

Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering



Software Engineering (CO3001)

Student Smart Printing Service System Modelling

Authors:

Nguyen Thanh Thao Nhi
Ta Ngoc Nam
Le Thanh Binh
Ly Tran Phuoc Tri
Phan Gia Bao
Hoang Tien Duc
Nguyen Huu Tho

Student's ID:

2152840
2152788
2112897
2153920
2153210
2152520
2153843

Instructors:

PhD. Truong Thi Thai Minh

Completion date: 19th November, 2023

Mục lục

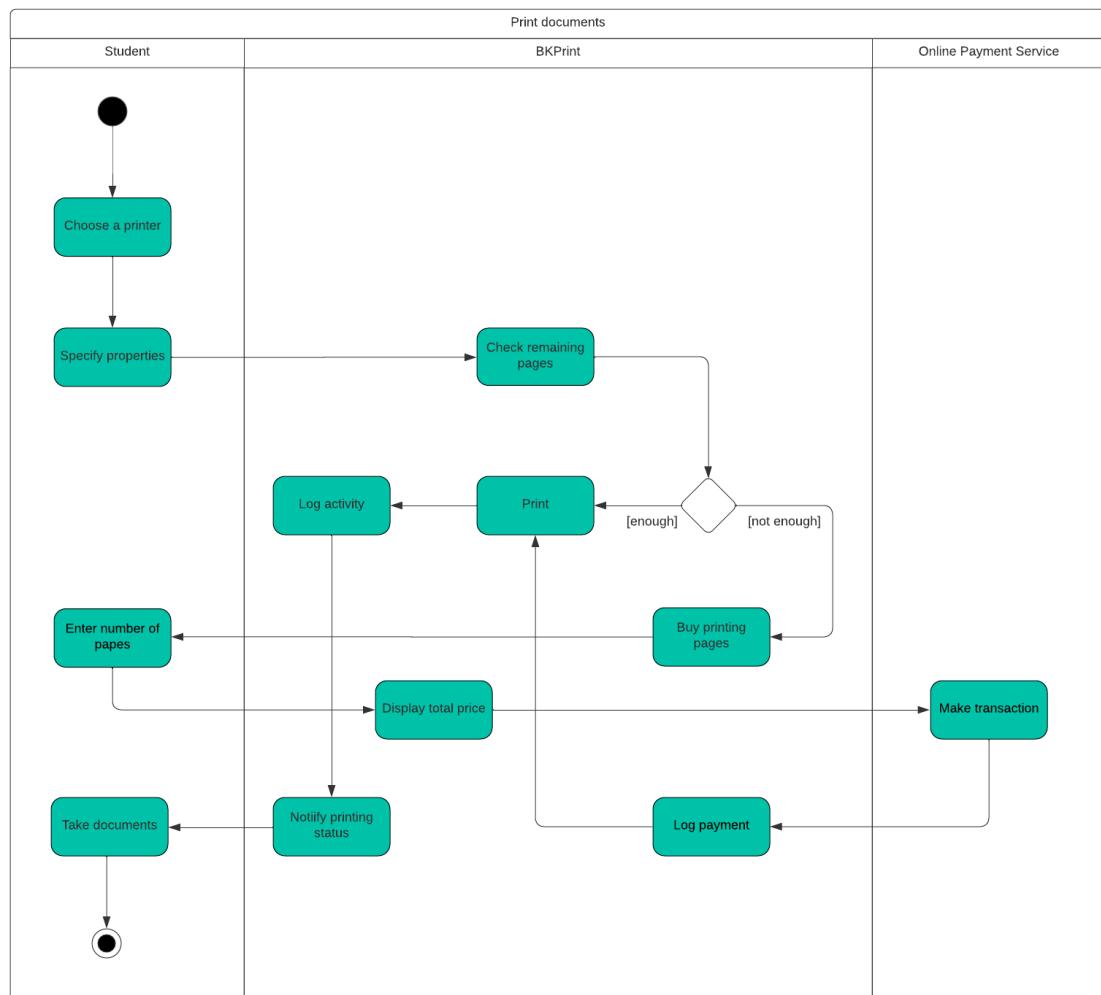
1	Introduction	3
2	Activity Diagram	3
2.1	"Print Document" module	3
2.2	"Manage Printer" module	4
3	Sequence Diagram	5
3.1	"Printing Document" module	5
3.2	"Manage Printer" module	7
4	Class Diagram	8
4.1	"Printing Document" module	8
4.2	"Manage Printer" module	10
5	User Interface	13
6	Conclusion	20

1 Introduction

This Modeling Document is a crucial component in designing and comprehending the BKPrint system. It consists of various visual representations, including Activity Diagrams, Sequence Diagrams, a Class Diagram, and the developed MVP 1 User Interface. These elements provide a comprehensive view of system-stakeholder interactions, chronological flow, structural design, and a tangible representation of the central dashboard for the chosen modules.

