Project Log

The project log is laid out through a table and will be updated weekly, showing the progress made each week over the development process. It also includes the stage of the development plan I am currently in (shown in the time plan in the Interim Report and Dissertation). I aim to follow the time plan for development but there is expected to be some delays, e.g. time off over Christmas. Because of this, I have left a significant amount of spare time at the end of the time plan to overrun into if I need to. However, I am expected to catch up on delays due to having more free time over the holidays, but the spare time at the end is still there in case I need it. All stated progress made for each week is backed up by my evidence of frequent commits on my GitLab repository, accompanied by detailed Commit Messages.

Week Commencing	Progress	Stage(s) of development in time plan
21/10/24	• Researched other existing vending machine systems. • Identified positive and negative features of these systems.	**1. Research & Planning**
28/10/24	• Based on research, decided upon which features from other systems are good enough to implement in my project, which features I can improve on from other systems and which additional features I can add in.	**1. Research & Planning**
04/11/24	• Defined Aims and Objectives for my project. • Identified challenges and originality of my project. • Compiled these together as the first section of the Interim Report.	**1. Research & Planning**
11/11/24	• Defined System Requirements for my project. • Created the project time plan by splitting up the development process into 6 stages and identifying which tasks need to be completed in each stage and giving a realistic time scale for each task. • Coded a Gantt Chart for the development stages using the Matplotlib library in Python. • Added the time plan and Gantt Chart to the Interim Report	**1. Research & Planning**
18/11/24	• Used previous research on other systems and conducted new research on other sources about vending machine functionality and user interface design to complete Literature Review section of Interim Report • Outlined my design process, integration, algorithms and data structures to complete Outline of Specification and Design.	**1. Research & Planning**

	• Added these sections into the Interim Report and finalised and submitted it.	
25/11/24	• Designed the system architecture using Spring MVC framework • Planned the database layout • Began creating wireframes for the User Interface	**2. System Design**
02/12/24	• Finished wireframes • Created project using Spring Boot and Gradle build tool • Added dependencies for Spring MVC and other required libraries.	**2. System Design** **3. Backend Development:** — 3.1 Set Up Backend
09/12/24	(No time spent on project, focused on completing coursework's for other modules)	
16/12/24	• Created entity classes and repositories for Product and Transaction tables • Set up database in MySQL and connected it to my project • Created service class for Product • Added a test method to check all database related functionality is working	**3. Backend Development:** — 3.2 Manage Products (Stock) — 3.3 Handle Transactions
23/12/24	(Some time off for Christmas) • Resolved issues to allow me to successfully connect my project to my GitLab repository • First commit of my project pushed to GitLab	**3. Backend Development**
30/12/24	• Fixed problem involving uploading to wrong GitLab branch by merging with the main branch • Uploaded Project Log and wireframes to GitLab • Implemented a findbyID method to allow me to create the method for updating stock • Amended database structure to allow a many to many relationship between products and transactions • Edited Product and Transaction class to implement new database structure • Created TransactionService class and added methods for fetching products, calculating total cost, validating payment and deducting stock	**3. Backend Development:** — 3.2 Manage Products (Stock) — 3.3 Handle Transactions

	• Tested new methods using sample printing methods in the main class	
06/01/25	(No time spent on project this week due to 2 exams and 1 coursework submission for other modules to prepare for)	
13/01/25	• Made changes to improve existing code, e.g. used Transaction constructor when creating a transaction instead of multiple lines of setter methods for each attribute • Created generateReceipt method in TransactionService class, which creates a formatted receipt for a transaction • Created TransactionController class and added a method to test the generateReceipt method works as intended and to test the REST API endpoint is working	**3. Backend Development:** — 3.3 Handle Transactions — 3.4 Generate Receipts
20/01/25	• Basic testing of product management and transaction logic features using testing methods and the TransactionController with the API endpoint for receipts • Altered project setup to allow database tables to be automatically created by the entity classes upon running rather than relying on prior creation in mySQL. Also inserting product data from external SQL file stored in the project folder rather than from mySQL. This is to fix errors and conflicts arising from the database creation and data insertion • Started frontend development using HTML and Thymeleaf: created and completed Home Page, created template for Main Page, Instructions Page and About Page • Added general styling for all pages using an external CSS stylesheet • Added specific styling for the Home Page using another external stylesheet • Created Navigation Controller and MainPageController with basic GET Mapping methods to allow dynamic movement between pages through button clicks	**3. Backend Development:** — 3.5 Testing & Debugging **4. Frontend Development:** — 4.1 Webpage Layout — 4.3 Styling
27/01/25	• Added a new method to ProductService which fetches all products from the product repository and returns it as a list	**4. Frontend Development:** — 4.1 Webpage Layout

