Ho Chi Minh City University of Technology

Faculty of Computer Science and Engineering

Software Engineering (CO3001)

Student Smart Printing Service

Report 5 - Task 4

Authors:	*Student's ID:*
Nguyen Thanh Thao Nhi	2152840
Ta Ngoc Nam	2152788
Le Thanh Binh	2112897
Ly Tran Phuoc Tri	2153920
Phan Gia Bao	2153210
Hoang Tien Duc	2152520
Nguyen Huu Tho	2153843

Instructors:

PhD. Truong Thi Thai Minh

Completion date: 19th November, 2023

Mục lục

1 Requirement elicitation

1.1 Domain context

1.1.1 Business Domain: Education

The smart printing system is aimed at serving the higher education sector, specifically within the Ho Chi Minh City University of Technology (HCMUT). The primary goal is to facilitate an efficient, user-friendly, and manageable printing service for students across various campuses of the university. The system aims to streamline the printing process, making it more convenient for the student body.

1.1.2 Technical Domain: Web and Mobile Applications

Technically, the system will be a web-based application accessible from multiple platforms. It will integrate with HCMUT's Single Sign-On (SSO) for secure user authentication. Payments for additional printing pages will be processed through specific online payment gateways, such as BKPay. This ensures a seamless and secure experience for the users.

1.1.3 User Domain

The system caters to two main types of users. Students will be able to upload documents for printing, view their printing logs, and purchase additional printing pages. On the other hand, System and Printer Service Operators (SPSO) will have administrative capabilities, such as managing printers, configuring system settings, viewing logs, and generating as well as viewing monthly and yearly reports.

1.1.4 Functional Domain

From a functional perspective, printers are identified by multiple attributes including ID, brand, model, a short description, and location details like campus name, building, and room number. The system allows for various printing properties to be configured, such as paper size, pages to print, and the number of copies. Additionally, the system maintains detailed logs of printing activities, including relevant details like student ID, printer ID, file name, and printing times. The system also enforces account balance restrictions, providing a default number of A4-size pages each semester, with the option to buy additional pages. It adheres to legal and regulatory guidelines by complying with data protection laws and financial regulations related to online payment systems like BKPay.

1.2 Relevant Stakeholders and their needs

1.2.1 Student

The primary stakeholders are the students, who require a system that offers the convenience of printing documents easily from multiple locations within the campus. They also need

the capability to view their printing history and track the number of pages they have printed. When their default printing quota is reached, students should have the option to purchase additional pages. A user-friendly interface that is accessible through both web and mobile platforms is essential to meet their needs.

1.2.2 Student Printing Service Officers (SPSO)

The Student Printing Service Officers (SPSO) are responsible for the overall management and smooth running of the printing service. They need the ability to manage printers, including adding, enabling, and disabling them across various campuses. SPSOs also require access to detailed logs of all printing activities for monitoring and troubleshooting. Configuration options are important for them, such as setting permitted file types and defining the default number of printing pages for each student. Additionally, they must have the capability to generate and view monthly and yearly usage reports for administrative purposes.

1.2.3 University Administration

Lastly, the University Administration has a vested interest in providing a reliable and efficient printing service that meets the needs of the student body. For budgeting and planning, detailed reporting tools are essential. The administration also needs the system to comply with all relevant regulations and standards. Maintainability is a crucial factor; thus, the system should have minimal downtimes and come with clear documentation for troubleshooting and future updates. Integration capabilities with existing systems, such as HCMUT's Single Sign-On (SSO) for authentication, are also vital for seamless operation.

1.2.4 More relevant stakeholders

IT Support Staff are essential for technical upkeep and troubleshooting, while Faculty and Staff use the system for administrative and educational purposes. Local and Campus-based Vendors, who supply paper and ink, are concerned about inventory and logistics. External Auditors review the system for compliance with standards and regulations. The Financial Department is responsible for the billing aspects and budget allocation for the printing resources. Lastly, Unions or Student Associations may advocate for student needs, ensuring that the system is equitable and meets the demands of the student body. These diverse stakeholders each have unique needs and concerns that must be addressed for the smart printing system to be comprehensively effective.

1.3 Benefits of the HCMUT Student Smart Printing Service

The system has been designed to cater to the needs of stakeholders and provide them with numerous advantages. These benefits include improved efficiency and productivity, enhanced decision-making capabilities, streamlined processes and workflows, increased

accessibility to information, cost savings, etc. Consider each of the stakeholders listed below.

1. **Students**

   The HCMUT Student Smart Printing Service provides students with the convenience of easily accessing printing services from various campus locations. This system allows students to monitor their printing pages and purchase additional pages as needed, offering transparency regarding their printing activities. They can easily track the number of pages printed, their printing history, and their account balance, making it simpler to manage their resources effectively. Furthermore, the system ensures the security of their uploaded documents, safeguarding their privacy and academic work.

2. **Student Printing Service Officers (SPSO)**

   The SPSO (Student Printing Service Office) can experience increased administrative efficiency through the system's tools for managing student accounts, and printers, configuring system settings, and automating report generation. This has significantly reduced manual tasks for SPSO personnel. Moreover, the system ensures the security and integrity of printing data, thereby enhancing the overall reliability of the system and maintaining the trust of users. Furthermore, the system provides tools and access to system data, enabling effective support for students in resolving printing-related issues and addressing their queries. This comprehensive approach to managing the printing service benefits both SPSO and students alike.

3. **University Administration**

   The system significantly contributes to sustainability initiatives by reducing paper waste and promoting eco-friendly printing practices. This aligns seamlessly with the university's steadfast commitment to environmental responsibility. Additionally, the system assists in meeting any external regulatory requirements related to printing services, thereby helping the university maintain compliance with established standards. Moreover, the usage data and comprehensive reports generated by the system serve as invaluable tools for effective budget planning. These insights empower the University Administration to allocate resources judiciously and make informed financial decisions. Furthermore, the University Administration ensures that only authorized users can access the system through the HCMUT_SSO authentication service, emphasizing the importance of security and data protection.

1.4 Requirements

1.4.1 Functional requirements

1. **General requirements**

- Printer's information: Each printer contains printer ID, brand name, printer model, a short description and location (including campus name, building name, and room number).
- Printing log: The system has to log the printing history of students (including student ID, printer ID, file name, printing start and end time, and number of pages for each page size).
- Monthly and annual reports: The reports about the use of the printing service have to be generated at the end of each month and each year. These reports have to be stored in the system.
- Other services connection: The system has to be connected to some online payment system (like BKPay, MoMo, ZaloPay) and HCMUT_SSO authentication service.

2. **Students**

- File uploading: Students can upload the permitted document files for printing.
- Printer choosing: Students can choose a printer.
- Printing configuration: Students can specify printing properties such as paper size, number of pages to be printed, one or double-sided, and number of copies.
- Checking printing log: Students can view their printing history and the summary of the number of printed pages for each page size for a month.
- Checking payment log: Students can access a payment history log detailing their transactions related to purchasing additional printing pages.
- Checking printing balance: Students are given a default number of A4-size pages each semester for printing and they can view the number of remaining pages available for printing.
- Checking printing status: Students can check the status of their print jobs, including whether the job is queued, in progress, or completed.
- Buying pages: Students can buy more pages to print through BKPay, MoMo, and ZaloPay.

3. **Student Printing Service Officer (SPSO)**

- File type management: SPSO can limit and configure the file types which are allowed to be printed.
- Printer management: SPSO can add, enable and disable a printer.
- System configuration management: SPSO can manage other configurations of the system, including changing the default number of pages for printing each semester, changing the dates to give the default number of pages for printing to all students, and changing the valid file types allowed to be printed.

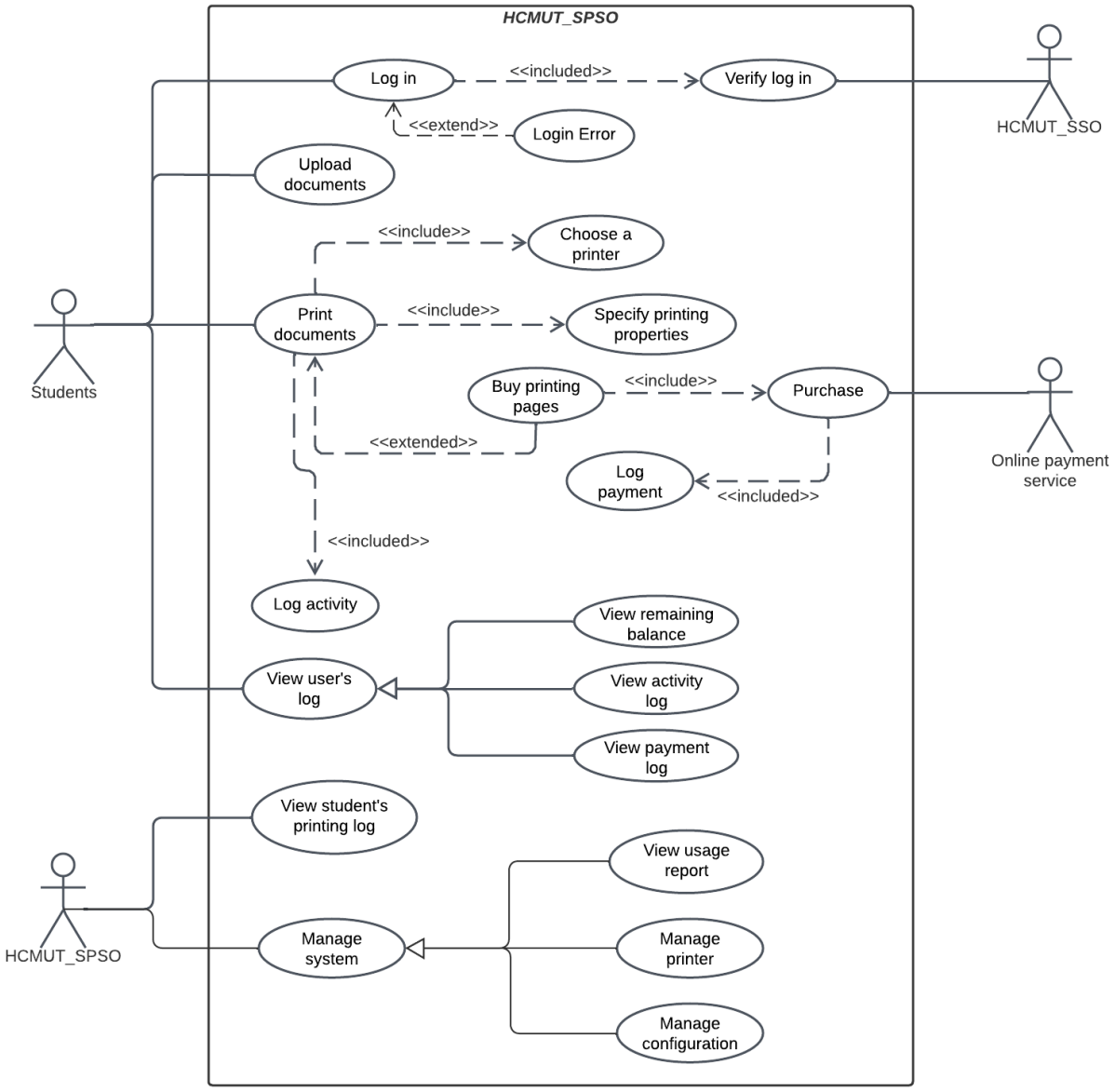
- Viewing reports: SPSO can view the monthly and annual reports of the use of the printing system anytime.
- Viewing printing log: SPSO can view the printing history of all students and all printers.

1.4.2 Non-Functional requirements

- Ease of use:
  + Students should be able to use the service after 10 minutes of training.
  + Administrative staff should be able to use all the functions of the system after 15 minutes of training.
- Performance:
  + Response time for user interactions (e.g., uploads, submissions, report generation) should be under 1 second when there is no network congestion.
  + Time to load the website should not exceed 2 seconds.
  + The system should be able to handle 150 concurrent users at one time.
- Capacity: Files uploaded at one time should be under 25 MB.
- Availability:
  + Users can access the web-based system 24/7.
  + The printing service should be available during office hours (6 A.M. to 6 P.M.) 7 days a week.
- Reliability: The percentage of failure that occurred should be under 5% in a month.
- Security:
  + Users can only log in to the system through one device at a time.
  + All users have to be authenticated by the HCMUT_SSO authentication service before using the system.
  + Payment gateway should be PCI DSS compliant.
- Scalability: The system should be scalable to adapt to the increasing number of users and printers during exam seasons.
- Compatibility: The website should be compatible with common web browsers such as Chrome, Safari, Edge, Opera, and Firefox.
- Portability: The website should be able to run on the new versions of the common web browsers (Chrome, Safari, Edge, Opera, Firefox) without changing behaviours and performance.

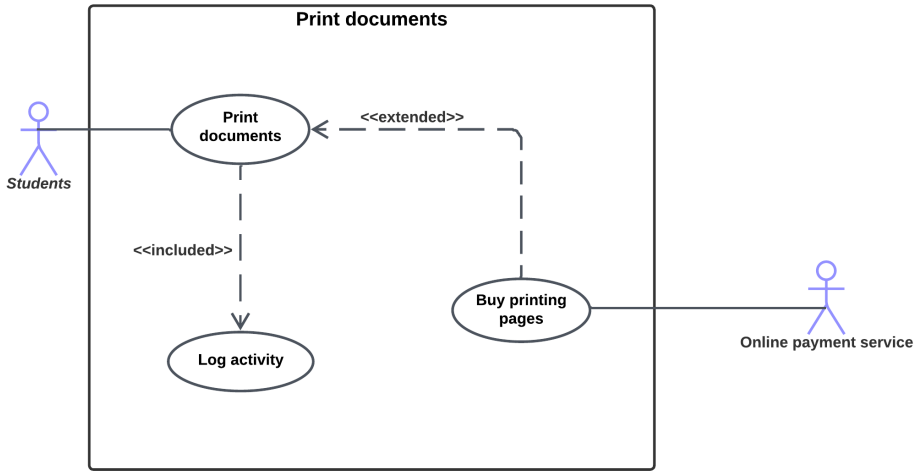
1.5 Use-case Diagram

1.5.1 For the whole system



Hình 1: The HCMUT_SSPS Use-case Diagram

1.5.2 For "Print Document" module



Hình 2: Printing Document

1. **Use case: Printing Document**

Use Case ID	01
Use Case Name	Print Document
Actor	Student, HCMUT SSO, Online payment service
Description	The process of a student printing a document with HCMUT_SSPS
Trigger	Click the ***In ngay*** button.
Preconditions	The student has logged into the system through HCMUT_SSO, has successfully uploaded one document file and has selected printer for printing.
Postconditions	The document is successfully printed according to the specified printing properties
Normal Flow	(a) Student chooses a printer and press ***Hoàn thành*** button. (b) The system accepts and displays the choosing properties session. (c) Students customize printer settings, including paper size, page selection, single or double-sided printing, and the number of copies. (d) Students press the ***In ngay*** button. (e) The system checks the student's remaining pages. (f) Printers start printing documents and the system shows the after-print page. (g) The system logs the printing activity.

Alternative Flows	*None*
Exceptions	*Exception 1: At step 6: The student's remaining pages is not enough* 6a. The system will ask the users to buy more pages, users will purchase for the buying page using an online payment service and the system will log this activity. *Use case continue from step 5.* 6b. If the users decide not to purchase additional pages, the system cancels the print job. *Use case end and return to the homepage.* 6c. If the payment transaction fails for any reason, the system provides options for the student to retry the payment or cancel the print job. *Use case end and return to the homepage.* *Exception 2: At step 7: Printer out of paper* 3d. The system displays a message and asks the user to choose another printer *Use case continue from step 1.*
Notes and Issues	If, during the printing process, technical issues occur such as paper jams, printer errors, or network problems: • Instead of manually selecting a printer, the system can automatically select the default printer and printer settings. • The system detects the issue and displays an error message. • The system may prompt the student to select a different printer if available. • The student can choose to retry the print job or cancel it. • If the student retries and the issue is resolved, the printing process continues.