2 Activity Diagram

2.1 "Print Document" module



Hình 1: The *Print Document* Activity Diagram

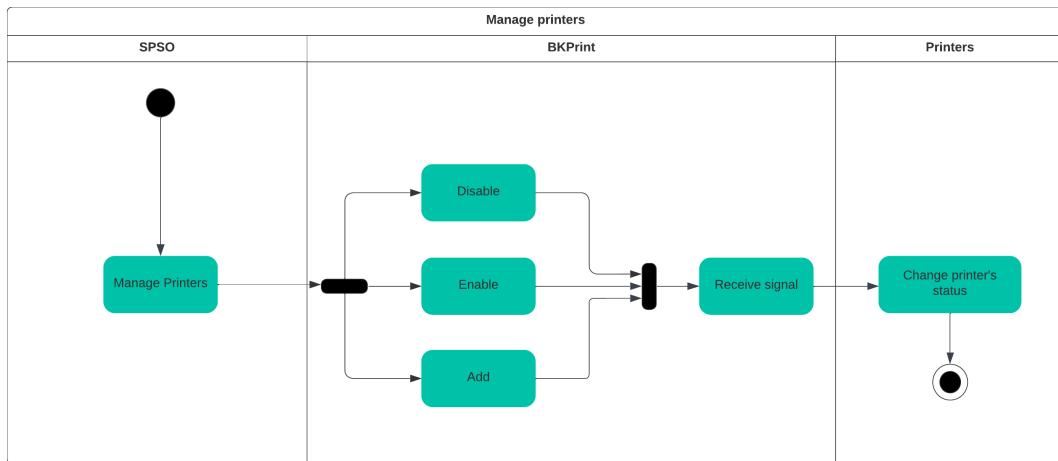
The diagram illustrates the document printing process for students using BKPrint, an online printing service, involving three key swimlanes: **Student**, **BKPrint**, and **Online Payment Service**.

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.

2.2 "Manage Printer" module

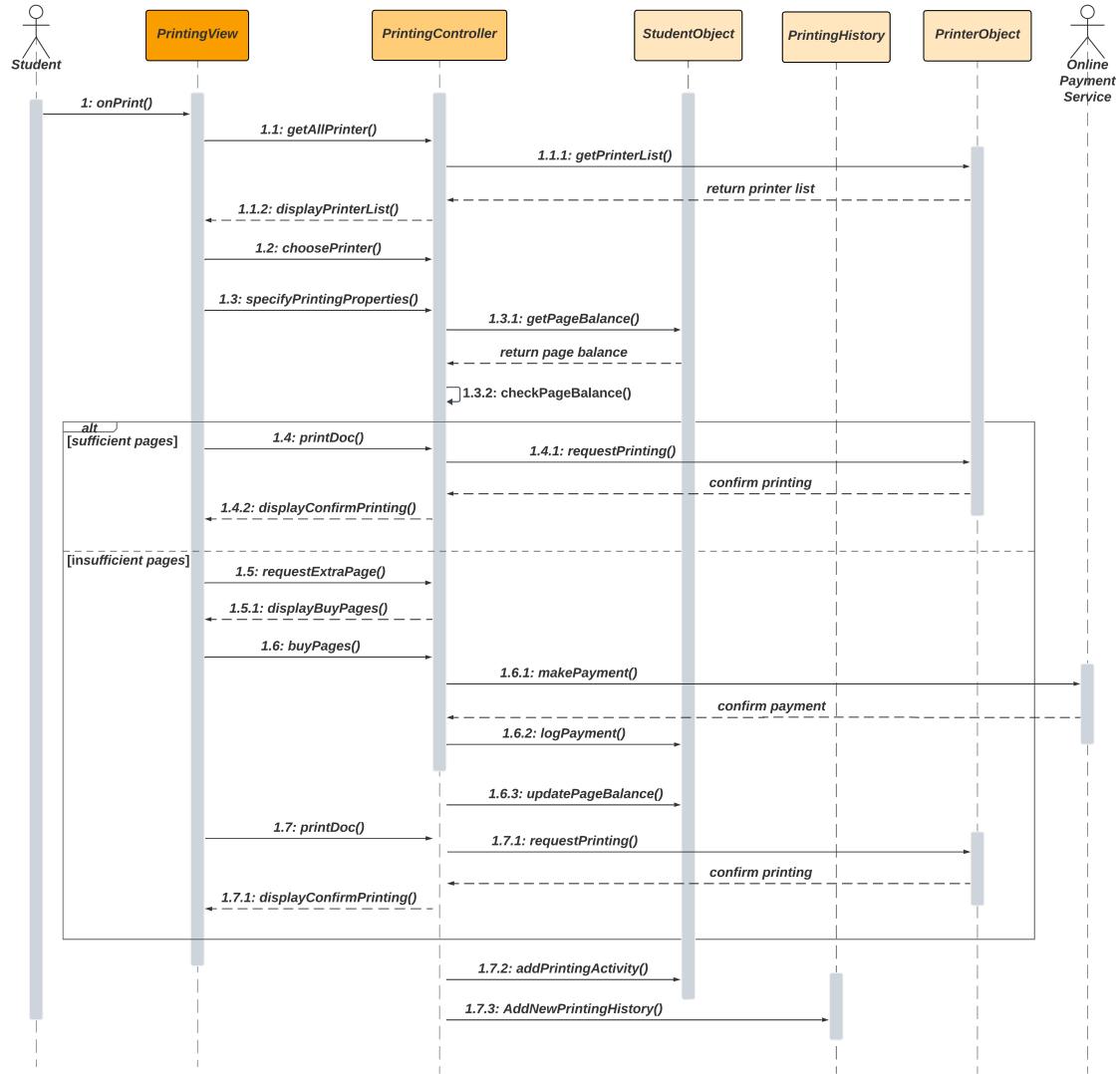


Hình 2: The *Manage Printer* Activity Diagram

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.