	• Edited MainController to now display the mainpage html file and pass in the products list as a model attribute • Began developing the main page by adding in a machine with all the products displayed in a grid, a numbers and letters keypad with control buttons CLEAR and ENTER, and a cart section to display items in the cart and the running total • Created a specific CSS file to style the main page • Meeting with supervisor to discuss project progress and prepare for interview • Created external JavaScript file for main page • Added JavaScript functions for full functionality of keypad: entering keys on to display, restricting key input for valid item codes only, clearing display, deleting last character from display and submitting the code on the display (with more validation) • Added more JavaScript functions and AJAX implementation for full cart functionality: dynamically display items in cart with their name, ID, price and quantity. Update item quantity and price when duplicates are added. Buttons to reduce quantity and remove items from cart. Display and auto update total cost for whole cart. Button to clear the cart. • More complex CSS styling to maintain usability and consistency.	— 4.2 Cart Functionality — 4.3 Styling — 4.4 Dynamic Features
03/02/25	• Edited JavaScript methods to become asynchronous and use 'awaiting' to ensure AJAX requests are properly carried out • Implemented more AJAX calls to check with database that there is sufficient stock of an item before adding it to the cart (or updating its quantity) • Added a '+' for each item in the cart so the quantities can be increased easier and quicker • Amended database layout so no settings are required to be changed between first run of the application and subsequent runs	**4. Frontend Development:** — 4.1 Webpage Layout — 4.3 Styling — 4.4 Dynamic Features

	• Implemented payment logic using a HTML Modal display which replaces the cart on the page. Contains dynamically updating remaining amount and inserted amount displays as well as coin buttons. Also contains a change display and confirm payment button which both only appear when the amount has been paid (or overpaid for change). JS Confirm alert checks the user is happy to proceed, then completes the payment. • Created separate JavaScript file containing multiple functions to handle payment logic • CSS styling for payment modal	
10/02/25	• New Controller method in TransactionController to handle AJAX call from frontend and allow storing transactions into the database after payments are made. • Modified backend Service and Controller methods to properly handle products names, quantities and prices as well as transactions for necessary frontend features. • Implemented deduction of product stock when transactions are completed. • Added proper rounding for numerical values to ensure proper displaying and avoid floating point errors when storing data. • Edited HTML and JavaScript in cart to auto enable/disable cart buttons when cart is empty/not empty. • Modified DB setup to prevent loss of changes to product stock every time the app is run. • Added new success modal item which appears when payment has been completed, displaying the total paid and change given (also to display purchased items later on). Fully functional containing a close button which closes the modal, a home button which navigates back to the home page and a view receipt button which navigates to the receipt page. Fully CSS styled. • Implemented new receipt HTML page which displays all the items purchased with their names and individual prices, the total cost,	**3. Backend Development:** — 3.4 Generate Receipts — 3.5 Testing & Debugging **4. Frontend Development:** — 4.1 Webpage Layout — 4.2 Cart Functionality — 4.3 Styling — 4.4 Dynamic Features — 4.5 Frontend Testing

	amount paid, and change given. Formatted and displayed on the page to look like a real receipt. Created new ReceiptController with a method to fetch all the necessary transaction data, using Service methods and add to the model for the receipt page to display. Receipt page contains a button to navigate back to home page and another button to download the receipt as a .txt file. Fully functional page using JavaScript and fully styled using CSS. • Modified database layout to remove the many to many relation (replaced by separate one to many relations) and explicitly create the TransactionProduct entity, which now also stores the quantity of each product in the transaction. Created new repository class for the entity and edited Transaction and Product entities to implement the new relation. Receipt Controller method and Transaction Service methods also edited to reflect the changes. • Thorough testing of changes, refreshed database schema, fixing errors/warnings and added logging to help manage errors and crashes better. • Preparation for Interview: began creating slides and practicing demo.	
17/02/25	• Added in automatic stock updating, by checking stock for each item in a transaction after successful purchase. If it falls below a set threshold, automatically increase stock by fixed it amount and send a JS alert to the user notifying them of the update. • Added in Product images. Modified Product entity to store an image url, relating to the file path of a specific image stored in an images folder in the resources/static directory. Added in 16 new images, one for each product to the images folder. Updated data.sql to insert proper product names and the new image urls. Changed mainpage.html to display the images in the items grid. New JavaScript function to add the images of the items purchased in a transaction to a new container in the success modal.	**Additional Features/Improvements**

