence, we intend to pursue an approach that augments the machinery that is already deployed in the student residences. Specifically, we aim to implement features including reporting time remaining, washer reservation and availability, electronic payment concepts, and completion notifications. We also aim to integrate IoT smart plugs to enable insights such as energy consumption.

1.4 Outline

This report is organized into four chapters: the first of which provides the necessary context for our motivation to propose this project as well as what we hope to achieve from carrying it out. In the second chapter, we define how we plan to accomplish the goals listed at the outset, including the hardware-software architecture and backend-frontend designs. The third chapter reports and discusses the results. Finally, in the fourth chapter, the report ends with a reiteration of what we achieved and our hopes for the project moving forward.
2 Methodology

The following sections in this chapter describe details of the approach we have taken with this project. Starting with a high-level overview to facilitate a general understanding of the methods, we then proceed to provide more information about the specifics of the backend-frontend architectures as well as the supporting technologies.

2.1 Overview

The guiding principle, as discussed in Section 1.3, for deciding on an approach for us was to not disrupt the existing equipment. As such, we chose to implement our solution based on augmentation: using external hardware to programmatically extract the time-remaining information displayed on the machine’s screen. Once access to such up-to-date information is established, we could move to focus our work on presenting it to the users in an easy-to-consume medium which, in our case, is a web application. The basic strategy of recognizing the time remaining reported by the machine is to point a camera module connected to a computing unit that can process the captured images by applying optical character recognition (OCR) and extracting the information which it can then relay to the user-facing frontend application.

2.2 Hardware

The hardware components facilitating the implementation of our project are a Raspberry Pi 3 Model B+, a Pi Camera V2.1, and a Tuya WHD09 smart plug.
The Raspberry Pi is a single-board computer that serves multiple purposes: it provides the hardware interface for the camera module to connect to as well as hosting and running the pre-processing code on the images captured using the camera; Through its Wi-Fi interface, the Raspberry Pi also makes appropriate function calls to our remote API server to achieve various features such as machine reservation validation and controlling the Tuya smart plug.

The Pi Camera module is utilised to capture the time remaining reported on the machine’s display as well as scan the QR codes used for machine reservations in our system.
The Tuya smart plug forms part of our method to enforce the machine reservation workflow concept by allowing the programmatic control of electrical power delivery to the machines only when the correct reservation is availed.

2.3 Backend

We have identified the FastAPI web framework for implementing the application programming interface (API) functionality on the backend for the frontend application to consume the extracted data. The rationale for choosing FastAPI is that it is a modern, fast (high-performance) web framework for building APIs with the Python language with a focus on standard Python type hints which aids in avoiding a common class of bugs in software development associated with incompatible data types. The FastAPI code is meant to additionally be supported by the logic to control the camera module by which it captures an image of the machine’s screen. The FastAPI application connects to a MySQL database instance to maintain necessary bookkeeping information organised. When it comes to controlling the Pi Camera and reading QR codes programmatically in Python, two libraries come into play: picamera and pyzbar. The picamera library is the standard option for managing Pi Cameras and offers a user-friendly and straightforward interface. On the other hand, the pyzbar library is a Python wrapper for the ZBar barcode scanner library, which allows for quick and efficient decoding of quick response, or better known as, QR codes.
and barcodes from images and video streams. We use QR codes to encode information about machine reservations which are decoded with the help of picamera and pyzbar.

2.4 OCR

Using an OCR engine, the textual data is extracted and made available for access by other applications (the frontend application in our case) via an HTTP endpoint of the backend API. Initially, we planned on using the Tesseract OCR engine (whose development is led by researchers at Google). However, in our testing, we have found its performance to be unsatisfactory in terms of execution time on the Raspberry Pi hardware as well as significant inaccuracy for the seven-segment font used in the machine displays. Hence, an alternative named SSOCR which is specifically developed for seven-segment fonts has been identified.