3 Sequence Diagram

3.1 "Printing Document" module



Hình 3: 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 **Printing Controller**, 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

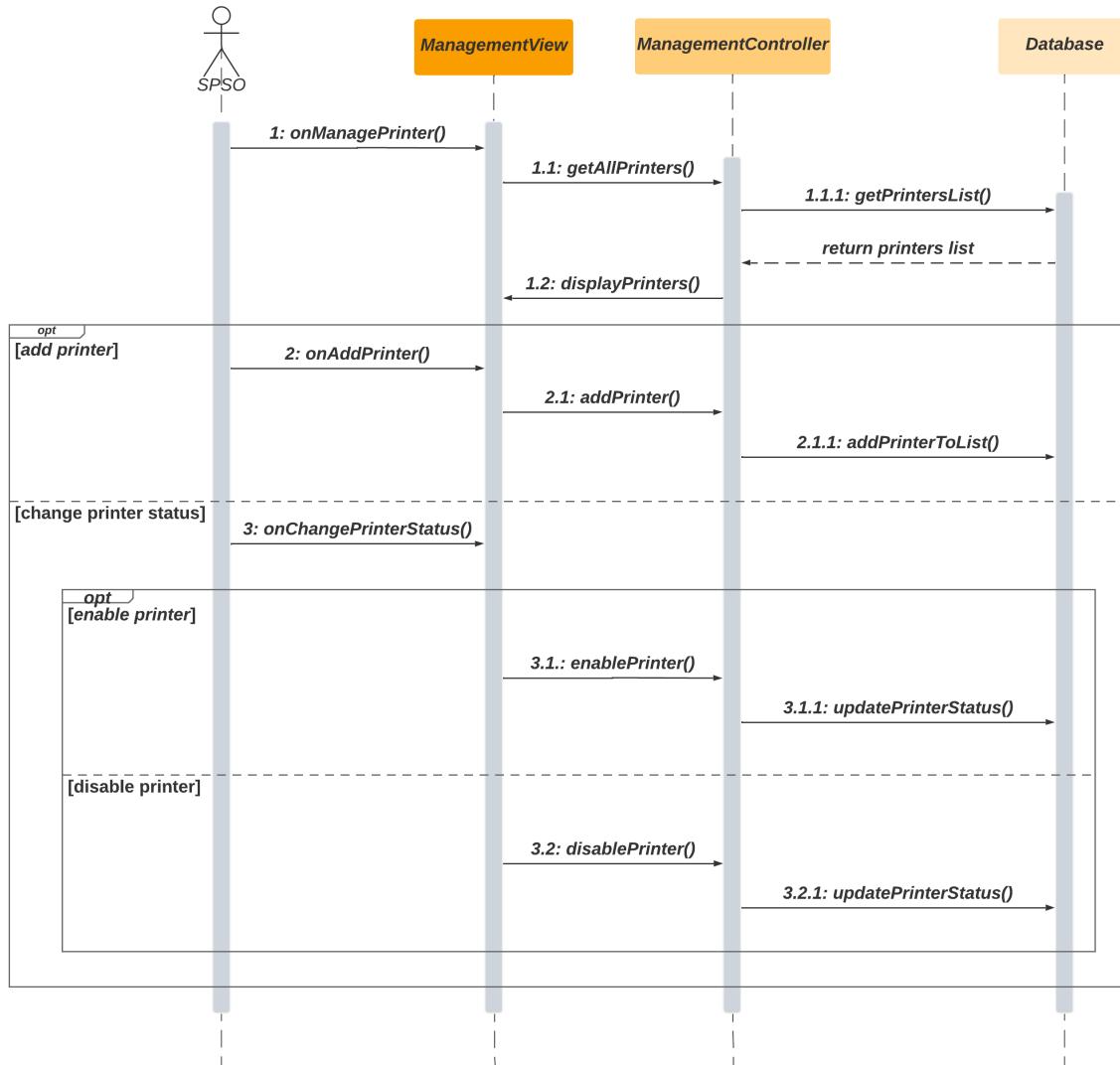
paper size and number of copies.

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 call **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.

3.2 "Manage Printer" module



Hình 4: 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()`.