	• Created new proper user manual and added it to the pdf viewer in the instructions page. Added general instructions for using the vending machine and for making payments to the mainpage. Modified the openModal and closeModal JavaScript functions to toggle between general instructions and payment instructions based on whether user is using the machine or paying. • Completed Project Interview • Added in basic logging in and registration functionality. Created new User entity and repository for storing user login details in the database. Created new login and register controllers to handle logging in, registering and switching between each other. Registering successfully navigates to login page and logging in successfully navigates to home page. Created new HTML pages for both and styled them. • CSS styling for all new elements.	
24/02/25	• Enhanced the Login and Register system using Spring Security. Added in Spring Security dependency to build.gradle file. Created SecurityConfig class to manage page access, handle authentication, allow login, logout and registering and to use encryption for storing passwords in the database. The class contains a SecurityFilterChain to define how all requests are authenticated. Also created UserDetailsServiceImpl class which implements Spring Security UserDetailsService, to verify the user exists then retrieve and return user details from the database when a user logs in. Modified the existing Login and Register controllers to implement the new Spring Security setup. Created logout button and user info display on the homepage, to show the username of the user currently logged in (or if not shows "Guest") and allow the user to logout back to the login page. Finally, more CSS styling improvements for the 'authpages' (Login and Register) • Implemented enhanced product display. Created new Item View Modal in mainpage.html, with enlarged item image,	**Additional Features/Improvements**

	and display of product details: Name, Code, Price and Stock. Added in JavaScript functionality to display the modal containing the information specific to an item when that item is double clicked in the display. Modified the main page HTML file to correctly store each product's attributes so they can be displayed properly in the modal. Modal also contains a button to add the item to cart directly (without requiring the keypad), created a simple JavaScript function for this, which calls the existing addItemToCart function and passes in the relevant item code. CSS styled the modal. • Added proper stock display updating. Edited the mainpage HTML file so that each item can have a class added to it to say whether or not it is out of stock based on if its quantity equals 0. Out of stock items are styled to not allow the cursor or clicking, to be half faded out and to have a big red 'X' covering them. Also created two new JavaScript functions, which are called after a transaction is completed, to ensure that the Thymeleaf attribute "data-stock" is properly updated in the mainpage. The functions use AJAX so that the stock level display of items (in the modal) can be auto updated after every transaction without requiring a page reload. Also, the function checks if any items have run out of stock after the transaction and if so, it adds the out-of-stock class to the item and displays the overlay (the 'X') over it to show it's out of stock. This also happens without the need for a page reload, due to the use of AJAX. CSS styling for the out-of-stock items, also created a pseudo-element with an out-of-stock message which appears when hovering over an item. • Created UserControllerAdvice class to ensure all controller methods for most pages pass in the username attribute to the model so all the pages can display the user-info (username of the person currently logged in). If nobody is logged in, it overwrites the Spring Security standard "anonymousUser" with "Guest". Using a	

	controller advice allowed me to add the username to the model of the homepage, mainpage, about page, instructions page and receipt page without having to repeat code for each. Also added some more CSS styling improvements for the main page.	
03/03/25	• Expanded the product range which is inserted in data.sql file by adding 8 more items in (2 rows of 4). The existing scalable vending machine design in mainpage.html automatically adds in the new products to the item grid when the application is re-run and new product data is inserted into the database. • Added in new category attributes for all the items to group them together (each row of products belongs to a different category): modified the Product entity to include the new category attribute and inserted relevant categories for every item in data.sql. Also modified MainController to also pass in a list of all unique product categories in the database to the model of the mainpage. This is used in mainpage.html to display each category name above the relevant rows in the items grid view. Also had to change the HTML layout slightly to fit this in. • Added in category filtering functionality: created a dropdown element in mainpage.html positioned just above the vending machine. It allows the users to select which category they want to see in the display, also containing an option for all categories (the default view). Created an event listener function in JavaScript for this to work, which detects the selection made in the dropdown and shows/hides each product row based on this. • CSS styling: additions and changes to help implement the improvements to the UI.	**Additional Features/Improvements**
10/03/25 (Weekly Upload and Last Commit for the week were late on 18/03/25 due to technical issue)	• Edited the category filtering dropdown to now use a checkbox system to allow selecting multiple categories at once (previously only one at a time). Edited HTML page to have checkbox inputs and labels for each category. Modified the previous	**Additional Features/Improvements**