2. **Use case: Buy printing pages**

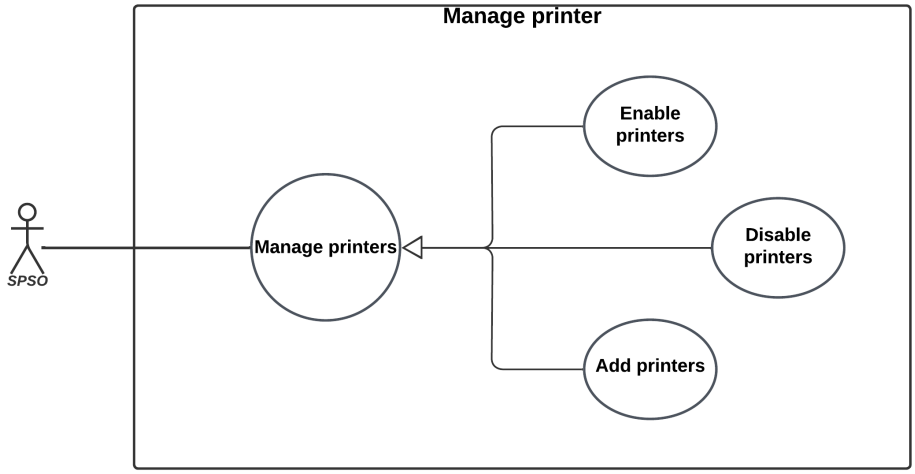
Use Case ID	02
Use Case Name	Buy printing pages
Actor	Student, Online payment service
Description	Student can buy more pages for their printing activities

Trigger	Student clicks the ***In ngay***button or students click on ***Mua thêm giấy*** button
Preconditions	After choosing printing properties, student clicks ***In ngay*** button but the remaining pages are insufficient, or student intends and navigate to account page to purchase additional pages.
Postconditions	The student's new remaining pages are updated
Normal Flow	(a) The system displays a window containing information on the price per page, the quantity of pages needed, and issues relevant warnings. (b) The student enters the number of pages. (c) The system automatically adjusts the total price. (d) The student choose ***Thanh toán*** button to make a transaction with Online payment service.
Alternative Flows	None
Exceptions	*Exception 1: At step 1* 1a. If the student click the ***Hủy giao dịch*** button instead of the ***Mua thêm trang*** button, go back to the printing window. *Use case end.*

3. **Use Case: Log activity**

Use Case ID	03
Use Case Name	Log activity
Actor	HCMUT SSO
Description	When the document is printed successfully, the system logs this activity.
Trigger	The printing process is successfully finished.
Preconditions	The document is successfully printed.
Postconditions	The system successfully logs the current printing activities
Normal Flow	(a) After successfully printing the document, the system logs that printing activity.
Alternative Flows	None
Exceptions	None

1.5.3 For "Manage System" module



Hình 3: Manage system

1. **Use case: Manage Printers**

Use Case ID	04
Use Case Name	Manage Printers
Actor	SPSO administrators
Description	The SPSO administrators can perform some operations on printers such as add, enable, disable
Trigger	Administrators click the ***Quản lí hệ thống*** button
Preconditions	Administrators have successfully logged into the system through HCMUT_SSO, a particular homepage appeared for admin role.
Postconditions	Navigate to the particular administrator's dashboard for managing printers.
Normal Flow	(a) The system displays admin dashboard. (b) Admins view printing activities of all students, list of all printer's information, modify printer's status or the time that reports are automatically generated or configure them.
Alternative Flows	None
Exceptions	None

2. **Use case: Enable Printer**

Use Case ID	05
Use Case Name	Enable Printer
Actor	SPSO administrator
Description	SPSO administrator can enable a printer
Trigger	SPSO administrator clicks the ***Khởi động*** button on the ***Tùy chọn*** column at the printer which needs to be enabled
Preconditions	Admins are in the management page, which has ***Quản lý hệ thống - Máy in*** section
Postconditions	The chosen printer will be enabled
Normal Flow	(a) The system changes the status of the printer into ***Đang hoạt động*** and saves that status.
Alternative Flows	None
Exceptions	No

3. **Use case: Disable Printers**

Use Case ID	06
Use Case Name	Disable Printers
Actor	SPSO administrator
Description	SPSO administrator can disable a printer
Trigger	SPSO administrator clicks the ***Buộc dừng*** button on the ***Tùy chọn*** column at the printer which needs to be disabled
Preconditions	Admins are in the management page, which has ***Quản lý hệ thống - Máy in*** section
Postconditions	The chosen printer will be disabled
Normal Flow	(a) The system changes the status of the printer into ***Ngưng hoạt động*** and saves that status.
Alternative Flows	None
Exceptions	No

4. **Use case: Add Printers**

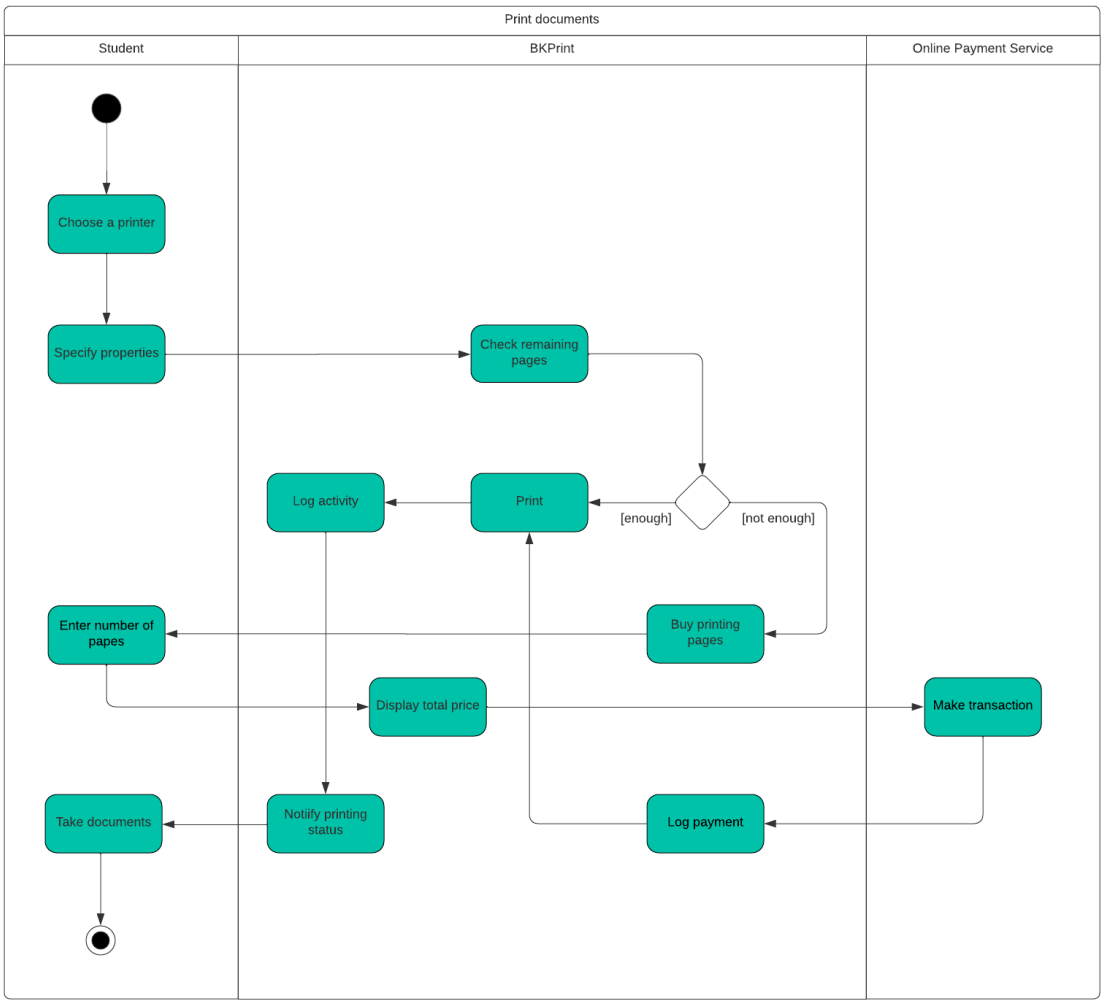
Use Case ID	07
Use Case Name	Add Printers
Actor	SPSO administrators

Description	The SPSO administrators add new printers to the system
Trigger	Administrators click the ***Thêm máy in*** button
Preconditions	Administrators have navigated to the management dashboard.
Postconditions	The newly added printer is showed in the printer list.
Normal Flow	(a) The system displays a blank area. (b) Administrators fill in all information about the new printers (ID, brand, position, state) (c) Admins click ***Thêm máy in*** button.
Alternative Flows	None
Exce ptions	*Exception 1: At step 1* 1a. If the SPSO administrators add a printer that already exists, the system will generate a notification indicating that this printer has already existed. *Use case continue at step 1.*

2 System Modelling

2.1 Activity Diagram

2.1.1 "Print Document" module



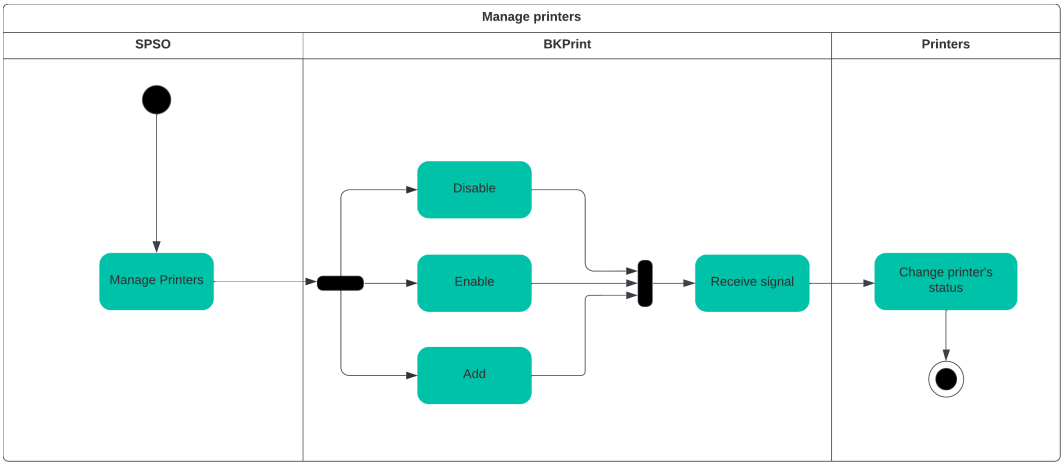
Hình 4: The *Print Document* Activity Diagram

The diagram illustrates the document printing process for students using BKPrint, an online printing service, involving three key swimlanes: **Student, BKPrint,** and **Online Payment Service**. **Students** begin by selecting a printer from various building locations and specifying their printing preferences, such as paper size, the number of copies, and single-sided or double-sided printing. If the student's page balance is sufficient, **BKPrint** proceeds with the document printing. In cases of insufficient pages, **BKPrint** prompts the student to purchase the necessary pages, with the student entering the page count. **BKPrint** calculates and displays the total price that student need to pay, then sends a request to the **Online Payment Service** to make transaction. Upon a successful transaction, payment details are logged, and the document is printed according to the student's specifications. This streamlined process ensures efficient and convenient printing for students, optimizing resource management and payment handling.
From the **Print** step in the diagram, there are two consecutive directions:

1. Activity is logged: **BKPrint** logs the printing activity so that the student can track their printing usage.

2. Printing status is notified: **BKPrint** notifies the student about the printing status, such as when the document is printing, finished printing, or cancelled.

2.1.2 "Manage Printer" module

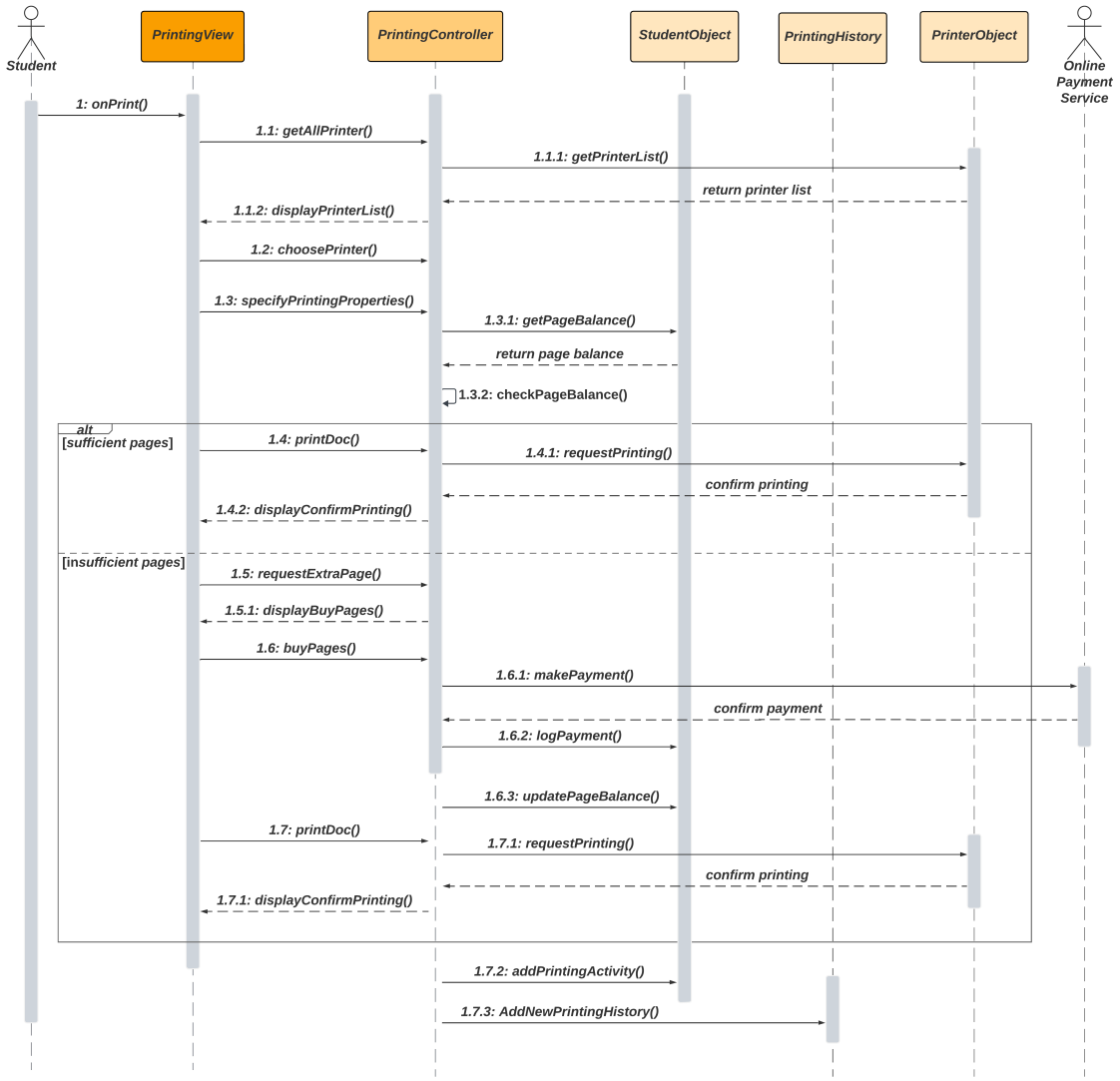


Hình 5: The *Manage Printer* Activity Diagram

In the system represented by the diagram, three key swimlanes play distinct roles in the management of printers: **SPSO** (System Print Service Operator), **BKPrint** (Printer Control Interface), and **Printers**. **SPSO** serves as the primary decision-maker, responsible for initiating printer management tasks. These tasks include three essential options: Disable, Enable, and Add. When **SPSO** decides to take action, it sends signals to **BKPrint**, which acts as an intermediary between the operator and the fleet of printers. These instructions pertain to changing the status of printers, such as disabling one to halt printing activities, enabling another for use, or adding a new printer to the system.

2.2 Sequence Diagram

2.2.1 "Printing Document" module



Hình 6: The *Printing Document* Sequence Diagram

The depicted sequence diagram involves two primary actors: **Student, Online Payment Service** , along with five key objects: **PrintingView, PrintingController, StudentObject, PrintingHistory** and **PrinterObject**. When the **Student** calls the **onPrint()**, the **PrintingView** will call method **getAllPrinter** to request the **Printing Controller** retrieve the available printers, by calling **getPrinterList** from **PrinterObject**. The **PrinterObject** will return the printer list to the **PrintingController**, and **PrintingController** will ask **PrintingView** to show the list of printers to **Student** through method **displayPrinterList()**. After that, the **PrintingView** calls **choosePrinter()** from **PrintingController** to suggest students to select their preferred printer and then calls **specifyPrintingProperties()** to elicit user input on printing properties such as paper size and number of copies.