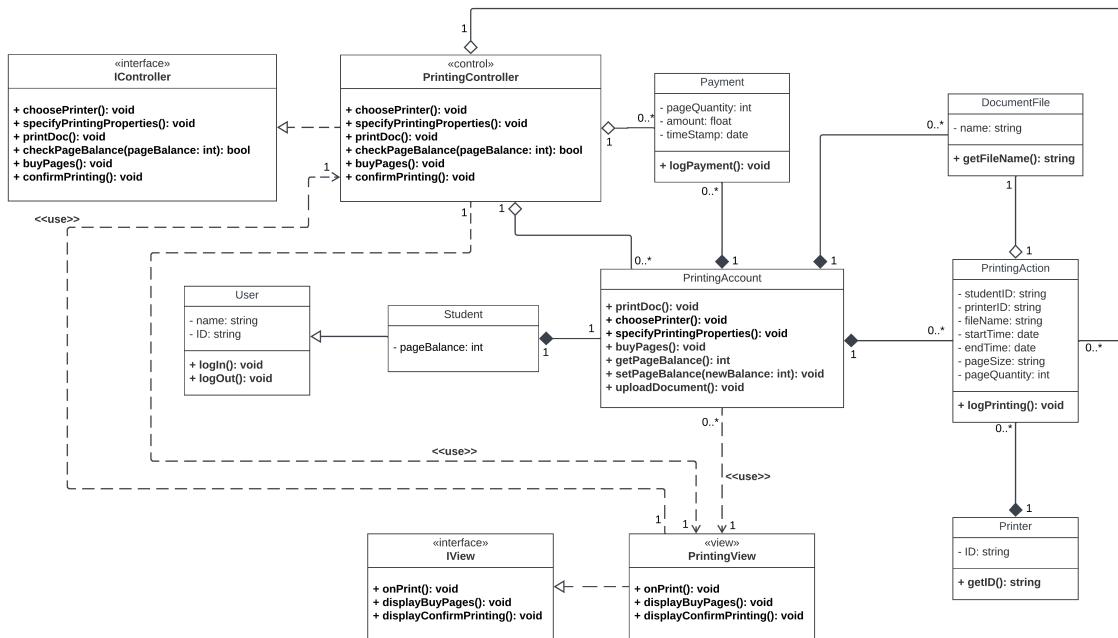
Beside checking list of printers, the **Manage Printer** module also contains 2 more optional method.

- 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.

4 Class Diagram

4.1 "Printing Document" module



Hình 5: 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:

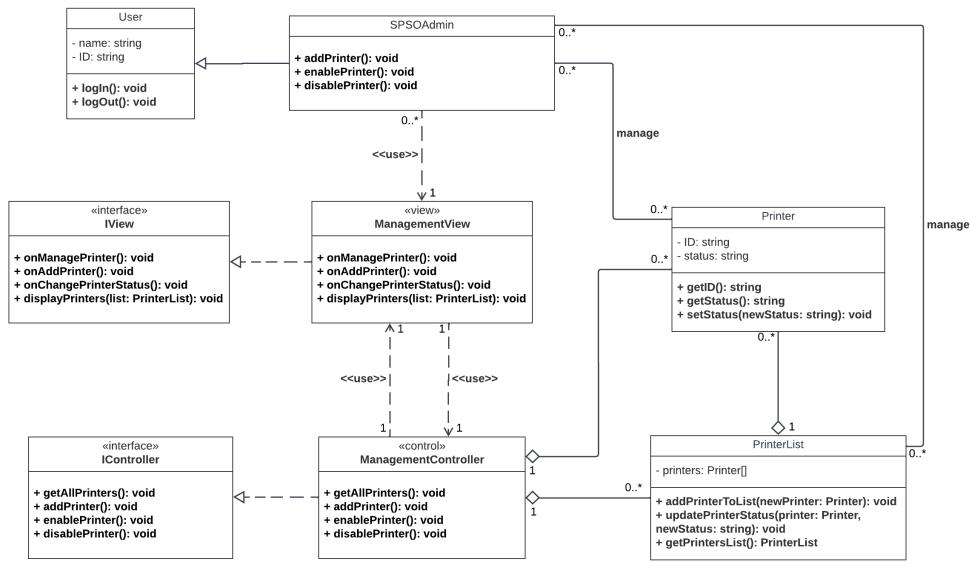
- The class **PrintingAccount** has composition relationship with the class **Student**, which means **Student** has only one **PrintingAccount**, but when **Student** is destroyed, **PrintingAccount** is also destroyed. A **Student** can have only one **PrintingAccount**, and a **PrintingAccount** can 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**.