	JavaScript event listener to: show the dropdown when the button is clicked, hide the dropdown when outside of it is clicked, adds a listener to every checkbox element in order to detect changes (checking and unchecking), automatically unchecks every standalone category checkbox if “All Items is selected and displays all categories for this, automatically unchecks “All Items” if another category is selected but prevents just unchecking “All Items”, and automatically re-checks “All Items” and displays all categories if all other categories become unchecked to prevent an empty display. The event listener calls two new JavaScript functions to work, one which handles updating the display for specific categories and one to display all categories (for the All Items checkbox). • Keypad updates: expanded the keypad grid in mainpage.html to now include letters ‘E’ and ‘F’ to accommodate the new rows of items. Amended the relevant JavaScript functions so that the RegEx validation now allows letters ‘E’ and ‘F’ as acceptable keypad input. Also modified the event listener in mainpage_script to now allow keyboard shortcuts for letters A-F and numbers 1-4 (full range of the item keypad). The event listener checks that the keyboard input is allowed and then calls the existing updateDisplay function (which already handles all item code validation) and passes in the keyboard input. Also removed the DELETE key as a shortcut for the DEL button and now made it a shortcut for the CLR button, which was missing a shortcut before. Put all keyboard shortcut checking conditions into one big nested IF statement to prevent unnecessary extra checks for every keyboard event. • Updated the instructions panel to now inform about being able to now use the keyboard for typing in item codes and using the DELETE key as an alternative to the CLR button.	

	• Added in images for all new products to complete vending machine grid view. Scalable design auto added the images to the display and to the product modals when image files are added to the resources/static/images directory. • Added in new button to mainpage.html which adds all items to the cart to easily populate the cart with the maximum number of items, to help with development. Created new JavaScript function to implement this which iterates through every item and calls existing addItemTCart function to add them. Modified the addItemToCart function to receive another argument which tells it whether all items are being added or only one. This is so it can prevent added to cart alerts for when all items are being added, to make development easier and quicker. • Edited product modal to now also display the category of each product. • Implemented user transaction viewing logic. Added new User attribute to Transaction entity. Modified TransactionController and TransactionService methods to ensure the username of the currently logged-in user is properly saved into the database as the User attribute of the transaction, when it is created. Created new transactionspage.html which contains all the standard elements from the other pages and also a blank table template. Created transactionspage_script.js which contains an event listener and functions to retrieve a list of transactions made by the user currently logged in using AJAX, and dynamically populate the table with the transaction details. Created new controller method, getUserTransactions, in TransactionController to create the API endpoint, retrieve the username as a model attribute and return the list of transactions (by calling a new TransactionService method which uses a new TransactionRepo option to fetch the transactions from the database) to allow the frontend to display the transactions in the table.	

	• Enhanced transaction viewing by adding querying/filtering transactions functionality into the transactions page. Created new menu in the page containing HTML elements to be used as filters: an input box for Transaction ID, Date From and a Date To date entry boxes for the date range and min/max boxes for Total Cost, Payment Received and Change Given. All attributes in the transactions table have filters for customised querying. Menu also contains buttons to apply filters and reset filters. Created JavaScript method to apply filters which collects all the parameters that are used in the filters and sends them in an AJAX fetch request to an API endpoint created by a new controller method in TransactionController. It then calls the already existing populateTransactionTable function and passes in the response from the controller to display the filtered transactions in the table. There is also a JavaScript function to reset the filters, which clears all the filter HTML elements and redisplays the full transactions list for the user in the table. The new controller method extracts all the parameters sent by the frontend and passes them into a call to a new TransactionService method which uses them to call a new Repository method. The new TransactionRepo method contains a custom SQL query to fetch a list of all transactions that meet the conditions set by the parameters from the filters. It allows parameters (excluding username) to be null so any combination of filters will work. Username is required to ensure the user can only see their own transactions and nobody else's. The response is retrieved from the database and sent back through the repository, through the service method and through the controller method back to the frontend to be displayed in the table. The table can be updated dynamically every time new filters are applied and reset dynamically due to the use of AJAX which allows	

	JavaScript to display responses without requiring a full page reload. • Implemented remember-me authentication functionality to the login page using Spring Security: Added a checkbox to the loginpage.html for users to choose if they want to stay signed in. Modified the SecurityConfig file to implement the remember-me using a cookie rather than the database and set it to remember the user for 1 day (easily changeable later). • CSS stylings include: — Styled the cart to always be in view when the user scrolls down the page, for a better UI. — Created new stylesheet: transactionspage_styles.css to style the new transactions page. — Added in a border underneath each product row in the vending machine and reduced the space between the category headings. — Styling of the transactions table and filtering/querying menu — Styling the category filter menu in the main page. — Other minor stylings.	
17/03/25	• Implemented sorting by columns functionality into the transactions table. Added ids to each th element in the table and these get passed into a new JavaScript function, sortTableByColumn, which is called when a table header element is clicked. This function checks if the column is already the one that is sorted, if so, it will toggle a variable stating the direction of the sort (ascending or descending). If not, it will set the direction as ascending by default and store the column as the currently sorted one (in a global variable). The column and the direction are passed into another new function which does the actual sorting. It sorts through the full array of transactions by comparing through each value and re-ordering them when needed. The function	**Additional Features/Improvements**