The **PrintingController** assumes a vital role in the **printDoc()** method, commencing with a call to **getPageBalance()** from **StudentObject**. Following the retrieval of page balance information, the **PrintingController** proceeds to **checkPageBalance()**. This

critical method then leads to one of two possible scenarios:

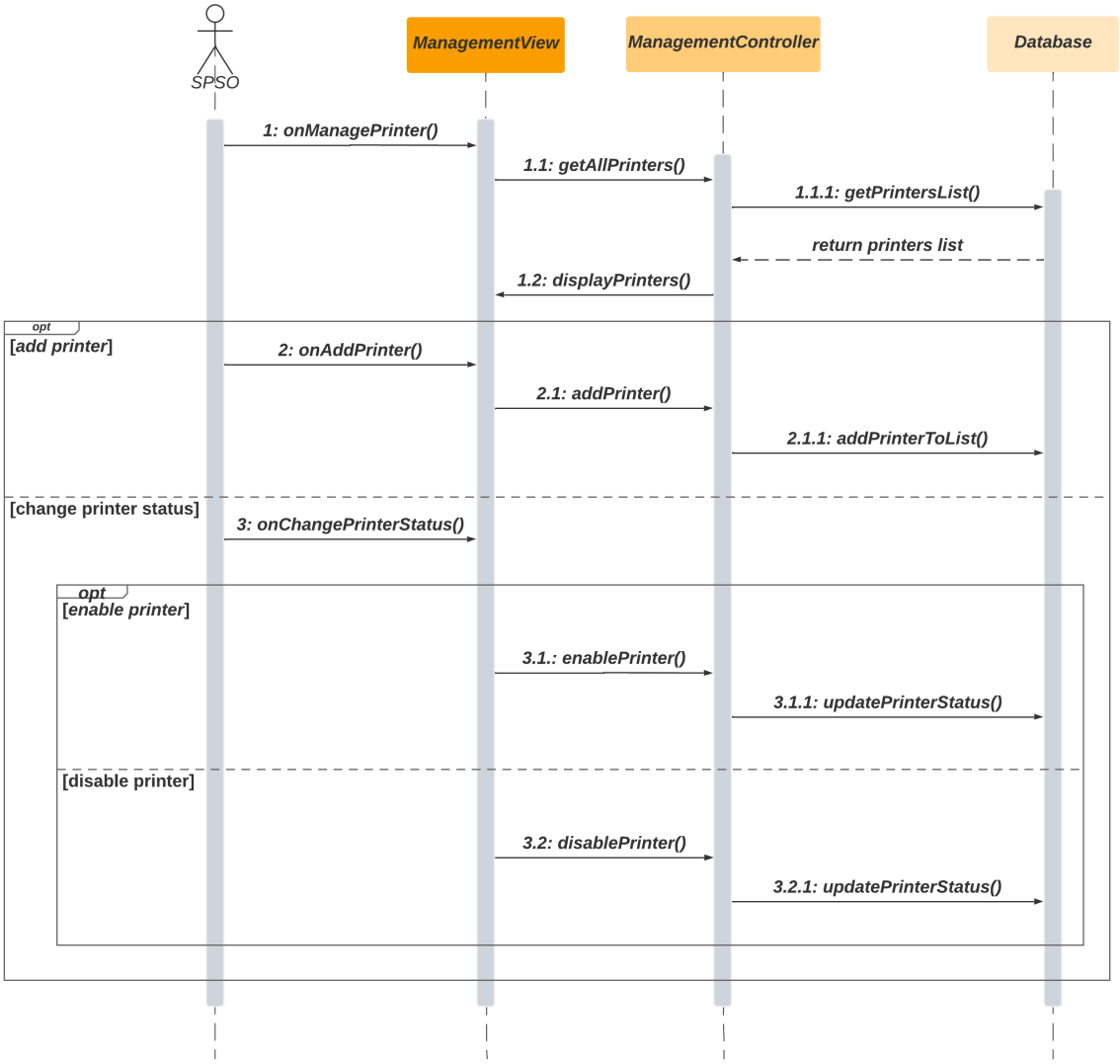
1. If the available page count is sufficient, after **Student** clicks the confirm button in the **PrintingView**, **PrintingView** will calls **printDoc()** method from **PrintingController** to perform the printing procedure. The **PrintingController** invokes **requestPrinting()** on **PrinterObject** to ask the chosen printer print the document file. Subsequently, **displayConfirmPrinting()** in **PrintingView** is triggered by the **PrintingController** before concluding the process.

2. In cases where the page count is insufficient, after **Student** clicks the confirm button in the **PrintingView**, **PrintingView** will call **requestExtraPage()** in **PrintingController**, then **displayBuyPages()** in **PrintingView** is triggered by **PrintingController**, prompting the student to purchase more pages. This involves a call to **buyPages()** in **PrintingController** by **PrintingView**, triggering a series of synchronized methods.

    - The **PrintingController** calls **makePayments()** method in the Online Payment Service, then receiving a confirmation message to validate the transaction.
    - Following payment confirmation, **logPayment()** is called by the **PrintingController** on **StudentObject** to record the transaction in the database.
    - With the successful storage of the transaction, **updatePageBalance()** is invoked by the **PrintingController** on **StudentObject** to update the new page balance.
    - After purchasing additional pages and updating the database, **PrintingView** reinvokes **printDoc()** to **requestPrinting()** on **PrinterObject**. It is noteworthy that the subsequent flow mirrors that of the scenario with the chosen printer having sufficient pages.

Finally, The **printDoc()** method concludes as the **PrintingController** calls **addPrintingActivity()** on **StudentObject** and **addNewPrintingHistory()** on **PrintingHistory** to store the printing activity details of that student and the chosen printer.

2.2.2 "Manage Printer" module



Hình 7: The *Manage Printer* Sequence Diagram

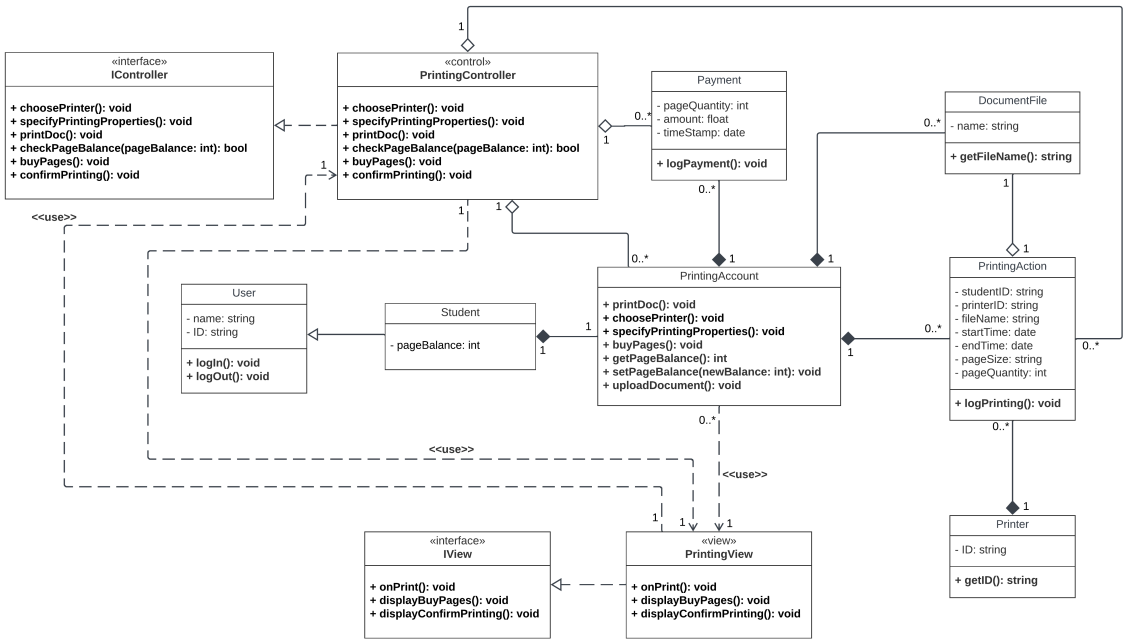
The above sequence diagram contains **SPSO** as the only actor and 3 objects: **ManagementView, ManagementController, Database**. When **SPSO** calls the **onManagePrinter()**, object **ManagementView** calls the **getAllPrinter()**, object **ManagementController** calls the **getPrintersList()** and request the database return all the printer's information. After that, **ManagementController** returns printer's information to the **ManagementView** through the method **displayPrinters()**.
Beside checking list of printers, the **Manage Printer** module also contains 2 more optional method.

1. When **SPSO** calls **onAddPrinter()**, **ManagementView** calls the **addPrinter()** to the **ManagementController**, and the **ManagementController** calls the **addPrinterToList()** to the database and the database will create and add a new printer item into the list.

2. When **SPSO** calls **onChangePrinterStatus()** to the **ManagementView**. The **ManagementView** can optional call **DisablePrinter()** or **EnablePrinter()** to the **ManagementController** to enable or disable a specific printer. In order to perform the method **EnablePrinter()** or **DisablePrinter()**, the **ManagementController** calls the **updatePrinterStatus()** to the database and the database will update the status of the chosen printer.

2.3 Class Diagram

2.3.1 "Printing Document" module



Hình 8: The *Print Document* Class Diagram

The above class diagram shows components in the system based on Model-View-Controller (MVC) pattern when a student prints document. The module includes classes **IController** and **IView** as interfaces; class **PrintingController** as controller and class **PrintingView** as view; classes **User**, **Student**, **PrintingAccount**, **Payment**, **DocumentFile**, **PrintingAction**, **Printer** as data model.

The relationships and multiplicity are also contained in the class diagram:

1. Inheritance relationship:

   - The class **Student** is a sub-class of the class **User**. **Student** inherits all attributes and methods from **User** while having its own attribute, which is **pageBalance**.

2. Realization relationship:

- The class **PrintingController** provides concrete implementation for the methods defined in the class **IController**.
- The class **PrintingView** provides concrete implementation for the methods defined in the class **IView**.

3. Aggregation relationship:

   - The class **PrintingController** has aggregation relationship with the class **PrintingAction**, which means **PrintingController** has many **PrintingAction** to control, and when **PrintingController** is destroyed, **PrintingAction** still exists. A **PrintingController** can have zero or more **PrintingAction**, while a **PrintingAction** can belong to 1 and only 1 **PrintingController**.
   - The class **PrintingController** has aggregation relationship with the class **Payment**, which means **PrintingController** has many **Payment** to control, and when **PrintingController** is destroyed, **Payment** still exists. A **PrintingController** can have zero or more **Payment**, while a **Payment** can belong to 1 and only 1 **PrintingController**.
   - The class **PrintingAccount** has aggregation relationship with the class **PrintingController**, which means **PrintingController** has many **PrintingAccount** to control, and when **PrintingController** is destroyed, **PrintingAccount** still exists. A **PrintingController** can have zero or more **PrintingAccount**, while a **PrintingAccount** can belong to 1 and only 1 **PrintingController**.
   - The class **DocumentFile** has aggregation relationship with the class **PrintingAction**, which means **PrintingAction** has **DocumentFile** and when **PrintingAction** is destroyed, **DocumentFile** still exists. A **PrintingAction** can have 1 and only 1 **DocumentFile**, and a **DocumentFile** can belong to 1 and only 1 **PrintingAction**

4. Composition relationship:

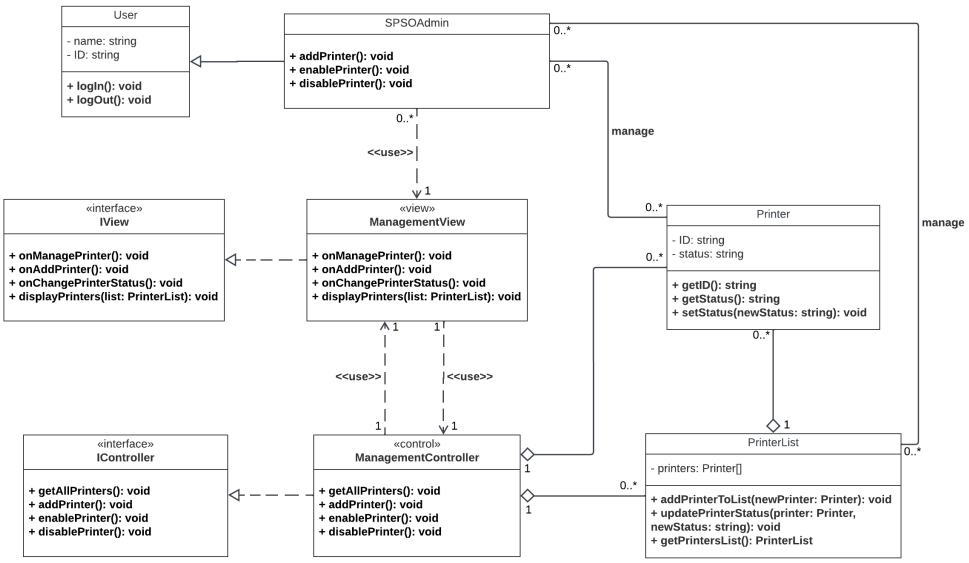
   - The class **PrintingAccount** has composition relationship with the class **Student**, which means **Student** has only one **PrintingAccount**, but when **Student** is destroyed, **PrintingAccount** is also destroyed. A **Student** can have only one **PrintingAccount**, and a **PrintingAccount** can belong to 1 and only 1 **Student**.
   - The class **PrintingAccount** has composition relationship with the class **DocumentFile**, which means **PrintingAccount** has many **DocumentFile**, but when **PrintingAccount** is destroyed, **DocumentFile** is also destroyed. A **PrintingAccount** can have zero or more **DocumentFile**, but a **DocumentFile** can belong to 1 and only 1 **PrintingAccount**.

- The class **PrintingAccount** has composition relationship with the class **PrintingAction**, which means **PrintingAccount** has many **PrintingAction**, but when **PrintingAccount** is destroyed, **PrintingAction** is also destroyed. A **PrintingAccount** can have zero or more **PrintingAction**, but a **PrintingAction** can belong to 1 and only 1 **Student**.
- The class **PrintingAccount** has composition relationship with the class **Payment**, which means **PrintingAccount** has many **Payment**, but when **PrintingAccount** is destroyed, **Payment** is also destroyed. A **PrintingAccount** can have zero or more **Payment**, but a **Payment** can belong to 1 and only 1 **PrintingAccount**.
- The class **Printer** has composition relationship with the class **PrintingAction**, which means **Printer** has many **PrintingAction**, but when **Printer** is destroyed, **PrintingAction** is also destroyed. A **Printer** can have zero or more **PrintingAction**, but a **PrintingAction** can belong to 1 and only 1 **Printer**.

5. Dependency relationship:

- The class **PrintingView** has dependency relationship with the class **PrintingController**, which means **PrintingView** uses methods in **PrintingController**. A **PrintingView** uses methods in 1 **PrintingController**, and a **PrintingController** is used by 1 **PrintingView**.
- The class **PrintingController** has dependency relationship with the class **PrintingView**, which means **PrintingController** uses methods in **PrintingView**. A **PrintingController** uses methods in 1 **PrintingView**, and a **PrintingView** is used by 1 **PrintingController**.
- The class **PrintingAccount** has dependency relationship with the class **PrintingView**, which means **PrintingAccount** uses methods in **PrintingView**. A **PrintingAccount** uses methods in 1 **PrintingView**, and a **PrintingView** is used by zero or more **PrintingAccount**.

2.3.2 "Manage Printer" module



Hình 9: The *Manage Printer* Class Diagram

The above class diagram illustrates components in the system and theirs relationships in Model-View-Controller (MVC) pattern where a user, specific SPSO admin manage the printers. The **User, SPSOAdmin, PrinterList** and **Printer** class are model. The view module includes **ManagementView** class and **IView** is its interface. The controller module is the **ManagementController** class, using to control the whole system. Its interface called **IController**.

In addition, the relationships depicted can be clearly seen in the provided class diagram:

1. Inheritance relationship:
    - The class **SPSOAdmin** inherits from the class User. This relationship signifies that a SPSOAdmin is a specialized type of User, inheriting all the attributes such as name and ID and methods of the User class including login() and logOut().
2. Realization relationship:
    - The class **ManagementView** implements the interface **IView**, indicating that **ManagementView** provides concrete implementations for all the methods defined in the **IView** interface.
    - With the implementation of the **IController** interface, the **ManagementController** class provides specific realizations for all the methods defined within the interface.
3. Aggregation relationship:

- The class **PrinterList** has an aggregation relationship with the class **Printer**. The **PrinterList** class manages a collection of Printer objects, enabling operations related to organizing and manipulating multiple instances of the Printer class. The multiplicity between the **PrinterList** class and the Printer class is one-to-many, meaning that a single **PrinterList** instance can manage multiple instances of the **Printer** class.
- The class **ManagementController** has an aggregation relationship wit with the **PrinterList** class, signifying that the **ManagementController** class manages instances of the **PrinterList** class. the multiplicity is one-to-many, indicating that a single **ManagementController** instance manages many **PrinterList** instances.
- **ManagementController** class has an aggregation relationship with the class Printer, so it can perform various functions related to managing and coordinating the behaviour of the Printer instances within the system. The multiplicity between them is one-to-many, suggesting that a single **ManagementController** instance can control multiple Printer instances within the system.