4.2 "Manage Printer" module

Hình 6: 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 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 **SPOAdmin** class has a dependency relationship with the **ManagementView** class. **SPOAdmin** class *use* the **ManagementView** class. The multiplicity between the **SPOAdmin** class and the **ManagementView** class is denoted as many to 1, each instance of **SPOAdmin** is associated with exactly one instance of **ManagementView**, and each instance of **ManagementView** is associated with zero or more instances of **SPOAdmin**.
- 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 **SPOAdmin** class and **PrinterList** class allows the **SPOAdmin** to manage and control the status of a list of

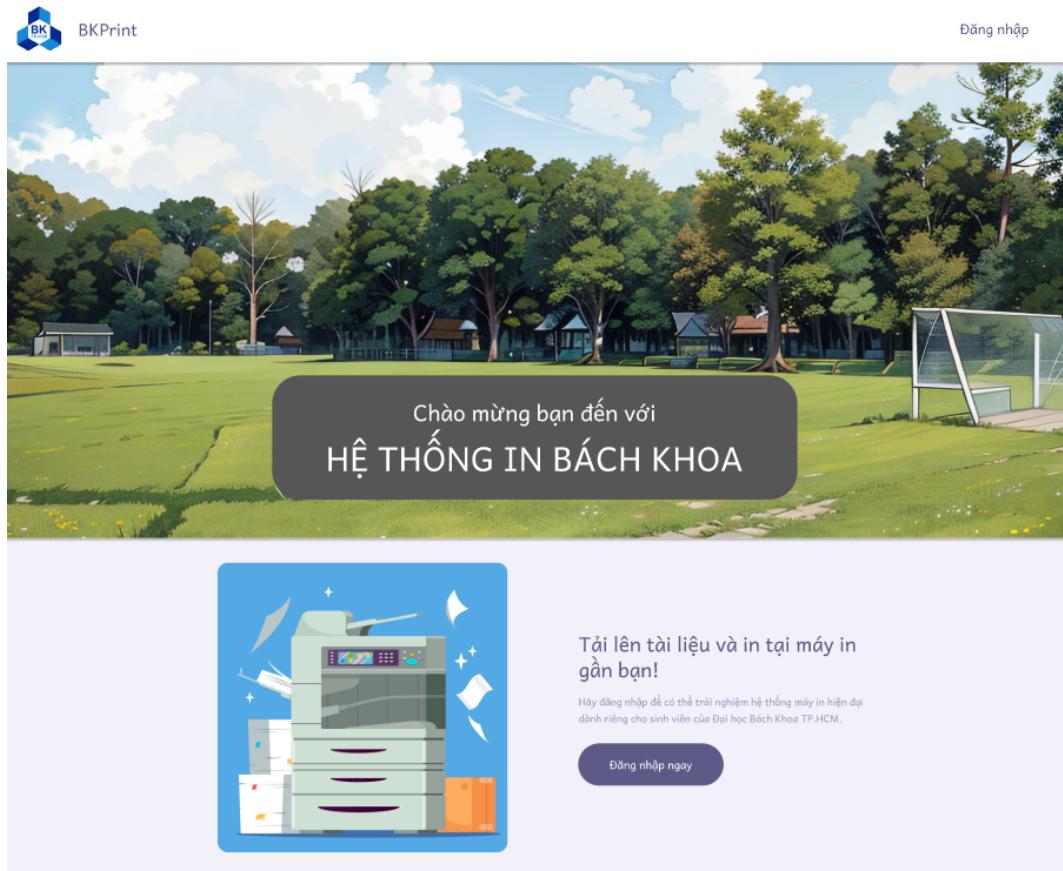
printers, as well as add new printers to the existing list and update their statuses (enable or disable) as required. The multiplicity between them is many-to-many, allowing multiple instances of the **PrinterList** class to be managed by a single instance of the **SPOAdmin** class and conversely.

- The **SPOAdmin** class is also associated with the **Printer** class, enabling the **SPOAdmin** 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 **SPOAdmin** class can be associated with multiple instances of the **Printer** class and vice versa.

5 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.

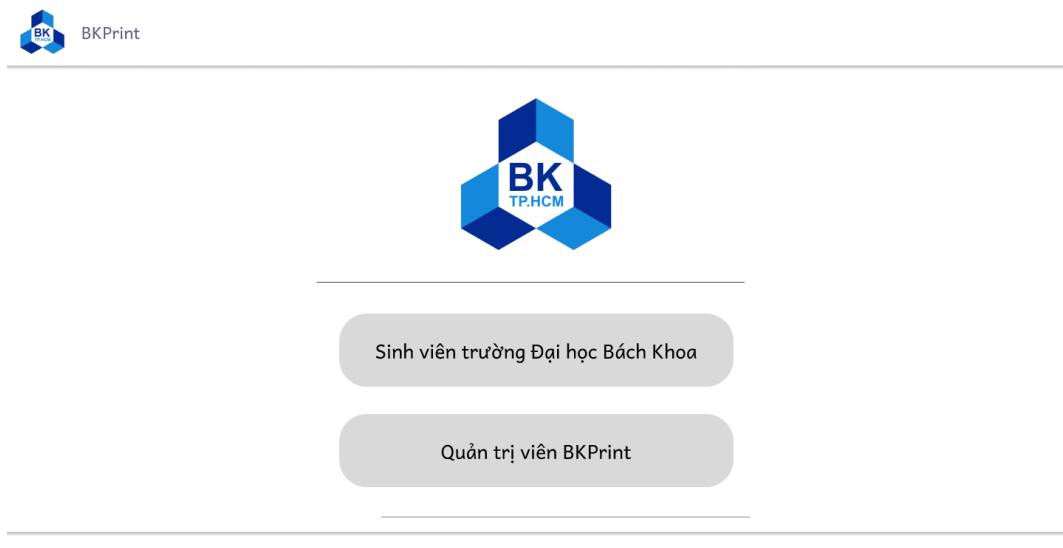
Overall, the design contains 24 frames which show the interface of the BKPrint app.