![Seven-segment font structure](image)

*Figure 4 Seven-segment font structure*

Instead of relying on machine learning like Tesseract, SSOCR uses a technique similar to template matching. The overall process is split into two main stages: segmentation and character recognition. Segmentation involves dividing the image into multiple regions, while character recognition involves determining the digit represented by each region. However, it should be noted that SSOCR
requires images to be in optimal condition, such as being monochrome, free of noise, and not skewed, because its algorithm is designed to detect seven-segment digits. Therefore, pre-processing steps are necessary to ensure that the images meet these requirements.

The SSOCR algorithm for segmentation starts by examining the image from the left side until it comes across the first pixel that belongs to the specified foreground colour, which indicates the presence of a digit. This location is used as the left boundary of the digit. The algorithm then continues scanning the image until it reaches a column consisting entirely of pixels of the specified background colour, which is used as the right boundary of the digit. This process is repeated until the desired number of digits has been found or no more digits can be identified. The same process is followed vertically to identify the top and bottom borders. The only difference is that gaps are allowed vertically because certain digits such as a zero have an empty space in the middle.

![Figure 5 SSOCR algorithm visualisation](image)

When trying to identify which digit an object represents, the algorithm for character recognition examines which sections are present or absent. For instance, the number 1 consists of two parts, while the number 8 has all seven. To locate the three horizontal segments of an eight, a vertical scan is conducted. Any foreground pixels in the upper, middle, and lower thirds are counted as part of the top, middle, and bottom segments, respectively. If a digit lacks any pixels in the middle third, it is deemed to have no middle segment. A
comparable process is applied to determine if the digit contains the other four segments, namely, two on the left and two on the right. The recognized segments are subsequently utilized in a table lookup, which is implemented as a switch statement, to identify the digit that is displayed.

### 2.5 Frontend

The tooling of choice for designing and architecting the frontend application is the Next.js framework. The main draw for Next.js is that it is based on the powerful React library for web frontend applications while also providing convenient features such as file-based routing and server-side rendering which allows for building web applications with maintainable architectures. Using React simplifies the process of creating and managing applications because it employs a structure based on components, which enables developers to divide user interfaces into smaller, reusable elements.

### 2.6 Summary

At the core of the system is the Raspberry Pi, which hosts the code for controlling the camera. The backend and frontend applications are deferred to be hosted on a separate, dedicated machine. The photo of the machine screen captured by the camera is passed to the backend application which applies pre-processing steps such as clipping and a monochrome filter to the image before feeding it into the OCR engine. Once the optical character recognition is complete, the result is consumed by the FastAPI code to make it accessible to the frontend application via an HTTP endpoint.
3 Results

This chapter lays out the outcomes of the implementation of our methods. Specifically, our image pre-processing steps are discussed at length first, followed by our methods of controlling the Raspberry Pi. The discussion then moves to describe the database schema and API design. The third-party services for payment processing and user authentication are discussed next. Finally, the user interface for the completed application is presented.

3.1 Image pre-processing

As previously mentioned in Section 2.2, before SSOCR can process the images, they must undergo pre-processing, which consists of three steps: monochromatization, noise removal, and deskewing. The algorithm requires a black-and-white version of the image to distinguish between the foreground digit pixels and background pixels. Any foreground pixels not properly identified could result in incorrect output. Noise elements must be eliminated because they can be mistaken for part of the digit by the SSOCR algorithm, leading to errors in the output. In some cases, light indicators on washer displays may cause issues during conversion to monochrome, as the noise from these indicators can be incorrectly identified as part of the digit. Deskewing the image is not always necessary, but it is a good insurance to ensure accuracy in case the camera is tilted during the pre-processing steps.

3.1.1 Monochromatization

The initial step involves converting the image to a black-and-white format. This can be achieved by changing the colour of the displayed time to either black or white, and in turn, changing the colour of the rest of the image to the opposite
colour. To carry out this conversion, a masking process is applied to the image. This process involves changing the colour of pixels to either black or white based on a specific condition. We elected to change the colour of all pixels that match the colour of the displayed time to white. However, it is important to have prior knowledge of the colour representation to use it as a parameter: Different colour spaces can be used to represent colours, which all use numerical values to describe the properties of a colour. RGB, an acronym for red, green, and blue, is a commonly used colour space that defines colour based on the intensity of these three primary colours, while HSV is an alternative way of representing colours, where the acronym stands for hue, saturation, and value. In the latter colour space representation, hue refers to the actual tone of the colour, saturation represents the intensity or purity of the colour, and the value represents the brightness of the colour.