	sortTableByColumn also calls another function, passing in the column and previous column (if one), which handles highlighting the sorted column and other visual effects for the sorting column. It adds a stylable class (for highlighting), an arrow displaying the direction (ascending or descending) and a relevant tooltip to the column heading. Before adding everything, it removes everything from the previous heading (if one) by calling another function, resetVisuals which undoes all the visual effects. Finally, there is an event listener to reset the sorting, by removing the visual effects (using resetVisuals on the currently sorted column) and re-sorting the data by transaction ids ascending (which is the default), when anything outside of the headers is double clicked. Added CSS styling for the highlighted headers, which only applies when the sorted-column class is added. Overall, this creates a handy, functional sorting system which is visually appealing and functions well. • Ensured filtering and sorting can be used together, sequentially without overriding each other. Initially, attempting to sort on filtered data would remove the filter(s) and bring back all transactions data to sort and attempting to filter on sorted data would override the sort and display the filtered data in default order (Transaction ID ascending). Modified applyFilters JavaScript function to store the filtered transactions in the global variable currentTransactions after they are fetched from the backend using AJAX. This is to ensure that any sorting after is performed on a current list of filtered transactions instead of just a full list of all transactions (the problem before). This allows sorting to happen on filtered data. Also added in a checking condition immediately after storing the filtered transactions in applyFilters, to see if there is a current sort active. If there is, the sortTableData function will be called to apply the sort to the filtered data first then	

	it will be added to the table, otherwise it will be added to the table straightaway. This allows filtering to happen on sorted data. Now sorting and filtering can both work seamlessly together in either order. • Added a JavaScript function to validate the filters, which is called in applyFilters to check the data before any filtering happens. It ensures the following conditions are met: — Date From is always before Date To — Transaction ID is a positive integer — All money values are positive The function contains individual checks for each of these different conditions. If any one of them fails the respective input fields in the HTML page will be cleared and the function will return False, stopping the filtering from happening. Also created an event listener which blocks invalid inputs from being inputted into the money fields, either by manually typing in or by using the arrow adjustors. Used RegEx validation in the oninput field for the Transaction ID input box to block the input if it is not a positive integer. Also set min values of 0 for the ID and money inputs for further validation. The date inputs are already quite robust as they are in date format and automatically do not allow invalid inputs. Lots of validation has been used (directly in the HTML elements and in the JavaScript function/event listener) because bad inputs can cause errors in the filtering logic code and because there are multiple ways to input data, so all grounds need to be covered. • Added in a small instructions panel for the transactions page, providing guidance and instructions for using the filtering and sorting features. • Receipt page improvements: Updated receipt format to display the username of the user at the top of the receipt, indicating who the transaction was made by. In case the user attribute for the transaction is missing or null it shows "Guest" as the user.	

	Made other minor changes to the receipt format. Added a back button to the receipt page which navigates back to the previous page (mainpage or transactionspage) or back to homepage if none (as a failsafe for now but should be one of the two mentioned in practice). Modified downloadReceipt JavaScript function to now save the downloadable receipt file as a pdf rather than a txt file, so it is not easily edited/overwritten. Also modified the whole logic inside the function to now capture an image of the receipt in the Receipt HTML page and embed this into the file rather than adding in all of the text like before. This is to ensure the exact format of the receipt from the page is copied into the downloaded file. It also now completely prevents editing the receipt content because it is essentially an image. • CSS stylings include: centered the filter menu in the page, added a '£' symbol to every money input box in the filter menu, more spacing, styled the new instructions panel, fixed some layout issues on the receipt page and transactions page, and other basic style improvements.	
24/03/25	• Created Admin Dashboard Page: New HTML page consistently designed like the other pages, with the webapp title, user info and logout button. Also contains a table to show each product, it's name, category and ID as well as its current stock level with some action buttons and finally settings for auto stock updating. Modified the security chain in the SecurityConfig class to use a new CustomLoginSuccessHandler class as the success handler. Now standard users can still log in to the homepage as before, while admins can log in and have access to the admin dashboard. Currently there is only one Admin login account which is inserted directly into the database from data.sql. it	**Additional Features/Improvements**