Bản quyền © 2023 Thiếu Nhi-CC02

Phát triển bởi Thiếu Nhi-CC02 | Điều khoản & điều kiện | Chính sách pháp lý

Hình 7: This is the Homepage where the user has not signed in yet.



Bản quyền © 2023 Thiếu Nhi-CC02

Phát triển bởi Thiếu Nhi-CC02 | Điều khoản & điều kiện | Chính sách pháp lý

Hình 8: After click "Đăng nhập", user has to choose whether they are students or admins



Dịch vụ xác thực tập trung

BKPrint

Bạn cần dùng tài khoản HCMUT để đăng nhập. Tài khoản HCMUT cho phép truy cập đến nhiều tài nguyên bao gồm hệ thống thông tin, thư điện tử, ...

Vì lý do an ninh, bạn hãy thoát khỏi trình duyệt Web khi bạn kết thúc việc truy cập các dịch vụ đòi hỏi xác thực!



Nhập thông tin tài khoản

Tên đăng nhập

Mật khẩu

Trợ giúp đăng nhập?

Đăng nhập

Bản quyền © 2023 Thiếu Nhi-CC02

Phát triển bởi Thiếu Nhi-CC02 | Điều khoản & điều kiện | Chính sách pháp lý

Hình 9: Student log in page



Dịch vụ xác thực tập trung

Administrator - BKPrint

Bạn cần dùng tài khoản admin để đăng nhập.



Nhập thông tin tài khoản

Tên đăng nhập

Mật khẩu

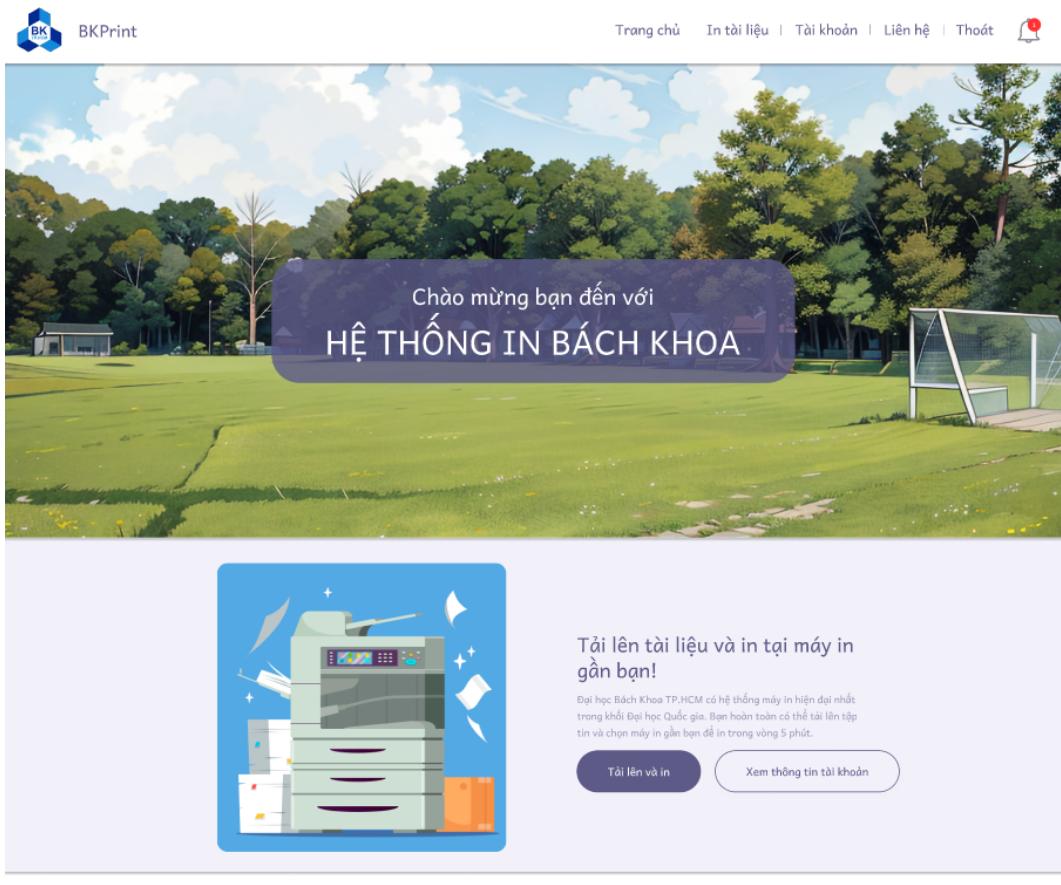
Trợ giúp đăng nhập?