4. Dependency relationship:

- The **SPSOAdmin** class has a dependency relationship with the **ManagementView** class. **SPSOAdmin** class *use* the **ManagementView** class. The multiplicity between the **SPSOAdmin** class and the **ManagementView** class is denoted as many to 1, each instance of **SPSOAdmin** is associated with exactly one instance of **ManagementView**, and each instance of **ManagementView** is associated with zero or more instance of **SPSOAdmin**.
- The **ManagementView** class depends on the functionalities provided by the **ManagementController** class to handle and manage various control operations within the system's view. The **ManagementView** class *use* the **ManagementController** class. The multiplicity between the **ManagementView** class and the **ManagementController** class is denoted as 1 to 1, indicating a one-to-one relationship, where each instance of the view relies on exactly one instance of the controller to handle its functionalities and control operations within the system's view.
- The **ManagementController** class depends on the functionalities provided by the **ManagementView** class to interact with and manage the view components and related operations within the system. The **ManagementController** class *use* the **ManagementView** class. The multiplicity is typically denoted as 1 to 1, implying a one-to-one relationship.

5. Association relationship:

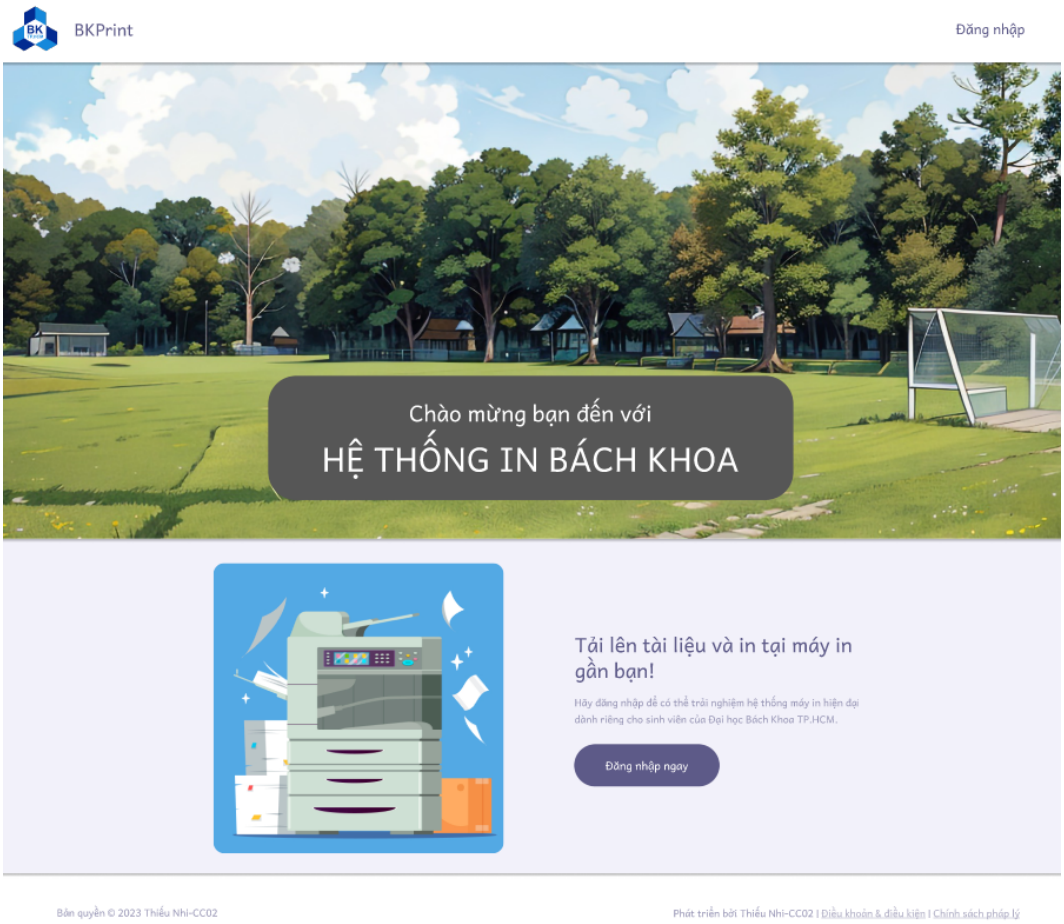
- The association relationship between the **SPSOAdmin** class and **PrinterList** class allows the **SPSOAdmin** to manage and control the status of a list of

printers, as well as add new printers to the existing list and update their statuses (enable or disable) as required. The multiplicity between them is many-to-many, allowing multiple instances of the **PrinterList** class to be managed by a single instance of the **SPSOAdmin** class and conversely.

- The **SPSOAdmin** class is also associated with the Printer class, enabling the **SPSOAdmin** to manage and perform various operations related to the individual printers such as enable, disable, or add new printers to the system as needed as well as access printer-specific functionalities such as printer ID and status. This is a many-to-many relationship, indicating that multiple instances of the **SPSOAdmin** class can be associated with multiple instances of the **Printer** class and ver vice.

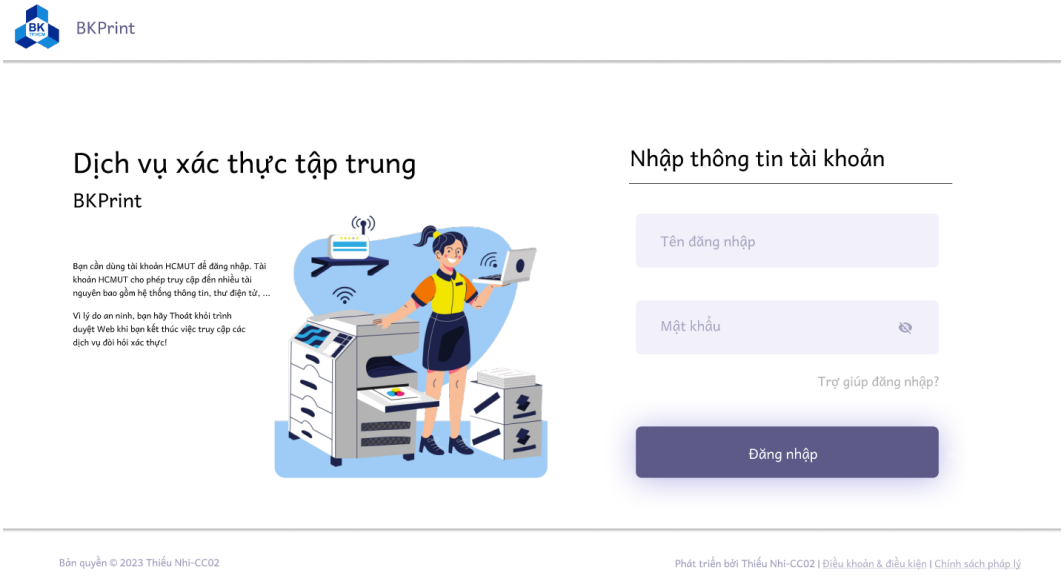
2.4 User Interface

Based on the above diagrams, the UI interface was designed to visualize the app. This section generally explains the UI design. For further experiment, please check the link which reference to the Figma design.
Overall, the design contains 24 frames which show the interface of the BKPrint app.

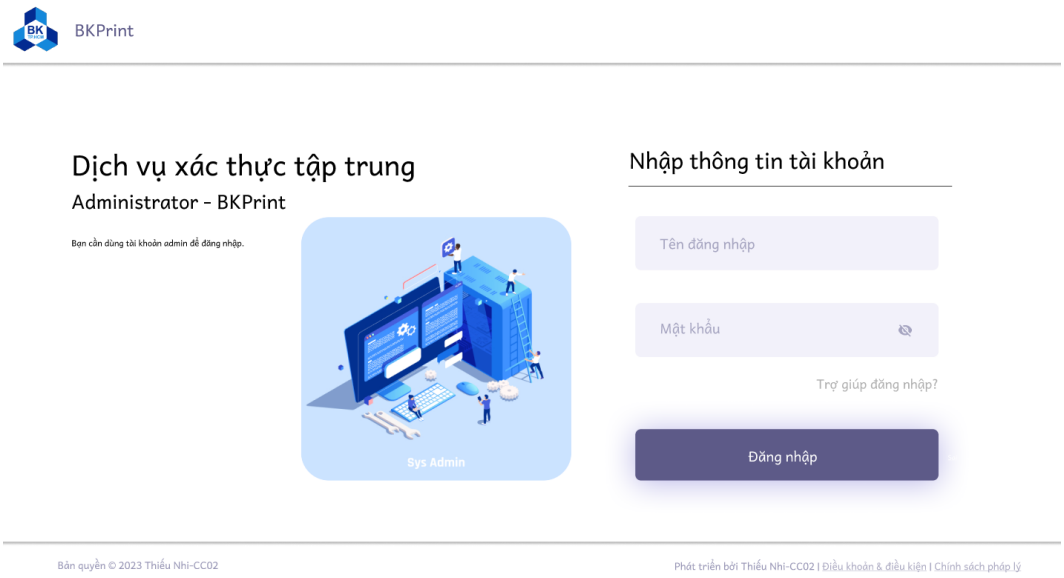


Hình 10: This is the Homepage where the user has not signed in yet.

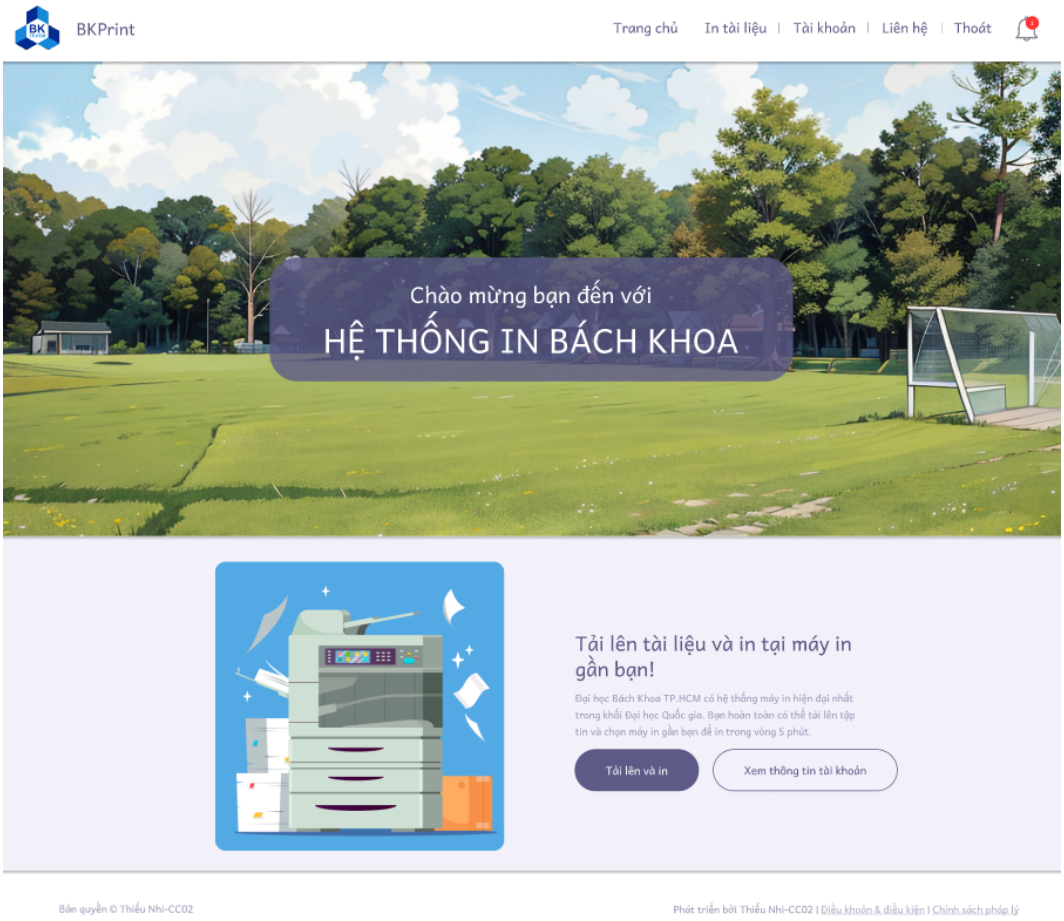
Hình 11: After click "Đăng nhập", user has to choose whether they are students or admins