![Figure 6 Example before monochomatization](image-url)
To extract the displayed time, we have chosen to use the HSV colour space instead of the RGB colour space. This is because HSV is more resistant to changes in lighting, which means that shades of the same colour have similar hue values but significantly different RGB values. Additionally, because the colour information is separated from the luminance, it is easier to define the necessary colour range. By utilizing OpenCV and NumPy in a few lines of Python code, it is possible to convert an image to black and white.

### 3.1.2 Deskewing

The next step involves straightening the image. We achieve this by determining the angle of rotation required to straighten one of the digits and applying it to the entire image since the digits are in a single line. Using the findCountours and minAreaRect functions provided by OpenCV, we obtain the angle of rotation, which, in turn, is used to straighten the entire image.
This angle is the angle produced by the horizontal axis of the image and the long axis of the bounding box. The success of this method is contingent on a digit being the largest object in the image, which is usually the case since the noise elements tend to be attributed to reflections and indicator lights.
An edge case that might arise from time to time is the resulting image becoming vertically oriented. This can be solved by adding one more step that checks if the minAreaRectangle result is wider than the image itself which signals that the vertical case has been encountered. The solution is to simply rotate by 90 or 270 degrees, as appropriate for the direction of the slope.

3.1.3 Noise removal

As part of the pre-processing steps, we eliminate any extraneous elements from the image that are not part of the time. These extraneous elements, also known as noise, will be perceived as digits by the SSOCR algorithm and could potentially disrupt the accuracy of the results, as previously mentioned. Various types of noise could be present, such as the light indicators visible that might be turned on to signal various information about the laundry cycle and spin. Additionally, the colon symbol separating the numbers, i.e., ':', must be eliminated because it could negatively affect the OCR's output.
The primary approach for eliminating non-digit foreground pixels involves identifying bounding boxes around all possible contours in the image. Once the digits have been separated, SSOCR is run to identify the extracted digits. To do this, we compare the bounding boxes of each contour to identify the digits we want to extract and exclude other visual elements. The OpenCV library comes packaged with the “boundingRect” function which produces bounding boxes of the same area and height for all digits except “1” in our application. Still, we can modify our criteria for identifying digits by their bounding box shape for the exceptional case to roughly \( \frac{1}{4} \)th the area of the others but with the height staying the same.

![Figure 11 Example before noise removal](image1)

![Figure 12 Example after noise removal](image2)

Finally, with the digits separated, a new solid black image is generated as a canvas to paste the extracted digits into. This is the ultimate image that is consumed by SSOCR after completing all pre-processing steps.
3.1.4 Miscellaneous pre-processing

Sometimes, based on the lighting conditions, the camera might produce images where the digits have significant gaps between the segments that make up the digits.

![Figure 13 Example before dilation](image1)

![Figure 14 Example after dilation](image2)

In such cases, the solution we adopt is to use the dilation capabilities provided by OpenCV to fill in the gap pixels with the foreground colour.

3.2 Controlling the Raspberry Pi

Two scripts are executed to control the Raspberry Pi and camera module: search_qr.py and start_ocr.py. The search_qr.py script runs in an infinite loop, constantly waiting for a QR code to be scanned by the student. Once it receives an input QR code that it can validate from the server, the other script is executed. The start_ocr.py script contains code used to extract the remaining
time from the reserved washing machine's display and it runs until the end of the reservation is reached. For the subsequent reservation, search_qr.py then starts running again to repeat the cycle.

search_qr.py depends on several libraries such as Pillow, pyzbar, and picamera, which are utilized in an infinite loop to implement the ability to scan QR codes. The image starts in the form of a binary object, which is then parsed into a PIL.Image object using the Pillow library. The pyzbar library is then used to decode the information stored in the QR code image. This value is passed to the FastAPI server for validation. If the server responds with a successful 204 status code, the time extraction script (start_ocr.py) is triggered, and the smart plug is turned on. If an unsuccessful status code is received, the system continues to search for a valid QR code until the next student submits one for a reservation. Additionally, the server returns the end date to instruct how long start_ocr.py should run if the QR code is valid.