Đăng nhập

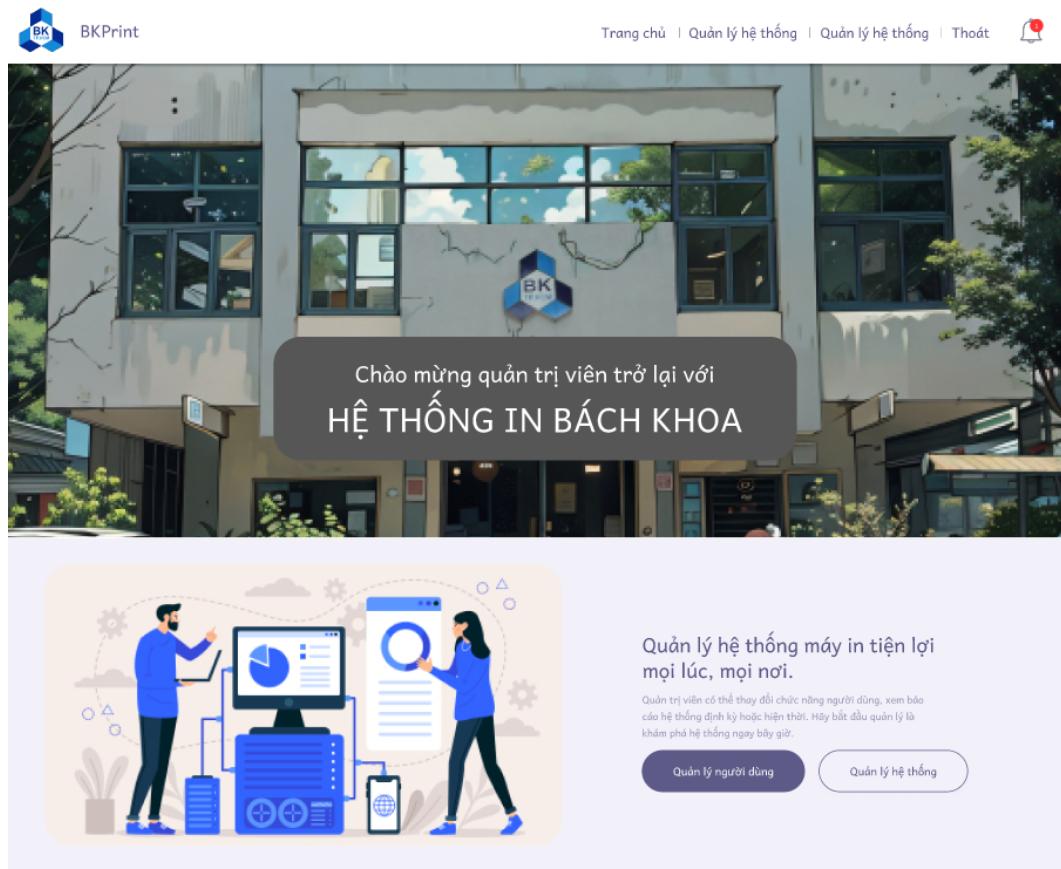
Bản quyền © 2023 Thiếu Nhi-CC02

Phát triển bởi Thiếu Nhi-CC02 | Điều khoản & điều kiện | Chính sách pháp lý

Hình 10: Admin log in page



Hình 11: Student homepage after log in



Hình 12: Admin homepage after log in

Hình 13: These are three phases of a printing process. Students are supposed to implement step by step.

The screenshot shows a "Buying Page" for BKPrint. On the left, there's a section titled "Lưu ý" (Notes) containing a bulleted list of requirements for printing. On the right, there's a "Buy" section with fields for selecting paper type (A3/A4/A5), quantity (10 sheets), and total cost (50,000đ). A large red error message at the bottom states: "Số giấy in của bạn không đủ để in. Hãy mua thêm tại đây." (The paper you have is not enough to print. Please buy more here.)

Lưu ý	Tùy chọn mua
<ul style="list-style-type: none"> Giá giấy A3: 1000đ/tờ. Giá giấy A4: 500đ/tờ. Giá giấy A5: 250đ/tờ. Số lượng mua tối thiểu: 10 tờ. Số tờ mua phải là SỐ CHẴN. Nếu giao dịch không thành công, hãy THỦ LẠI. Nếu bỏ qua bước mua thêm giấy, mọi thao tác in của bạn sẽ bị HỦY BỎ. 	Loại giấy: <input type="button" value="Chọn loại giấy"/> A3 (297 x 420)mm A4 (210 x 297)mm A5 (148 x 210)mm Số tờ: <input type="button" value="−"/> <input type="text" value="10"/> <input type="button" value="+"/> Thành tiền: 50.000đ <input type="button" value="THANH TOÁN"/>

Hình 14: If the balance of the user is not efficient (not enough paper), the interface will redirect to this "buying page".

The screenshot shows a student account page. At the top, there's a profile picture placeholder with the letter 'N' and the student's name "Tạ Ngọc Nam" and ID "2152788". Below this, there's a "Logout" link. To the right, there's a box containing the student's email "nam.ta8989@hcmut.edu.vn" and their major "Ngành học Kỹ thuật máy tính".