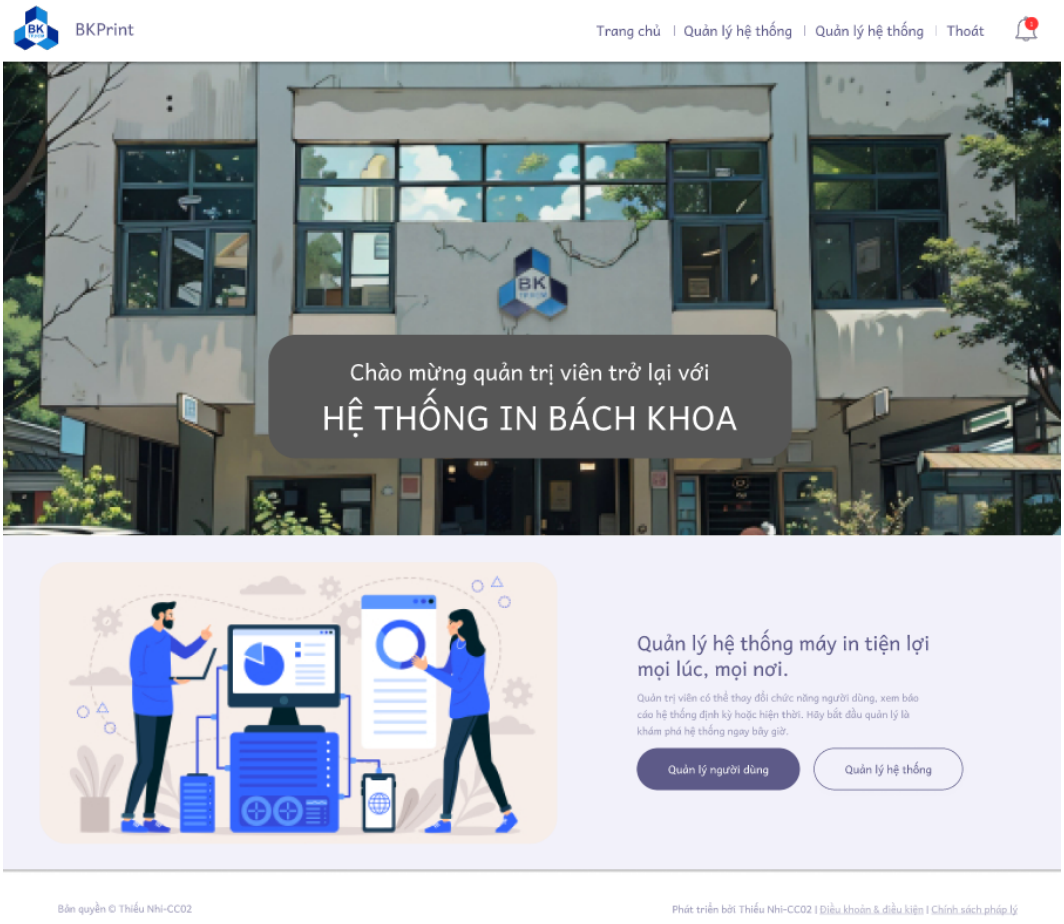
Hình 12: Student log in page



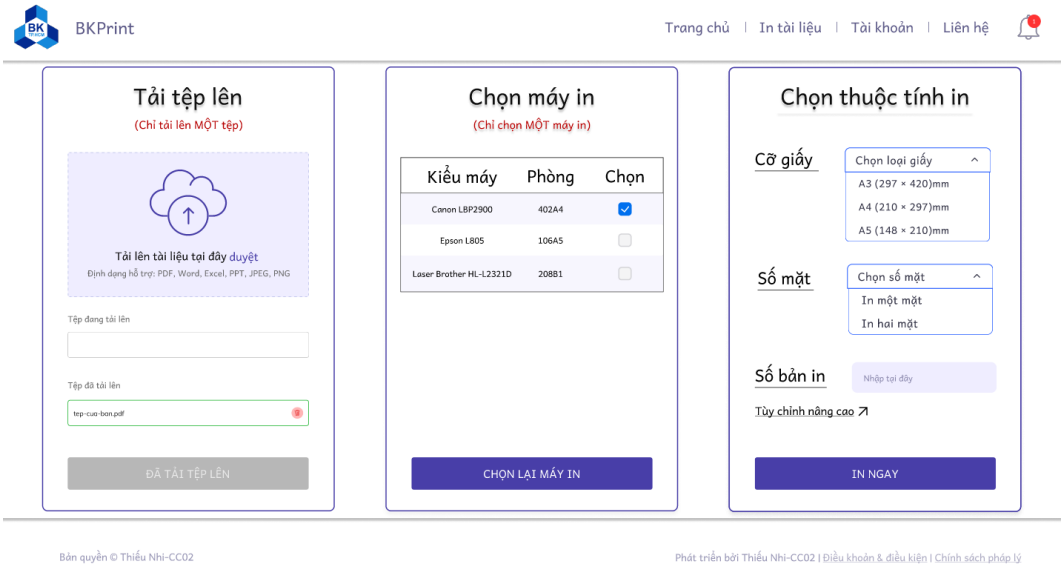
Hình 13: Admin log in page



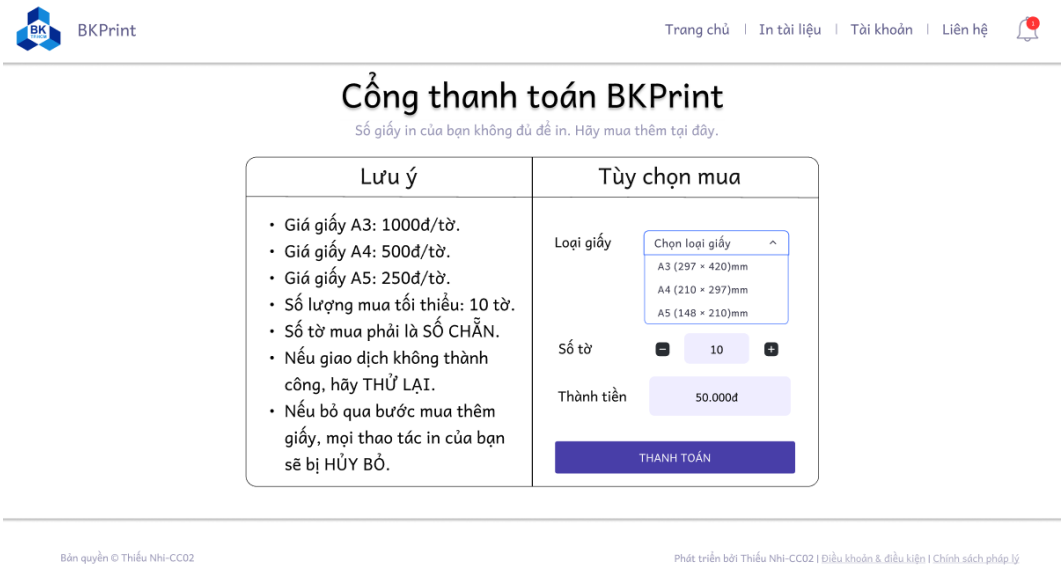
Hình 14: Student homepage after log in



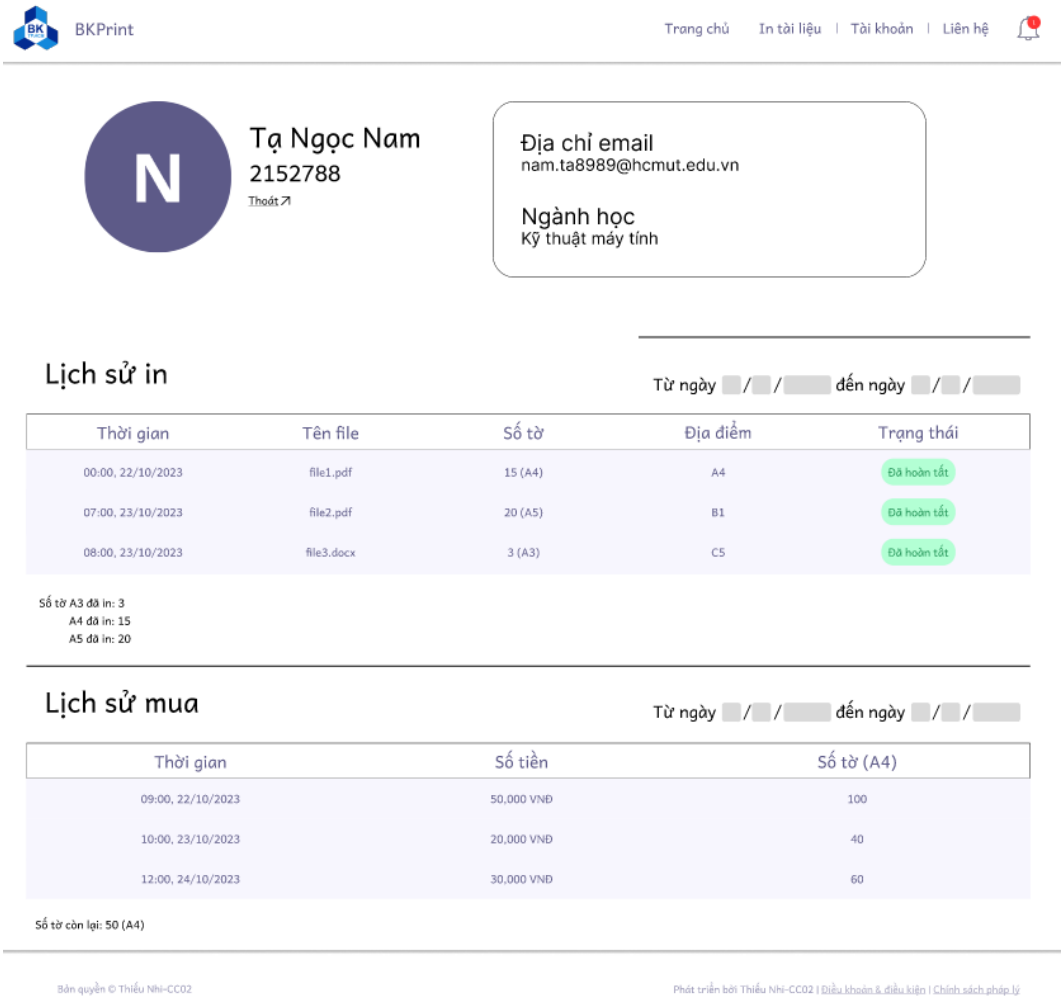
Hình 15: Admin homepage after log in



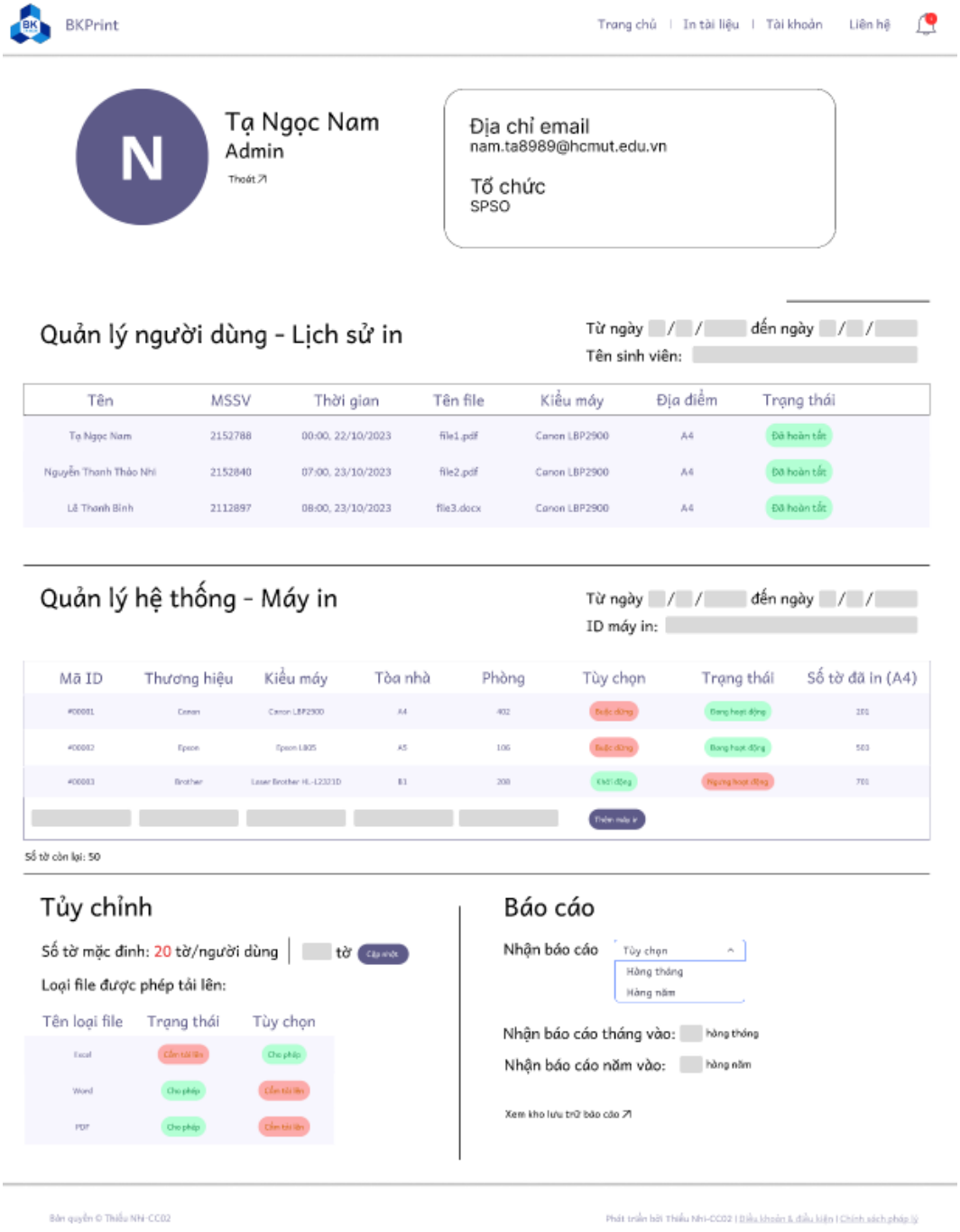
Hình 16: These are three phases of a printing process. Students are suposed to implement step by step.



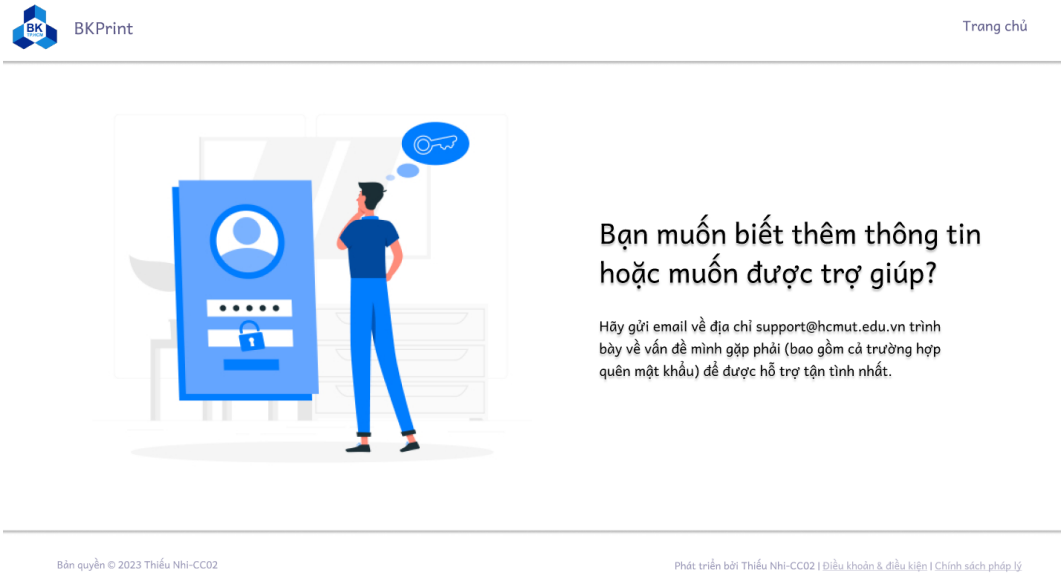
Hình 17: If the balance of the user is not efficient (not enough paper), the interface will redirect to this "buying page".



Hình 18: Student account page



Hình 19: Admin account page



Hình 20: If any error happens (such as user forgot their password, ...), they will be redirected to this page

As mentioned above, these images are just the general explanation. Please take a look at our UI design for better insight. Full UI wireframe design can be view at the link: CC02_Thiếu Nhi_Wireframe. The web demo of website can be viewed by clicking the "Present" button or pressing "Shift + Space".

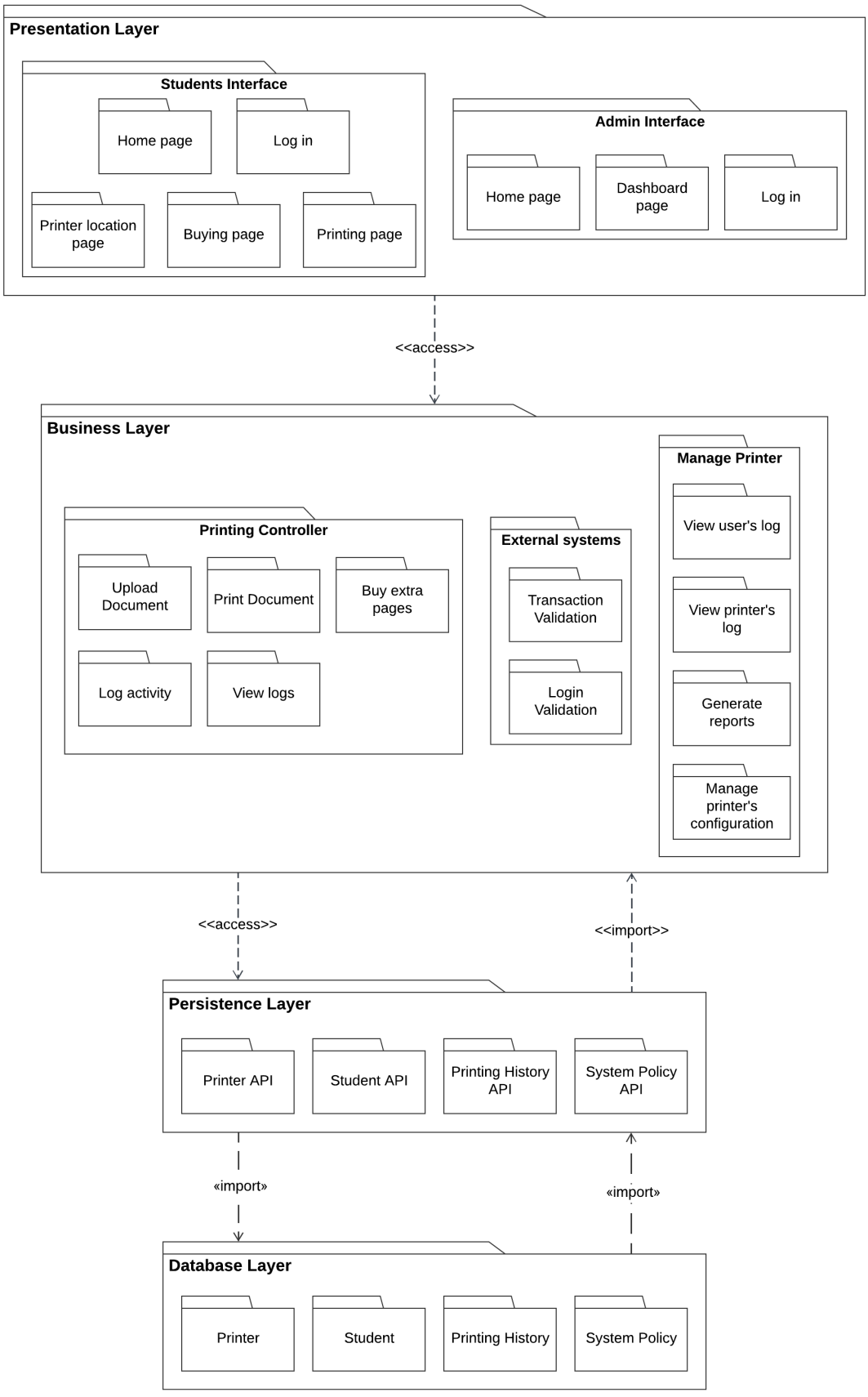
3 Architecture design

3.1 Layered Architecture

Layered architecture is a design pattern used in software engineering to organize a complex system into discrete, logical layers. Each layer has a specific set of responsibilities and interacts with the layers above and below it through well-defined interfaces or APIs (Application Programming Interface). The layered architecture for our system includes the following layers:

- Presentation layer
- Business logic layer
- Persistence layer
- Database layer

Below are detailed specifications of each layer in the architecture.