### 3.3 Database schema

The database structure consists of four tables: students, reservations, energy_consumption, and prices. The students table is populated with information about the dormitory where a student lives, with three fields: email (used for unique identification), dorm_name (the name of the dorm), and dorm_id (the Tuya cloud asset identification number corresponding to this dorm). The dorm_id field is necessary for the application to display only the washers available in the student's dorm, which are obtained via an API call to the Tuya cloud platform.

The reservation table holds information about each student's individual reservation. It includes fields such as email (used for unique identification),
start_date and end_date (the date and time when the student can start and stop using the washer respectively), qr_code (used to activate the smart plug), washer_id (the identification number of the washer), time_remaining (the current time displayed on the washer), and status (the current state of the reservation, with possible values of incomplete, in_progress, or completed). The status field is used to inform users about the status of their reservations through the application.

The energy_consumption table is composed of the three fields date, email, and energy (calculated cumulatively via API calls to the server using information provided by the smart plugs).

The prices table is used to maintain the prices of washers per cycle, which we arbitrarily populated for example.

### 3.4 API design

The design of the API was based on the principles of RESTful architecture. This included using clear and consistent names, as well as a hierarchical structure for the routes. For instance, routes related to reservations were grouped under the “reservation” route. The HTTP methods were chosen appropriately to reflect the intended usage of the routes, with “PUT” used for updating and “POST” used for creating. The endpoints return appropriate HTTP status codes, such as “204 No Content” for successful operations with no return values, “404 Not Found” for resources not found in the database. Additionally, the API follows the conventional use of path and query parameters, with the former parameters indicating specific resources (e.g., reservations) and the latter used for filtering.
<table>
<thead>
<tr>
<th>Method</th>
<th>Route</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/create-checkout-session</td>
<td>Create e-payment session for topping up user balance.</td>
</tr>
<tr>
<td>POST</td>
<td>/email</td>
<td>Notify user about laundry completion via email.</td>
</tr>
<tr>
<td>PUT</td>
<td>/energy</td>
<td>Update database with energy usage data.</td>
</tr>
<tr>
<td>POST</td>
<td>/payment</td>
<td>Record new payment on checkout.</td>
</tr>
<tr>
<td>POST</td>
<td>/reservation</td>
<td>Insert a new reservation.</td>
</tr>
<tr>
<td>GET</td>
<td>/reservation/{email}</td>
<td>Retrieve the list of reservations made by the user (shown in the app).</td>
</tr>
<tr>
<td>DELETE</td>
<td>/reservation/{qr_code}</td>
<td>Delete a reservation.</td>
</tr>
<tr>
<td>PUT</td>
<td>/reservation/{qr_code}/status</td>
<td>Update the reservation status (e.g., to in_progress after reservation is availed).</td>
</tr>
<tr>
<td>GET</td>
<td>/reservation/{qr_code}/time</td>
<td>Retrieve the current time remaining of the washer for that reservation.</td>
</tr>
<tr>
<td>PUT</td>
<td>/reservation/{qr_code}/time</td>
<td>Update the time remaining of the washer for that reservation.</td>
</tr>
<tr>
<td>GET</td>
<td>/reservation/check</td>
<td>Check if the given dates given by the user overlap with existing reservations.</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>Endpoint Path</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>GET</td>
<td>/users/{email}/dorm-info</td>
<td>Retrieve the user's dorm information.</td>
</tr>
<tr>
<td>GET</td>
<td>/washers/price/{washer_id}</td>
<td>Retrieve the price of individual washers.</td>
</tr>
<tr>
<td>GET</td>
<td>/users/{email}/balance</td>
<td>Retrieve user's balance.</td>
</tr>
<tr>
<td>GET</td>
<td>/users/{email}/transactions</td>
<td>Retrieve user’s transaction history</td>
</tr>
</tbody>
</table>

### 3.5 Payment processing

Stripe is a well-known platform for processing online payments in a secure and efficient manner. In addition, Stripe offers a variety of payment methods, including Apple Pay and Google Pay, which makes paying for laundry a seamless experience for students. The pricing model for payments has been chosen to be in the form of top-up payments, with choices of HKD 20, HKD 50, and HKD 100. Stripe also provides webhooks, which are URLs that are called upon successful payment, enabling the database to be updated securely and accurately to record users' balances.