Lịch sử in

Từ ngày / / đến ngày / /

Thời gian	Tên file	Số tờ	Địa điểm	Trạng thái
00:00, 22/10/2023	file1.pdf	15 (A4)	A4	Đã hoàn tất
07:00, 23/10/2023	file2.pdf	20 (A5)	B1	Đã hoàn tất
08:00, 23/10/2023	file3.docx	3 (A3)	C5	Đã hoàn tất

Số tờ A3 đã in: 3
A4 đã in: 15
A5 đã in: 20

Lịch sử mua

Từ ngày / / đến ngày / /

Thời gian	Số tiền	Số tờ (A4)
09:00, 22/10/2023	50,000 VND	100
10:00, 23/10/2023	20,000 VND	40
12:00, 24/10/2023	30,000 VND	60

Số tờ còn lại: 50 (A4)

Hình 15: Student account page

Tài khoản

Tạ Ngọc Nam
Admin
Thời gian

Địa chỉ email
nam.ta8989@hcmut.edu.vn

Tổ chức
SPSO

Quản lý người dùng - Lịch sử in

Từ ngày / / đến ngày / /
Tên sinh viên:

Tên	MSSV	Thời gian	Tên file	Kiểu máy	Địa điểm	Trạng thái
Tạ Ngọc Nam	2152788	00:00, 22/10/2023	file1.pdf	Canon LBP2900	A4	Đã hoàn tất
Nguyễn Thành Thảo Nhí	2152840	07:00, 23/10/2023	file2.pdf	Canon LBP2900	A4	Đã hoàn tất
Lê Thành Bình	2112887	08:00, 23/10/2023	file3.docx	Canon LBP2900	A4	Đã hoàn tất

Quản lý hệ thống - Máy in

Từ ngày / / đến ngày / /
ID máy in:

Mã ID	Thương hiệu	Kiểu máy	Tòa nhà	Phòng	Tùy chọn	Trạng thái	Số tờ đã in (A4)
#00001	Canon	Canon LBP2900	A4	402	Đang hoạt động	Đang hoạt động	100
#00002	Epson	Epson L805	A5	106	Đang hoạt động	Đang hoạt động	500
#00003	Brother	Brother HL-L2321D	B1	200	Chờ xử lý	Đang hoạt động	700

Số tờ còn lại: 50

Tùy chỉnh

Số tờ mặc định: 20 tờ/người dùng |

Loại file được phép tải lên:

Tên loại file	Trạng thái	Tùy chọn
Excel	Cần tải lên	Cho phép
Word	Cho phép	Cần tải lên
PDF	Cho phép	Cần tải lên

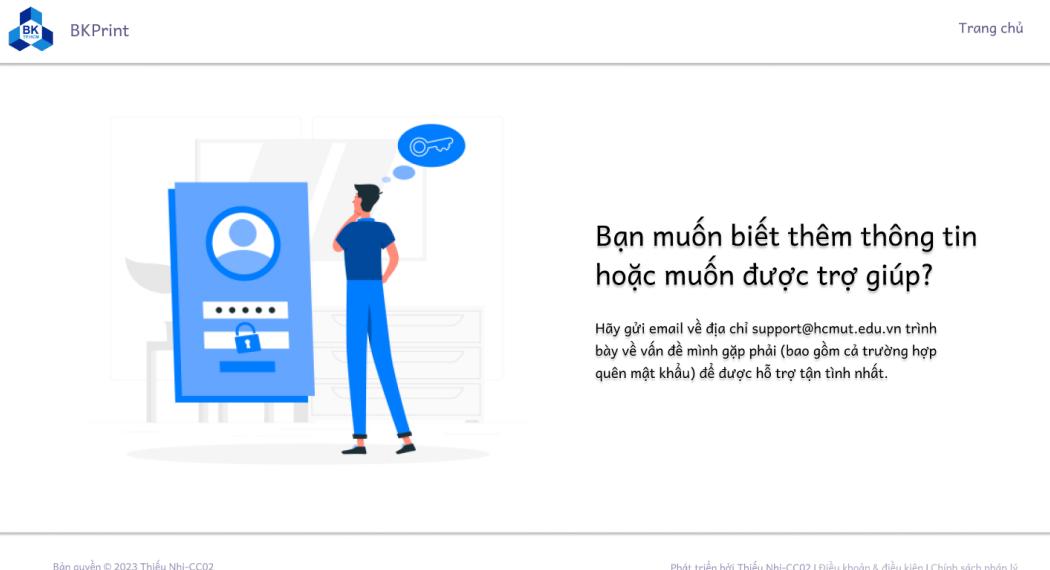
Báo cáo

Nhận báo cáo
 Hàng tháng
 Hàng năm

Nhận báo cáo tháng vào: hàng tháng
Nhận báo cáo năm vào: hàng năm

Xem kho lưu trữ báo cáo

Hình 16: Admin account page



Hình 17: 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".

6 Conclusion

This Modeling Document serves as a central repository for visual models and UI representations, providing a multi-dimensional understanding of the system. By incorporating diverse notations, diagram types, and UI wireframes, the document serves as a guiding reference for system architects, developers, and stakeholders. This comprehensive resource is essential for ensuring a smooth and efficient development process for the BKPrint system.