Hình 21: Layered architecture of the whole system

3.1.1 Presentation strategy (Presentation layer)

This layer is responsible for user interface and interaction. It handles user's command, display information to the user. There are two separate components: ***Student Interface*** and ***Administration Interface***. Both interfaces share some common elements, such as a ***Home page***, which serves as the initial point of entry for users upon accessing or logging into BKPrint. Additionally, there is a ***Login page***, which varies depending on whether the user is a student or an administrator. The ***Student Interface***, specifically, offers a range of features that cater to the needs of students. It includes a ***printer location page***, enabling students to view the available printers and their current status. This feature is invaluable in helping students find nearby, operational printers, saving time and enhancing convenience. The ***Printing page*** within the Student Interface allows users to upload documents and specify the desired printer and printing properties, ensuring that their printing needs are met efficiently. In the event that a student's paper supply is running low, the interface smoothly transitions to the ***Buying page***, where students can effortlessly top up their paper balance to meet their specific requirements. On the other hand, the ***Administration Interface*** is tailored for administrators, equipping them with powerful tools to manage the BKPrint system. The ***Dashboard page*** is the command center for administrators, offering a comprehensive overview of user activities through print history, providing insights into printer quantity and status, and allowing administrators to make real-time changes to printer settings as needed. Furthermore, administrators can export periodic reports from this interface, which is invaluable for tracking system performance and usage trends. In summary, both User Interface and Interaction layer acts as a bridge between users and the system's functionality, enhancing the overall efficiency and effectiveness of printing services.

1. **Students interface**

   The students interface contains 5 pages:

   - Home page
   - Log in
   - Printer location
   - Buying page
   - Printing Page

2. **Admin interface**

   The admin interface contains 3 pages:

   - Home page
   - Log in
   - Dashboard page

   Details of the UI are in this section: User Interface

3.1.2 Business logic layer

This layer contains the core application logic of our smart printing service. It processes request from the presentation layer and communicate with the Persistence layer to perform operations on data. This layer has 3 modules:

1. **Printing controller**

   This module contains the necessary packages for printing actions, which are:

   - Upload documents: When students upload their documents, the **Printing controller** will accesses the **Persistent Layer** and asks **System Policy API** to import **System Policy** from **Database** to get the list of permitted file types. Then, it will check if the type of the document file is in that list and decide whether to accept or reject the file.
   - Print documents: The **Printing controller** will get the list of printers from **Database layer** through **Printer API** from **Persistence layer** to let student choose a printer for printing document. Then, it will let student specify printing properties such as paper size, the number of pages to be printed, one/double sided and the number of copies. The **Printing controller** also maintains a queue of print jobs, which allows multiple users to send their print requests. It manages the order in which documents are printed. After a print job is done, the **Printing controller** will update the page balance of **Student** and the printing log in **Printing History** in **Database layer** through **Student API** and **Printing History API**, respectively, in the **Persistence layer**.
   - Buy extra pages: When student's page balance is not enough for printing document and student is required to buy more pages, or when student want to buy more pages although their page balance is still enough for printing, the **Printing controller** take the number of pages student need to buy, calculating the price and then delegating the transaction to online payment services, which are external systems.
   - Log activity: When a printing job or a transaction occurs, it is saved to the database by the **Printing controller** through the APIs in **Persistence layer**. For example, after the transaction is finished, the **Printing controller** will update the transaction log of **Student** in **Database layer** through **Student API** in **Persistence layer**.
   - View logs: **Printing controller** gets the printing logs and transaction logs from **Student** and **Printing History** in **Database layer** through **Student API** and **Printing History API** in **Persistence layer** and haves that data display to user from the **Presentation layer**.

2. **External systems**

   This module consists of other systems which are integrated with HCMUT_SSPS system, including 2 packages:

- Transaction validation: This belongs to online payment services which make transactions when student buys more pages.
- Login validation: This belongs HCMUT_SSO authentication service which authenticates users before they use the HCMUT_SSPS system.

3. **Manage Printers**

   This module is dedicated for SPSO adminstrator to manage printers, including:

   - View user's log: The SPSO adminstrator can view the printing history of all students. This package will get the printing logs from **Student** in **Database layer** through **Student API** in **Persistence layer**.
   - View printer's log: Similar to User's log, all the activity of the printers are stored in the **Database layer**. When SPSO administrator wants to view printer's log, this package will get data from **Printing History** in **Database layer** through **Printing History API** in **Persistence layer**.
   - Generate reports: The reports about the usage of all printers in HCMUT_SSPS system are generated automatically at the end of each month and each year, then stored in **Printing History** in **Database layer** through **Printing History API** in **Persistence layer**.
   - Manage printer's configuration: The SPSO administrator can add new printers, and change the printer's status through **Printer API** in the **Persistence layer** and these changes will be updated in **Printer** in **Database layer**. SPSO administrator can also change the default number of pages, the dates that the system will give the default number of pages to all students, the list of permitted file types through **System Policy API** in the **Persistence layer** and these changes will be updated in **System Policy** in **Database layer**.

In general, this module is dedicated to the SPSO administrator to track the activity of the system. When there are requests from the administrators, the "Manage Printer" module gets the data through the API provided in the Persistence layer, and based on what the request is, returns the corresponding data to the administrator.

3.1.3 Data storage approach (Database layer)

We will be using MongoDB to store our data, and our database primarily comprises two main entities: **Printer** and **Student**.

1. **Printer** The printer schema contains the following attributes

Attribute	Data Type	Description
Printer ID	String	This represents a unique code for the printer
Printer Brand	String	This represents the brand of the printer

Printer Name	String	This represents the name of the printer
Location	Object	An object containing **Building** and **Room Number** attributes
Building	String	The building where the printer is located
Room	String	The room number where the printer is located
Status	Boolean	State of the printer (enable or disable)
Printed Pages	Integer	This field represents the number of printed pages

Note that in the above attribute table, *Building* and *Room* are two sub-attributes of **Location**. In order to provide a clearer illustration of how our group store data, let's examine the following example.

```
"printer":{
    PrinterID: 1,
    PrinterBrand: "Canon",
    PrinterName: "Canon LBP2900",
    Location:{
        Building: "A4",
        Room: "402",
    },
    Status: True,
    PrintedPages: 201,
}
```

2. **Student**

The student schema contains the following attributes

Attribute	**Data Type**	**Description**
Student ID	String	This represents a unique identifier for the student
Student Name	String	The name of the student
Student Email	String	The email of the student
Student Faculty	String	The faculty of the student
Remaining Pages	Integer	The number of remaining pages for the student

Transaction History	Object	A list of object which containing the price, purchased pages and the time of the transaction
Time	DateTime	The time that the transaction was conducted
Price	Float	The cost associated with the purchase
Purchased pages	Integer	The number of pages purchased
Printing History	Object	A list of object which containing printing details
Filename	String	The name of the file printed
Time	DateTime	The time of the printing job
Printed pages	Integer	Show the number of pages that was used for the printing job
Paper type	String	Type of papers that was used for the printing job (A3, A4, A5, etc)
Location	String	Represents the building of the used printer

Note that in the above table *time* of the **TransactionHistory** is different from *time* of the **PrintingHistory**.

- **Transaction History** is a list of derived attribute that consist of
  - Time
  - Price
  - Purchasing pages
- **Printing History** is a list of derived attribute that consist of
  - Filename
  - Time
  - Printed pages
  - Paper type
  - Location

In order to provide a clearer illustration of how our group store data, let's examine the following example.

```
"student":{
    StudentID: "2152788,
    StudentName: "Ta Ngoc Nam",
    Email: "nam.ta8989@hcmut.edu.vn",
    Faculty: "Ky Thuat May Tinh",
```

```
        RemainingPages: 20,
        TransactionHistory: {
            Time:
            Price:
            PurchasedPages:
        },
        PrintingHistory: [
            {
                Time: 00:00, 22/10/2023,
                FilenName: "file1.pdf",
                PrintedPages: 15,
                PaperType: "A4",
                Location: "A4",
            },
            {
                Time: 07:00, 23/10/2023,
                FileName: "file2.pdf",
                PrintedPages: 20,
                PaperType: "A5",
                Location: "B1",
            }
        ]
        TransactionHistory: [
            {
                Time: 09:00, 22/10/2023,
                Price: 50000
                PurchasedPages: 100
            },
            {
                Time: 10:00, 23/10/2023,
                Price: 20000
                PurchasedPages: 40
            }
        ]
    }
```

3. **Printing History**

   The printing history schema contains the following attributes

Attribute	**Data Type**	**Description**

Index	Integer	This represents a unique index for each printing activity
Student ID	String	This represents a unique identifier for the student provided by University
File Name	String	The name of the file being printed
Printing Time	Datetime	The time when the printing occurred
Printer Name	String	The name of the printer used for printing
Building	String	The building where the printer is located

In order to provide a clearer illustration of how our group store data, let's examine the following example.

```
"printer":{
    index: 0,
    StudentID: 2152788,
    StudentName: "Ta Ngoc Nam",
    FilenName: "file1.pdf",
    PrintingTime: 00:00, 22/10/2023
    PrinterName: "Canon LBP2900",
    Buidling: "A4",
}
```

4. **System Policy**

The system policy history schema contains the following attributes

Attribute	**Data Type**	**Description**
Default Page	Integer	This represents the number of pages that student received.
Allocated Date	Datetime	The date when system give the default number of pages to all students.
Maximum File Size	Integer	Represent the maximum size of the file. Default unit is MB
Permitted File Type	List of String	The list contains every permitted types of files to be printed.

In order to provide a clearer illustration of how our group store data, let's examine the following example.

```
"systemPolicy":{
    defaultPage: 50,
    DateTime: 00:00, 30/11/2023,
    MaximumFileSize: 10MB,
    PermittedFileType: ["pdf","docs"],
}
```

3.1.4 API management (Persistence layer)

API stands for application programming interface, which is a set of definitions and protocols for building and integrating application software. APIs let your product or service communicate with other products and services without having to know how they're implemented.
According this HCMUT_SPSS system, there are 3 main API services are used

- API connect between **Authentication Serive** and **System Controller** to validate user account.
- API connect between **Online Payment Serive** and **System Controller** to help student buy pages in their wants.
- API connect components within the system to transfer data from controller to the database and vice versa.

In this section, our group focus on the third API service. It can be observed in our architecture diagram that their are 3 main object APIs corresponding to 3 objects in the database

1. **Printer API**

   This object API has the following methods

Method	**Passing Parameter**	**Description**
getPrinterBrand	String (ID)	Pass the printer ID to get the information of printer's brand from the database
getPrinterName	String (ID)	Pass the printer ID to get the information of printer's name from the database
getPrinterStatus	String (ID)	Pass the printer ID to get the information of printer's status from the database

getPrinterBuilding	String (ID)	Pass the printer ID to get the information of printer's building location from the database
getPrinterRoomNumber	String (ID)	Pass the printer ID to get the information of printer's room number location from the database
getPagePrinted	String (ID)	Pass the printer ID to get the information of how many page has been printed by this printer from the database
addNewPrinter	Object (Printer Information)	Create a new printer and add it to the database
changePrinterStatus	String (ID)	Pass the printer ID to change the printer status and also update that status to the database

This API will be called in some following cases

- When student calls the **Printing Document**, the **Printing Document** calls the **PrinterAPI** to modify the number of pages has been printed.
- When SPSO (admin calls the **View Printer's Log**, the **View Printer's Log** calls the **PrinterAPI** to get printer's information.
- When SPSO (admin) calls the **Manage printer's configuration**, the **Manage printer's configuration** calls the **PrinterAPI** to update printer's status or add a new printer to the database.

2. **Student API**

This object API has the following methods

Method	**Passing Parameter**	**Description**
getStudentName	String	Pass the student ID to get the information of the student's name from the database.
getStudentEmail	String	Pass the student ID to get the information of the student's email from the database.

getStudentFaculty	String	Pass the student ID to get the information of the student's faculty from the database.
addPrintingActivity	Object (Printing Activity)	Create a new printing activity and add it into the printing history list.
addTransactionActivity	Object (Transaction Activity)	Create a new transaction activity and add it into the transaction history list.
getPrintingTime	String (ID), Integer (Index)	Pass the student ID and the index of the chosen printing activity to get the time of that printing activity.
getPrintingFileName	String (ID), Integer (Index)	Pass the student ID and the index of the chosen printing activity to get the filename of that printing activity.
getPrintingPaperNumber	String (ID), Integer (Index)	Pass the student ID and the index of the chosen printing activity to get the information about how many pages have been printed by this printer from the database
getPrintingPaperType	String (ID), Integer (Index)	Pass the student ID and the index of the chosen printing activity to get the type of paper have been printed in this printing work (A3, A4, A5)
getPrintingLocationBuilding	String (ID), Integer (Index)	Pass the student ID and the index of the chosen printing activity to get the information of the printer's building location from the database.
getTransactionTime	String (ID), Integer (Index)	Pass the student ID and the index of the chosen transaction activity to get the time of that transaction.
getTransactionCost	String (ID), Integer (Index)	Pass the student ID and the index of the chosen transaction activity to get the price that student have to pay for this transaction.

getTransactionPage	String (ID), Integer (Index)	Pass the student ID and the index of the chosen transaction activity to get the number of page that student purchase in this transaction. In default, the paper type is A4 paper.

When student go to their personal account page. The controller render all necessary information through this API. For example, consider our group User Interface (UI) which has been introduced in previous sections. The account page contains this following information:

- Student name (Ta Ngoc Nam)
- Student ID (2152788)
- Student email (nam.ta8989@hcmut.edu.vn)
- Student Faculty (Ky Thuat May Tinh)
- List of all printing activity
- List of all transaction

Moreover, when student complete a printing work. Then this API is called in order to log printing action and transaction action into the database.

3. **Printing History API**

   Since, this API is designed for admin to check the printing history of **all** student so that it only contains 2 methods

   - **GetAllPrintingHistory** is called when admin want to retrieves all printing activities in chronological order, starting from the most recent and extending to the furthest in time
   - **AddNewPrintingHistory** is called every time a student complete a printing activity, adding that printing activity into the database.

4. **System Policy API**

   This object API has some following methods

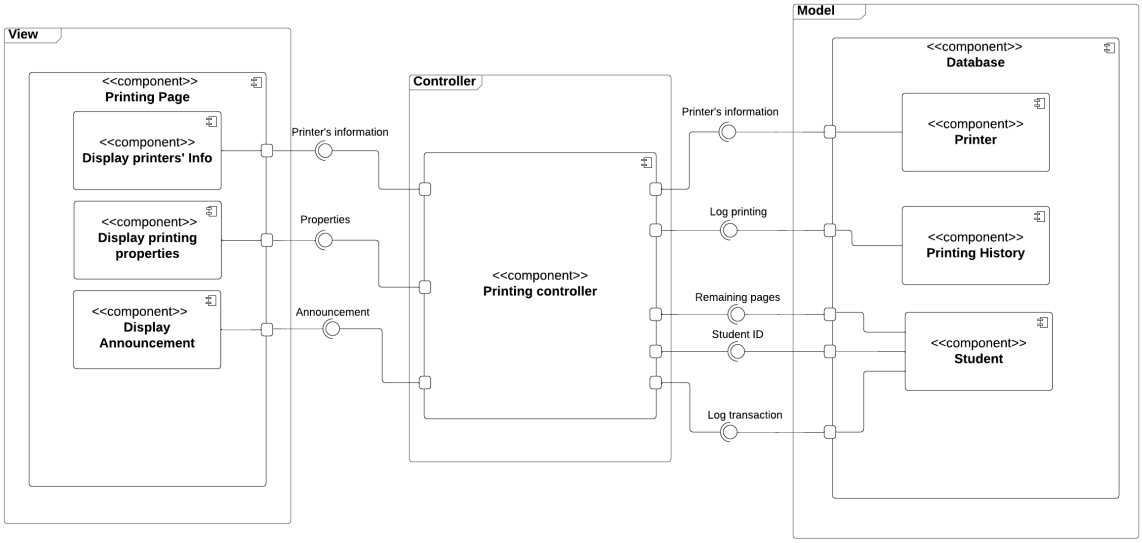
Method	**Passing Parameter**	**Description**
getDefaultPageNumber	None	Get the number of pages that student received

setDefaultPageNumber	None	Change the number of pages that student received
getAllocDate	None	Get the date when system give the default number of pages to all students
setAllocDate	None	Change the date when system give the default number of pages to all students
getMaximumPageSize	None	Get the maximum size of the file
setMaximumPageSize	None	Change Get maximum size of the file
getPermittedFile	None	Get the list contains every permitted types of files to be printed
setPermittedFile	None	Change the list contains every permitted types of files to be printed

This API is called when admin want to change the configuration of the system such as changing the default number of pages, the dates that the system will give the default number of pages to all students, the permitted file types accepted by the system.