### 3.6 User authentication

In this project, we utilized Next.js, which comes with built-in support for authentication from several well-known providers such as Auth0, Firebase, and Supabase. After considering the widespread use of Auth0, which is a cloud-based identity management system that uses Google accounts, we decided to use it for our authentication needs. Unlike implementing custom login systems, which require developers to securely handle sensitive user information, Auth0 provides a comprehensive solution that includes a secure infrastructure for
utilising Google's battle-tested authentication service and a user-friendly login interface that has been tested extensively.

3.7 User interface

This section introduces the screens corresponding to the various features of the app.

3.7.1 Sign in

The initial sign in title screen contains a button that starts the Auth0 authentication with Google.

![Sign in screen](image)
3.7.2 Dashboard

After logging in, users are presented with a homepage that displays all the washers available in their dorm. The app's pages consist of three elements: a header, a sidebar, and a main content area. The header displays the user's email, dorm, remaining balance, and a sign-out button. The sidebar has links to pages for “Washers,” “Your Reservations,” “Add Payment,” and “Energy Consumption”.

Figure 16 Google authentication screen
The “Washers” page is the default homepage and shows different sections for each washer, indicating their current availability and price. The availability here indicates whether a washer is currently reserved, and users can reserve it for future use by clicking on it.

### 3.7.3 New reservation

When a user clicks on a washer from the homepage, the reservation form appears. The user can select their preferred date and time, and the system will check if the information entered is valid before confirming the reservation.
The input must not be empty, the duration of the reservation must be greater than zero and set in the future, and there should be no overlap with any other existing reservations.

### 3.7.4 Current reservations

Users have the option to go to the “Your Reservations” page where a list of their booked reservations is displayed containing information about the washer, the time slot, and its current status. The reservation can have one of three possible statuses: Not Completed, In Progress, or Completed.
When a reservation is Not Completed, it means that the user has either not yet started using the washer because it is not yet their turn or they have already started their turn but have not yet provided the QR code to use the washer. The status changes to In Progress after the QR code is presented and to Completed when the end time is reached.
The behaviour for clicking on a reservation depends on its status: for Not Completed reservations, the QR code for availing the reservation is presented.
For ones with the In Progress status, the remaining time is shown. Users are provided with the option to delete Completed reservations.

### 3.7.5 Balance and payments

On the “Add Payment” page, users can top up the balance on their account to pay for the laundry services. Users have the option to add HKD 20, HKD 50 or HKD 100, and a table of their transaction history is presented underneath.

![Add Balance Screen](image)

**Figure 22 Add balance screen**
When one of the payment amounts is clicked on, the user is redirected to the Stripe payment gateway and once the payment is confirmed, they are redirected back to the laundry app.

### 3.7.6 Energy consumption

Users can gain insights about the energy usage associated with their laundry by navigating to the “Energy Consumption” page.
They are presented with a graph of their energy usage plotted against the days they did their laundry, along with some interesting facts and statistics at the bottom.

### 3.7.7 Email notifications

An automated email is sent to the user any time a 0:00 time is detected by the system to notify the user about the completion of the machine's laundry cycle.
We avoided sending an email simply at the end time of the reservation since a user might have consecutive reservations.

4 Conclusion

Anecdotal evidence of one of the most common issues that cause dissatisfaction among residents of HKU dormitories is the friction involved in the process of completing their regular laundry. We have identified an aspect of this problem that might significantly improve the whole process: Providing a degree of certainty about the availability of a machine. This report has described how we are proposing to attempt this modernization effort via a combination of hardware and software.
Moving beyond the proof-of-concept phase, we have identified several areas of improvement. The most pertinent is expending effort to integrate it with HKU’s user management workflows. Further improvements may include the addition of an admin page and a mobile app version. An ambitious pursuit for making lives easier at HKU dorms could be to centralize common services such as laundry, printing, and air-conditioning into a single, convenient platform.
References