	has the role 'ADMIN' which grants access to the Admin Dashboard. Created new AdminController class with methods to render the page, fetch the products and apply the changes. Added in functionality using a new external JavaScript file. The table is dynamically populated upon the page load by a JS function which fetches the products from the backend, using AJAX and generates HTML rows for each to display on the page. Added in the functionality for admins to be able to manually change the stock of the products. The admin can use the action buttons to Increase, Decrease or Empty the stock. They can also use the Set Stock input box to set it at a specific value. This is displayed on the page next to the current stock as a "provisional" value, ready to be saved. Modified Product entity to now include 3 new attributes one for specifying if auto stock is enabled (boolean), one for specifying the stock threshold and another for the update value. These are read from the backend and displayed on the table. The Boolean value is visually represented by a toggle switch. Admins are free to turn the switch on/off. Having it on requires them to set a threshold and update value. This is for the auto update functionality in the backend: after a transaction, all purchased products are checked, and if their stock level has fallen below the threshold it will update their stock by their update value. All changes can be saved by clicking a global save button which submits the changes to the backend via a controller method API endpoint using AJAX. Added in basic validation for the values: stock cannot be set below 0, stock cannot be set below the threshold (if one). Also added in button validation, which dynamically enables/disabled the buttons where relevant, e.g. Decrease and Empty buttons are disabled when stock is 0, save button is disabled when no changes have been made. Each row contains an Undo button which	

	resets all the changes that have been made in the session. Additionally, there is a global Reset button which resets all changes in the session for the whole table. Finally, CSS styling for the page: layout, headings, text, buttons and table. • Altered project setup to be more secure: use an environment variable as a placeholder for the database password in application.properties file, so the password isn't visible anymore. • Overall: fully implemented admin stock management.	
31/03/25	• Touch ups for the admin dashboard page: Finalised the JavaScript code to ensure proper disabling of the buttons when necessary to ensure robust validation against bad inputs. CSS styling for the page to improve the layout and UI quality. • Fully implemented new Analytics page for users. Analytics page contains 3 sections: Summary, Purchase Frequency and Spending Trend. Summary section contains cards showing key summaries of the user's overall transactions history: total number of purchases, total money spent, most active day and number of unique items purchased. Created these cards as HTML elements and styled them using CSS to fit the UI theme. Second section contains a bar chart showing the number of purchases made in each month, and was created using the chart.js library. Third section contains a line graph showing how much the user has spent through each month, showing the increase and decrease between therefore portraying an overall trend. This was also created using chart.js. Both were styled within the JS code and externally through CSS to be interactive and also match the UI theme. JavaScript code for the sections consists of a fetch and render function for each. The render function is called by the fetch function and populates the cards with the passed in data/fetches and populates the charts with the passed in data. The fetch function for	**Additional Features/Improvements**

	each makes an AJAX GET request to an api from a controller method. The controller method calls a service method which calls a new repository method. The repository methods each consist of different specialised queries, created to fetch specific data from the database based on the analytics category. All transaction/product data is fetched only for the currently logged in user by their username. Created new DTO (Data Transfer Object) class for each section. The relevant queried data (for the charts) is returned to the service method as DTO objects for easy storing and transferring of data. For the summary cards, the service method receives all data from the repository and then creates the DTO object itself, before passing it back to the controller. The DTO objects are returned to the frontend as JSON responses using AJAX and the JS render methods handle the data and add it to the page dynamically. The Analytics page is accessible via a new button on the Home page, but this and now also the Transactions History button are disabled if a user isn't logged in (Guest). This validates against trying to access pages where information won't be available (because display data is fetched using the username). Also validated the cards to show default values in case no data is fetched and validated the charts to not be visible and instead display a message in case they are empty (no data). Overall created new HTML, JS and CSS files for the page as well as a new AnalyticsController class, AnalyticsService class and new query methods in both TransactionRepo and TransactionProductRepo. Page is complete. • Added in Smart Recommendations to the main page. These can be easily turned on/off using a toggle switch. When turned on, Recommended items will have a gold glow around them in the vending machine display. A JavaScript event listener detects when the switch is toggled, if turned off it calls a function to remove highlights and if turned on it calls a function to apply	