3.2 Component Diagram

3.2.1 "Print Document" module



Hình 22: The *Print Document* Component Diagram

The component diagram above shows components of the system in the **Print Document** module in MVC pattern. The **View** package contains **Printing Page** component, which is the user interface when a student prints document. **Printing Page** consists of 3 smaller components:

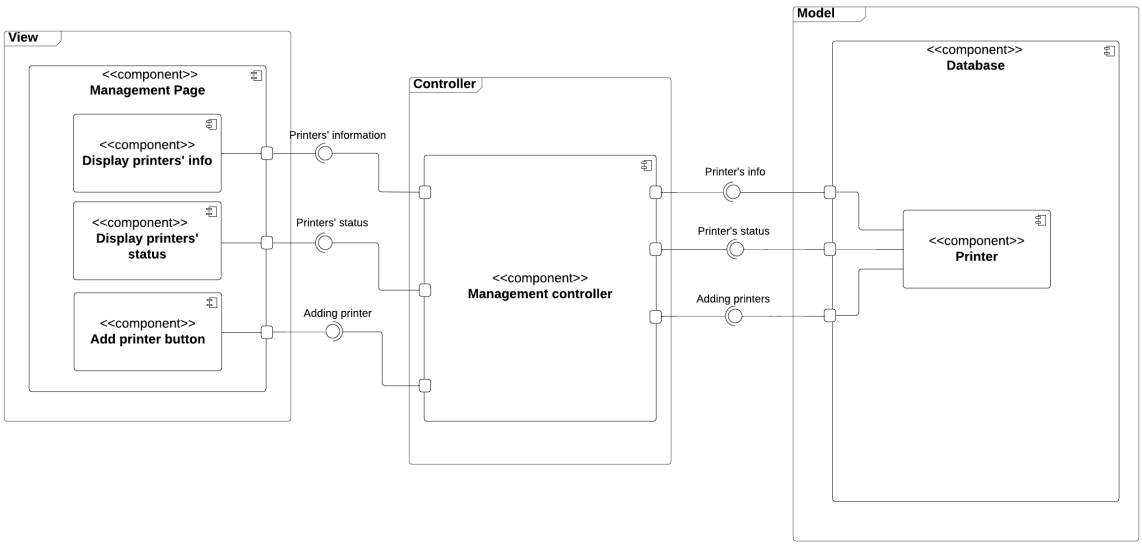
- **Display printers' info** component: the section which displays a list of printers and their information such as printer model and location for student to choose. This requires **Printer's information** interface from the component **Printing controller**.
- **Display printing properties** component: the section which allows student to specify printing properties such as paper size, the number of pages to be printed, one/double sided and the number of copies. This requires **Properties** interface from the component **Printing controller**.
- **Display announcement** component: requires **Announcement** interface from the component **Printing controller** and consists of
  - The page which notifies student that their document is printing and reminds student to go to the printer to take the document.
  - The page which notifies student that his/her page balance is not enough and requires he/she to buy more pages.

The **Controller** package contains **Printing controller** component, which provides interfaces for **Printing Page** component in View and requires interfaces from **Database** component in Model.

- **Printing controller** component requires **Printer's information** interface from **Printer** component in **Database** to provide for **Display printer's info** component.
- **Printing controller** component requires **Log printing** interface from **Printing History** component in **Database** to log the printing action of students as well as printers.
- **Printing controller** component requires **Remaining page** interface from **Student** component in **Database** to check whether the page balance of the student is enough or not.
- **Printing controller** component requires **Student ID** interface from **Student** component in **Database** to log the printing actions, which requires student ID.
- **Printing controller** component requires **Log transaction** interface from **Student** component in **Database** to log the number of pages that a student buys and the amount of money he/she pays.

The **Model** package contains the **Database** component, which consists of 3 smaller components: **Printer** component, **Printing History** component and **Student** component. The **Database** component delegates the providing interfaces to its internal components through ports.

3.2.2 "Manage Printers" module



Hình 23: The *Manage Printers* Component Diagram

The component diagram above shows components of the system in the **Manage Printers** module in the MVC pattern. The **View** package contains **Management Page** component, which is the user interface when a SPSO administrator manages printers. **Management Page** consists of 3 smaller components:

- **Display printers' info** component: the section which displays a list of printers and their information such as printer ID, brand name, printer model, and location for the SPSO administrator to manage. This requires **Printer's information** interface from the component **Management controller**.
- **Display printers' status** component: the section which displays the current status of each printer and a column which allows the SPSO administrator to change the printer's status. This requires **Printers' status** interface from the component **Management controller**.
- **Add printer button** component: provides **Adding printer** interface for the **Management controller** component to add a new printer to the printers' list.

The **Controller** package contains **Management controller** component, which provides interfaces for **Management Page** component in View and requires interfaces from **Database** component in Model.

- **Management controller** component requires **Printer's info** interface from **Printer** component in **Database** to provide for **Display printer's info** component.
- **Management controller** component requires **Printer's status** interface from **Printer** component in **Database** to retrieve the current status of the printer, as well as changing the status of that printer.
- **Management controller** component requires **Adding printers** interface from **Printer** component in **Database** to add a new printer to the printers' list.

The **Model** package contains the **Database** component, which consists of a smaller component: **Printer**. The **Database** component delegates the providing interfaces to its internal component through ports.

4 Implementation

4.1 Version control system

4.1.1 Setting up an online repository

- Github repo: bkprint-web

4.1.2 Adding documents

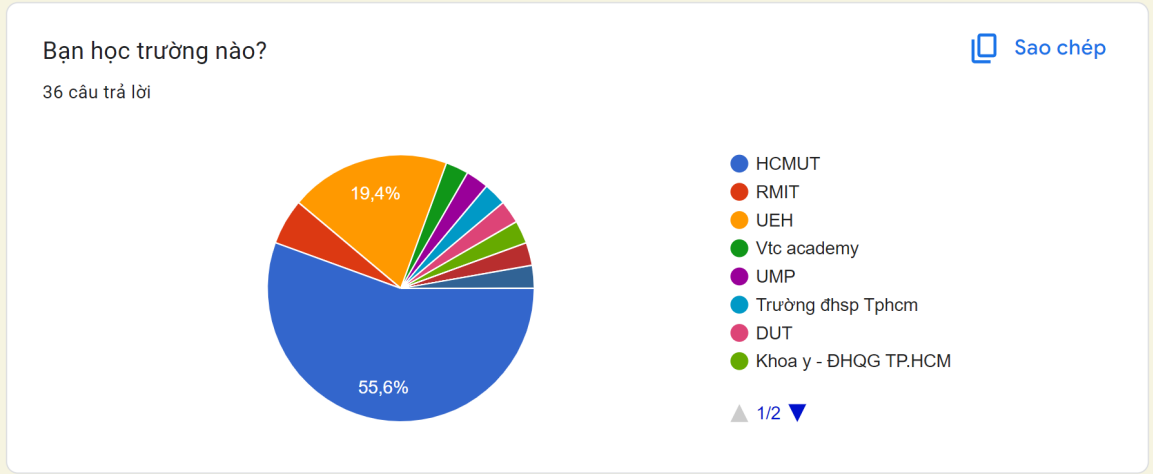
- Requirement: Requirement.pdf
- System modelling: Modelling.pdf
- Architectural design: Architecture.pdf

Moreover, there is a CHANGELOG.md that shows the implementation diary of our team.

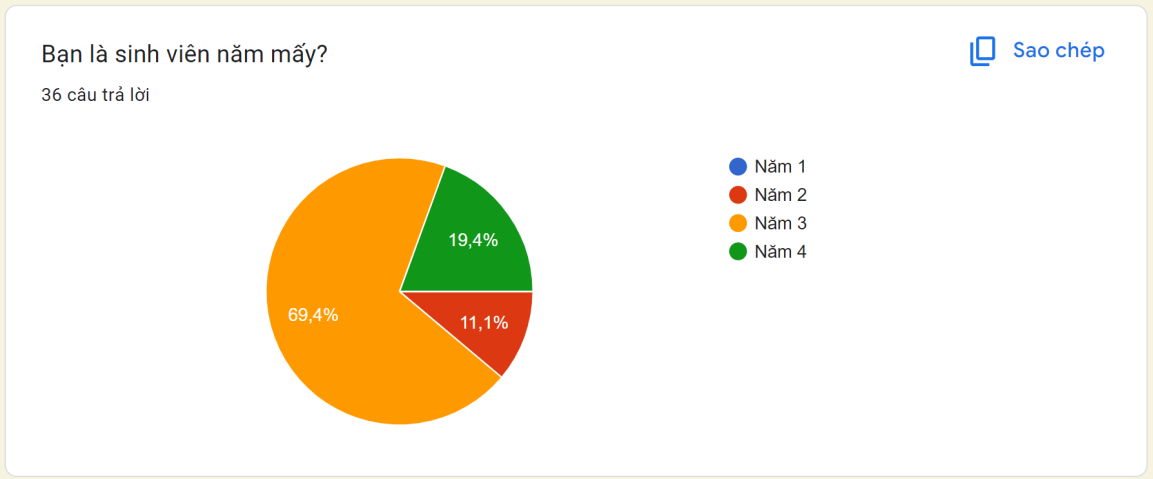
4.2 Usability test

4.2.1 Participants / Testers

The testers for this test is students from universities in Ho Chi Minh City. Most of them are third year student while the rest spread from second year to last year (forth year).



Hình 24: Student from HCMUT ocupied the highest percentage (55.6%)



Hình 25: There's no freshmen involve in this test

4.2.2 Task Definition

This section of the report focuses on the specific tasks assigned to participants to evaluate the usability of our MVP's user interface. Testers will receive the Google Form link reference to the survey (through social media). In this form, there are 9 questions, each question has two part: grading section and comments section. Participants in the test are provided two link which are reference to the prototype of User Interface designed in Figma as well as the form. They suposed to experience the UI interface and complete the form.

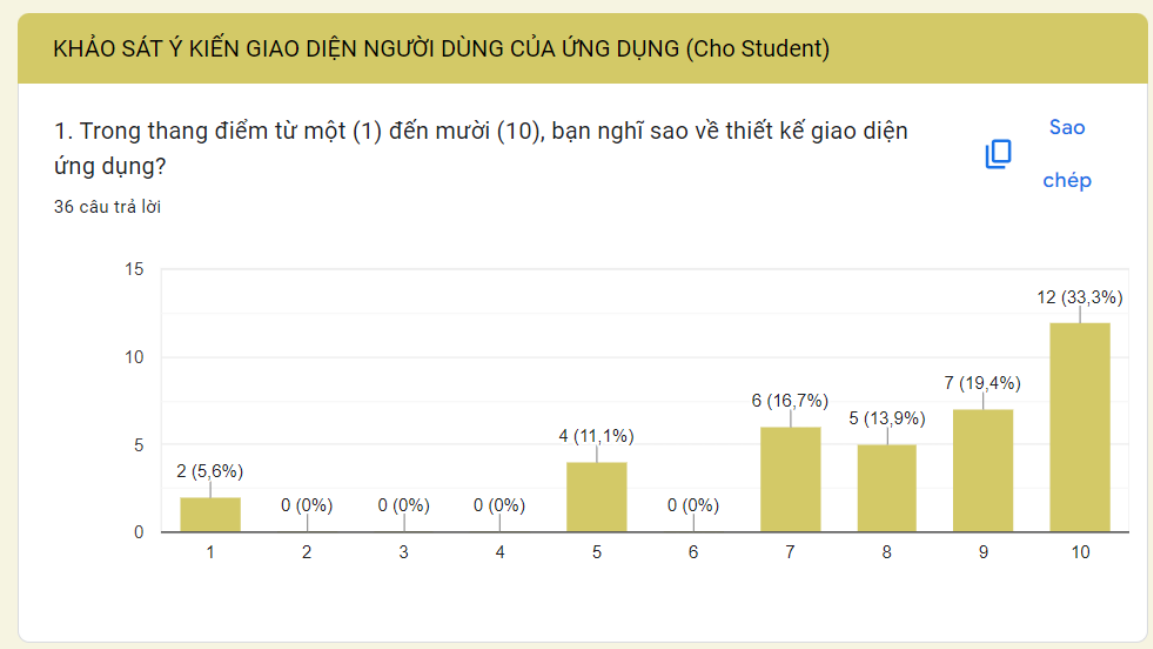
- Survey form: Google Form
- Figma prototype: UI Prototype

4.2.3 Test Strategy

Our test strategy employs a remote unmoderated approach with a focus on qualitative analysis. This method has been chosen for its flexibility and efficiency, allowing participants to engage with the user interface at their convenience, without the need for real-time interaction with a facilitator. Participants will complete tasks designed to evaluate various aspects of the user experience. Their interactions will be recorded for subsequent qualitative analysis. This approach not only facilitates a natural user experience by removing the potential influence of a moderator but also provides rich, qualitative data. We'll analyze the recordings to gain insights into user behaviors, preferences, and challenges encountered, focusing on subjective measures like user satisfaction, perceived usability, and overall experience. This strategy is particularly effective for understanding the user's perspective in a realistic setting, offering invaluable insights for enhancing the interface's user-friendliness and intuitiveness.

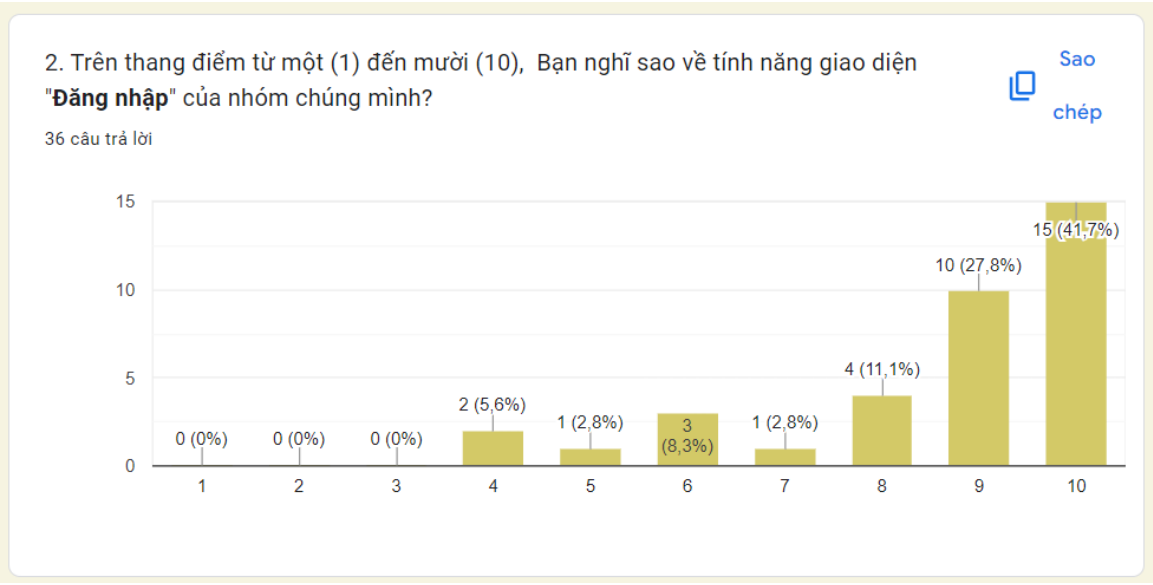
4.2.4 Feedback from testers

In this critical section of our usability test report, we delve into the valuable insights gathered directly from our participants. In total, we received 36 answers from testers. The feedback provided by testers is a cornerstone of our analysis, offering a first-hand account of user experiences, perceptions, and challenges encountered while interacting with our MVP's user interface. This feedback is not just a collection of individual responses; it serves as a vital tool for understanding the nuances of user interaction, pinpointing areas of success, and identifying aspects in need of improvement. By analyzing these candid responses, we can extract meaningful patterns and trends that will guide our next steps in refining the user interface. This section not only highlights the specific comments and suggestions from participants but also contextualizes them within the broader scope of our usability goals, providing a comprehensive understanding of how real users interact with and perceive our product. The following image show the score that the testers gave for each main section of the prototype:



Hình 26: Score chart for the overall design

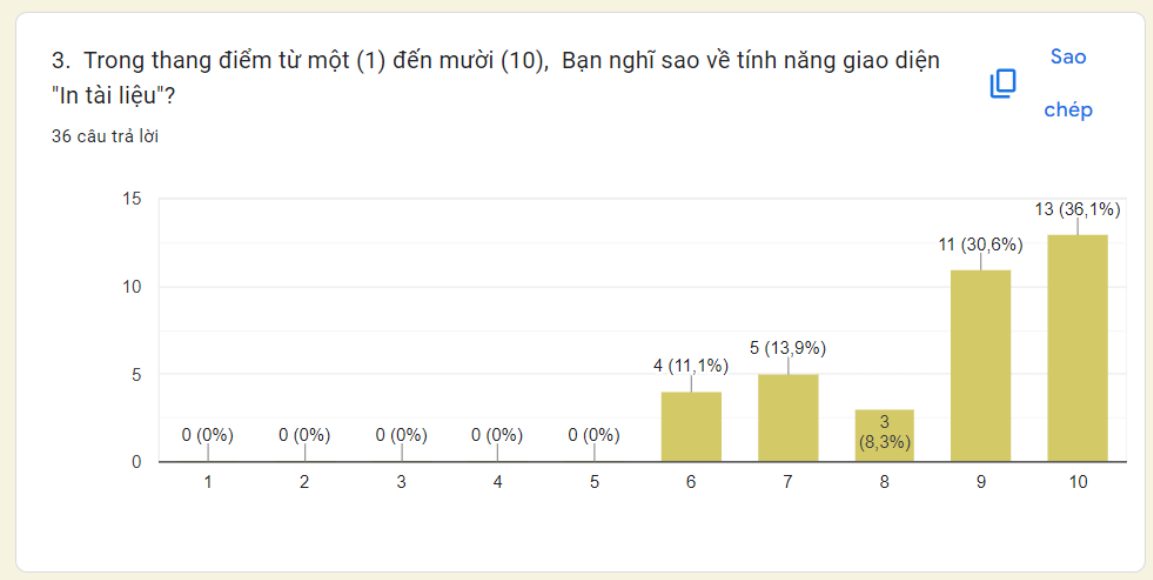
The highest number of respondents, 33.3%, gave the design the highest rating of 10, indicating a very favourable impression. The second most common score was 7, with 19.4% of responses, suggesting that a significant portion of users also found the design to be good, though with some room for improvement. Lower scores were less frequent, indicating that overall, user sentiment towards the interface design was quite positive. Some giving lower scores comment that the backgrounds used on the website are not really relevant to its purpose.



Hình 27: Score chart for Login

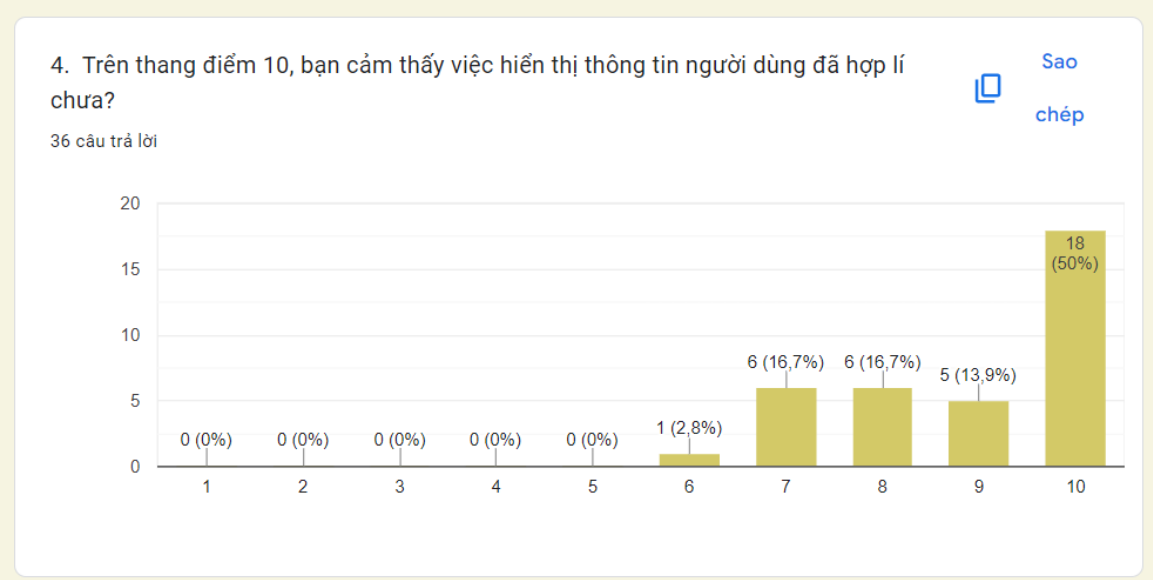
Most users are satisfied, with the highest percentage of respondents (41.7%) giving a score of 10, indicating they view the feature very favorably, followed by 27.8% awarding a score

of 9, suggesting general user satisfaction with the login interface. The reason for the bad responses was about the backgorund and text align (designing field).



Hình 28: Score chart for Printing Document

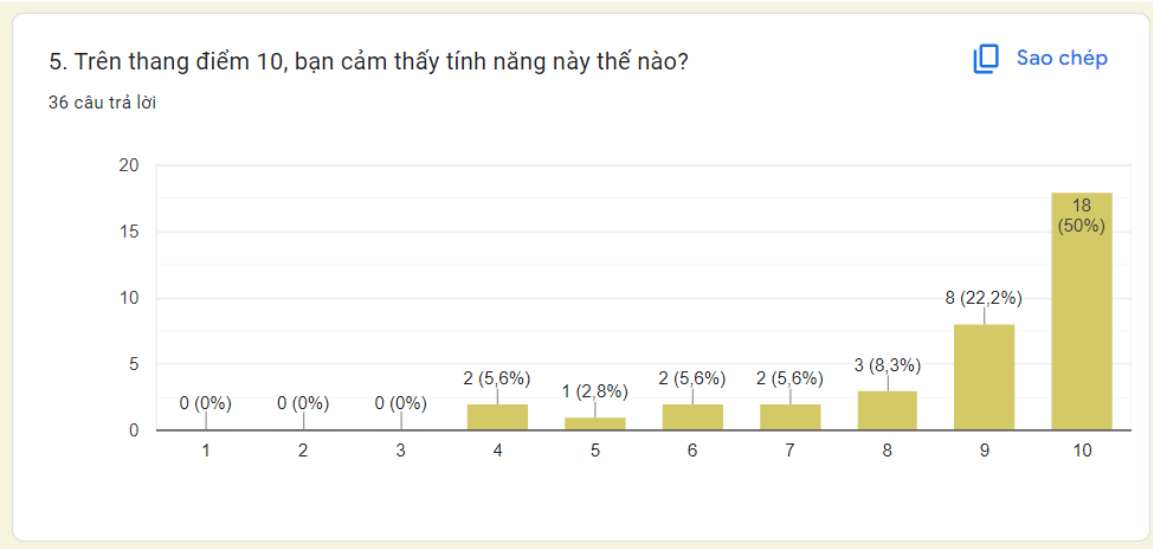
The majority of respondents, 36.1%, gave the highest rating of 10, suggesting a very positive reception for this feature. Another significant portion, 30.6%, rated it 9, reinforcing the strong approval. The least favorable rating given by participants was 5, with only 11.1% of respondents choosing this mid-scale rating. No ratings were given at the lowest end of the scale (1 to 4), indicating an overall high satisfaction with the "In tài liệu" feature among the users surveyed.



Hình 29: Score chart for User Infomation

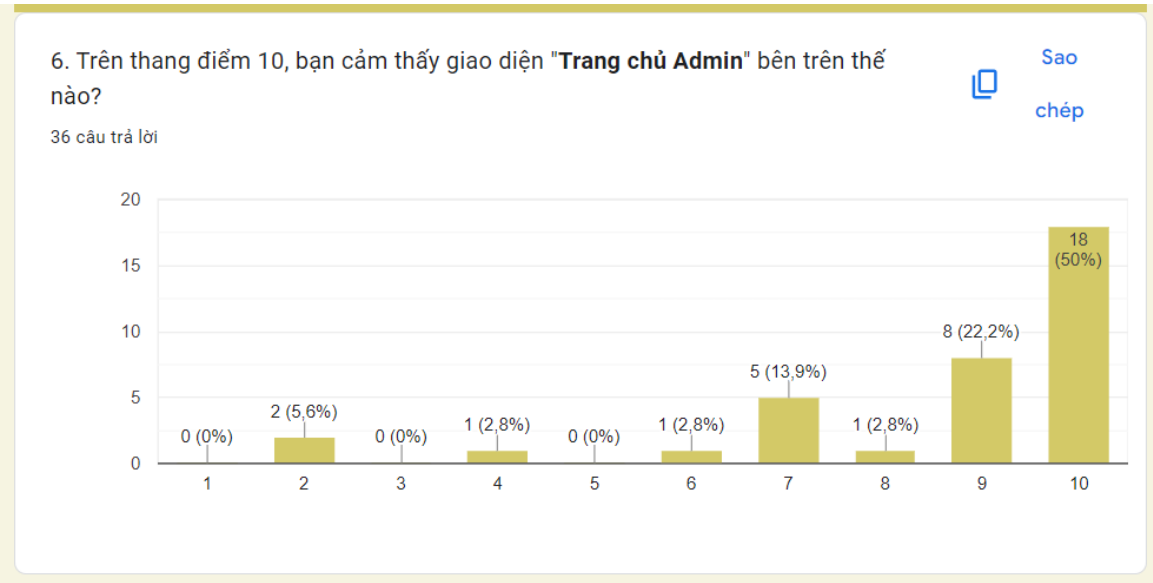
Half of the respondents (50%) gave a perfect score of 10, indicating they find the information

presentation entirely appropriate. The rest of the scores are distributed between 6 and 9, with no responses below 5, suggesting overall positive feedback with some room for improvement in how user information is displayed.



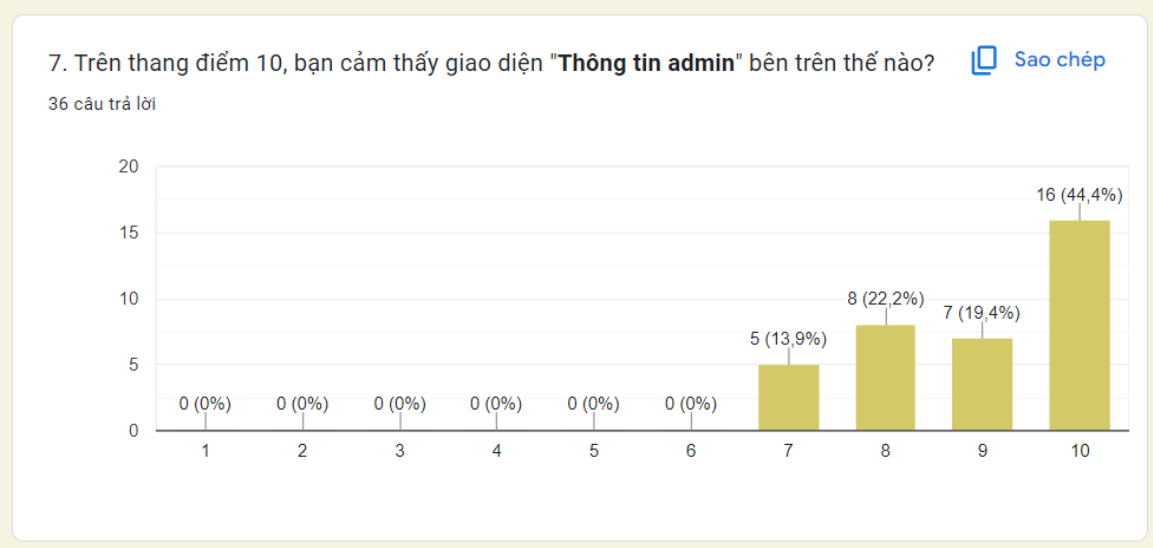
Hình 30: Score chart for Contact / Support

A significant majority, 50%, rated the feature as a 10, indicating a very high level of satisfaction. There are also notable responses at the lower end of the scale, with ratings of 4 and 7 each making up 5.6% of the responses, and a single respondent (2.8%) giving a rating of 5. The overall distribution suggests that while the majority of users are extremely satisfied, a few users see room for improvement.



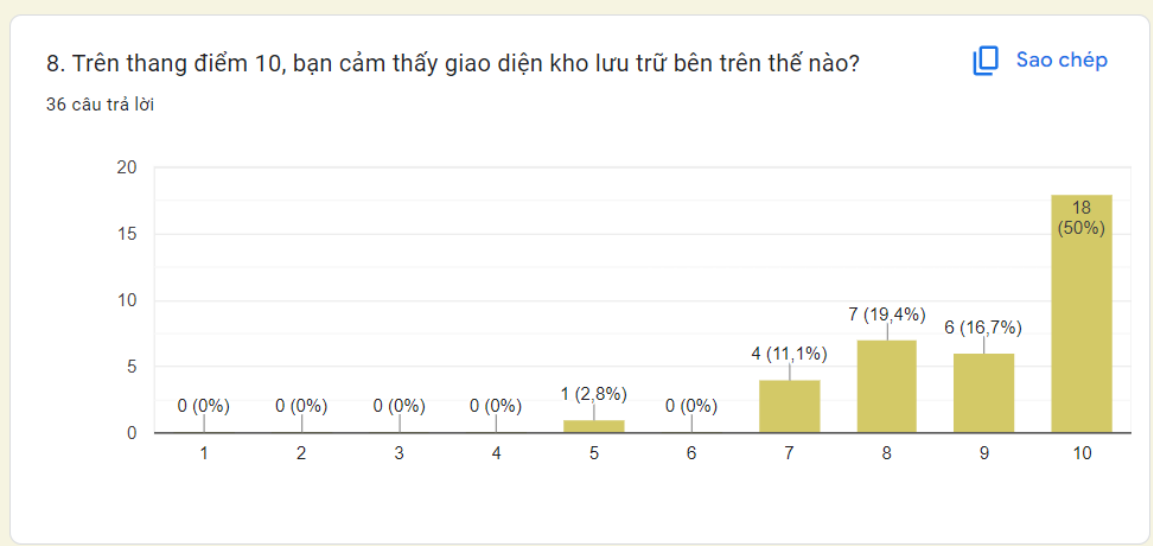
Hình 31: Score chart for Admin Homepage

The **Admin Homepage** (Trang chủ Admin) feature also saw 50% of users giving a score of 10, reflecting a high level of user satisfaction. A fair number of users rated it 9 (22.2%), reinforcing a positive view of this feature.



Hình 32: Score chart for Admin Info

The **Admin Information** (Thông tin admin) feature received a top score of 10 from 44.4% of users, with the next most common score being 7 (19.4%). This suggests that while many users are satisfied, there's a notable group that sees room for improvement.



Hình 33: Score chart for Report Storage

For the **Storage Interface** (Kho lưu trữ), 50% of respondents rated it a perfect 10, suggesting excellent satisfaction. Scores of 7 and 9 were also given by 19.4% and 16.7% of respondents, respectively, indicating generally positive feedback.

Overall, the feedback across all three features is quite positive, with the majority of users indicating high levels of satisfaction, particularly with the **Storage Interface** and **Admin Homepage**. However, there is some divergence in opinions on the **Admin Information** feature, signalling potential areas for enhancement.

4.2.5 Conclusion

The results show that testers highly evaluate the prototype for its innovative approach and user-friendly design (score 5 to 10). On the other hand, the design still contains lots of areas that require refinement. Participants pointed out specific elements that could be improved for a more seamless user experience. Key feedback included suggestions for a more intuitive navigation system, as some found certain menus and features not immediately apparent. Additionally, while the overall aesthetic was well-received, there were recommendations to adjust certain visual elements, such as font sizes and color contrasts, to enhance readability and accessibility. Another noteworthy point of feedback was related to the responsiveness of the application across different devices, with a few testers experiencing inconsistencies when switching between mobile and desktop platforms. This feedback is invaluable as it provides actionable insights into how we can evolve the prototype into a more polished and universally user-friendly product. By addressing these areas of concern, we can not only meet but exceed user expectations in future iterations.