	highlights. The apply function makes an AJAX call to a new controller method which calls a new service method, which calls a new repository method to query the database for a list of recommended products (similar to how the analytics works but without the use of a DTO). For all the products in the list, it adds a class to the item elements of the products in the main page, which is specifically styled in CSS to highlight. • Fixed an error in the payment logic of the main page where payments were not being processed as complete when lots of the same coin was being used consecutively. This was down to a floating-point error causing the payment comparison to fail. Fixed this by adding 2 d.p. rounding to the insertCoin JavaScript function. • Added in a new field to the register page for confirming passwords. Modified the RegisterController to now check the passwords match before successfully registering the user, otherwise it displays a relevant message stating that the passwords must match. • Implemented OAuth2 authentication with Spring Security to allow logging in with Google. Added the dependency for OAuth2 to build.gradle file. Generated new Client ID and Client Secret keys directly from Google, stored them as environment variables and added them to the application.properties file along with other settings for the OAuth Google login. Modified SecurityConfig class to add in oauth2Login to the SecurityFilterChain, ensuring it uses the custom login page and CustomSuccessHandler class. Modified the CustomLoginSuccessHandler class to now handle OAuth2 logins: checks for the user in the database, by their email, if they already exist log them in otherwise register them and generate them a new username based on their display name and add on suffix numbers to the end if their display name is not unique. Used a while loop to keep	

	adding suffixes on until their provisional new username is unique. Modified the UserControllerAdvice class: removed specific class scoping as it has become apparent that by default all controllers receive the username model attribute. Also added in logic to check if the user has logged in using the form or by OAuth2 and based on this properly retrieves the username (added in fallbacks to use the email or a default username for OAuth2 logins in case of errors). Modified the AnalyticsController to contain a new helper method for fetching the username which handles fetching OAuth2 usernames as well. All the other controller methods now call this method to fetch the username rather than using @AuthenticationPrincipal UserDetails, which was causing the analytics page to crash due when OAuth2 users were trying to access it. Added in a new <a> link element to the login.html page with an embedded Google logo image inside it for users to access the Google OAuth2 login feature. Styled this using CSS in authpage_styles.css. • Modified payment modal in mainpage.html to now use proper coin images instead of standard coin buttons. This allows a unique and more realistic look to the coins. Also modified the CSS styles to make the coins bigger and be separated out into two rows of 3 to improve visibility and readability. Coins still function the same as before but now have a much-improved realistic look. • Added in coin animations to simulate the coins being inserted into the vending machine. New JavaScript function to clone the coin image and translate the clone towards the new visual coin slot at the top left of the machine. The translation is animated using a CSS style. Also, the vending machine is styled with temporary visual effects and a coin sound audio is played.	
07/04/25	• Working on Dissertation.	**6. Finalisation**

14/04/25	• Replaced all JavaScript alerts and confirm windows with a custom display modal to allow better styling and avoid basic JS popups. Created new JavaScript file: global_alert_modal.js which contains two functions, one to display the modal for a standard alert and one to display the modal for a confirmation. Calls are made to these methods in place of "alert" and "window.confirm" in JavaScript files and the same messages are passed into the new JS methods. Created the modal in a separate HTML file and added it in to the relevant pages as a Thymeleaf fragment to avoid repeated code. Styled the modal in generalstyles.css. • Modified JavaScript event listener for the Smart Recommendations toggle to disable the toggle switch when the user is not logged in to prevent redundant use of the switch, due to recommendations requiring a username to be generated. • Populated the aboutpage with general information about the app and styled it with a new stylesheet: aboutpage_styles.css. • Added in a new consent checkbox to the register page, ensuring users consent to collection of transaction data (required for transactions history and analytics pages) before they register. It is required for a user to register and access the full features of the system. • Also added in a “Login as Guest” button to the login page, which allows users to browse and simulate using the vending machine without having to log in, but they can’t use any other features. Added a small HTML banner to the top of the home page to show if the user is browsing as a guest. Modified the SecurityConfig file to allow guest users to skip the default authentication logic to access the home page. • Added a new visual to the analytics page, a pie chart showing the breakdown of purchased item quantities. Implemented this the same way as the other analytics charts: created new DTO class to store the	**Additional Features/Improvements** **6. Finalisation**

	data. Created new repository method to query the database to retrieve the name, id and sum of quantities for each product purchased by the user. Created new service method to call the repository method and store the returned data as the DTO. New controller method to call the service method and pass the data back to the frontend. New HTML analytics section created in the analytics page. New JavaScript load function calls the controller method via the API and new render function creates the pie chart using chart.js. All loading from backend happens via AJAX to allow dynamic rendering on to the page without requiring a full page reload. The pie chart uses unique colours for each different slice to differentiate between different products and has a key to help identify them. The pie chart is additionally styled through CSS. • Worked on Dissertation.	
21/04/25	• Improved Smart Recommendations functionality to give proper recommendations (before it was very basic off a one-line query). Created a new RecommendationService class containing a method which generates a list of smartly recommended products. The method uses an advanced hybrid algorithm which incorporates basic ML principles, taking into account user preference of product categories, average spending similarity and collaborative filtering based on purchases made by other users. Added an additional helper method in the service to calculate the average amount spent by a user, which is used for price similarity scoring. Added a new query method in TransactionRepo which retrieves the most popular products purchased by other users to be used for collaborative filtering. Modified the existing controller method for recommendations to now call this new service method, replacing the basic functionality of simply selecting top 5 products. This has made the Smart	**Additional Features/Improvements** **6. Finalisation**

	Recommendations feature more advanced, personalised and dynamic. • Further improved the Smart Recommendations algorithm by adapting the scoring system. Modified the algorithm to only apply price similarity scoring for users who have transactions (so new users with no purchases no longer get meaningless price-based recommendations, through comparing to their average spend of £0). Added a new helper method which dynamically adjusts the boost applied to collaborative filtering based on the user's transaction history, so collaborative recommendations are stronger for users with few purchases but weaker for users with more purchase data. Further modified the algorithm to calculate the contribution of category, price and collaborative filtering to each product's final score, and added a new RecommendationDTO class to store both the product ID and the dominant reason for its recommendation. Updated the HighlightRecommendedItems function in mainpage_script.js to display custom reason tooltips when hovering over smart recommended products based on their dominant reason. Styled the tooltips using CSS to match the system theme. Modified processTransaction() to automatically reset the Smart Recommendations toggle and remove highlights/tooltips after a purchase to properly reset the page. • Further fine-tuned the Smart Recommendations algorithm. Applied a log scale to the category score boosts to prevent runaway scores from heavily skewing recommendations towards certain categories. Added a multiplier to the price boost scoring to give price-based recommendations more impact. Tweaked the final collab boost values so collaborative filtering now balances properly with category and price scoring based on user data.	

	• Added a new visual insight feature to the Analytics page. Created a new section for Smart Recommendation Insight Cards styled with a gold theme matching the Smart Recommendations UI. Each card displays a key factor used in generating recommendations within the algorithm: top category, average spend and collaborative strength. Added a new service method in AnalyticsService to retrieve and calculate these insights, and a new controller method to pass the data to the frontend. Created a new JavaScript fetch method to retrieve the insights data via AJAX and a new render method to dynamically insert it into the page. The cards flip on click to reveal the top 3 products suggested for each insight factor (category, price, collab). Cards and tooltips were CSS styled consistently with the rest of the system. This new section helps users understand how their smart recommendations are generated, linking frontend visuals to backend analytics. • Added in simulated card payment functionality using Stripe Checkout. Inserted Stripe dependency into build.gradle file and added the test secret key into application.properties file. Created new PaymentController and StripeService classes. PaymentController contains a method to call the service to create Stripe Checkout sessions and also handles success/cancel redirects. StripeService securely fetches the user's email from the database, validates it, and defaults to a guest email if invalid. It uses this while creating the actual Stripe checkout session and feeding it back to the controller. Created a "Pay by Card" button inside the payment modal, only shown before any coins are inserted. Modified payment_script.js to add a new fetch method to create a Stripe session and redirect to Stripe's hosted payment page. Also now backs up cartItems	

	and payment amount into localStorage before redirecting to allow restoration after payment. After successful payment, restores cart and inserted amount, completes the transaction using the existing processTransaction() logic, and resets the UI. On payment cancellation, restores the cart and shows a cancellation alert. Added URL resetting after success/cancel to prevent retriggering of alerts on page refreshes. Pre-filled the user's email into the Stripe Checkout page where possible for a smoother experience. Stripe integration is fully simulated, secure and works seamlessly (in test mode) with the existing vending machine system alongside coin payments. • Worked on Dissertation.	
28/04/25 (Final Week)	• Created JUnit tests for ProductService and TransactionService classes. • Final system touch ups: re-enforced all Spring Security configurations. Secured page access through controller methods which now properly check if the user has logged in or used the guest button to prevent accessing pages directly from URL manipulation. Receipt viewing is now secured, ensuring users can only access receipts for transactions made by themselves. • Fixed Instructions Page: Created new updated user manual and fixed error from before where the iframe showing the user manual as a PDF viewer was being blocked by Spring Security. Added in JavaScript for full screen button. • Final Commit uploaded to GitLab: containing all final product and coin images, final touch ups, style improvements, cleaned up unused code. • Completed and Submitted Dissertation. • Project ready to be marked. • Preparation for Mini Viva.	**6. Finalisation**