Software Requirements Specification for Student Smart Printing Service

Version 3.3 approved

Prepared by:

1. **Nguyen Quang Phu - 2252621**
2. **Nguyen Ngoc Khoi - 2252378**
3. **Nguyen Nhat Khoi - 2252379**
4. **Nguyen Quang Vinh - 2213973**
5. **Nguyen Minh Khoi - 2252376**

Department of Software Engineering

Faculty of Computer Science and Engineering

Ho Chi Minh City University of Technology – VNU-HCM

05/11/2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Draft Plan	19/09/2024	Draft Version	1.0
Update Domain Context	20/09/2024	Update Domain Context	1.1
Update 1.1.2 & 1.1.3	21/09/2024	Improve 1.1.2 & 1.1.3 by reconsidering stakeholders	1.2
Update 1.1.4 & 1.1.5	22/09/2024	Improve 1.4 & 1.5 by adding more requirements	1.3

Update Use case	25/09/2024	Update use case	1.4
Further Elaboration on use case	27/09/2024	Improvement	1.5
Beautify use case diagrams	30/09/2024	Improve on diagrams	1.6
Update Activity Diagrams (2.1)	04/10/2024	Refinement of activity diagrams for clarity and alignment with the latest functional requirements	2.0
Sequence Diagram Improvement (2.2)	08/10/2024	Enhance sequence diagrams for better representation of Printer Management interactions	2.1
Class Diagram Update (2.3)	13/10/2024	Update class diagrams with new attributes and methods	2.2
UI Diagram Adjustment	16/10/2024	Minor UI updates to reflect changes in workflow	2.3
Complete System Modeling Report	20/10/2024	Final modification of System Modeling part in this report	2.4
Update MVC Architecture Section	23/10/2024	Added advantages, disadvantages, and reasons for MVC	3.0

		choice over layered architecture	
Architectural and Deployment Diagrams Refinement	26/10/2024	Refined architectural and deployment diagrams for alignment with system components	3.1
Component Diagram Update	27/10/2024	Enhanced component diagrams with detailed descriptions for authentication, printing, and printer management modules	3.2
Final Review and Quality Check for Report of Task 3	30/10/2024	Final proofreading, consistency check, and format adjustments for the Architecture Design section	3.3

I. Requirement elicitation

1. Overview (Task 1.1)

1.1. Domain Context

The HCMUT Smart Printing Service for Students (HCMUT-SPSS) is designed to streamline document printing for students at Ho Chi Minh City University of Technology (HCMUT). By offering on-campus printing, students can save time, avoid travel, and conveniently access nearby printing facilities.

Currently, the limited number of print points (only 3-4 across both campuses) cannot meet the high demand, especially during midterms and finals, leading to long waiting times. Manual printer operations further limit efficiency, while restricted working hours prevent students from printing at their convenience.

In traditional off-campus printing, students face even more difficulties. The number of students far exceeds the number of print shops, resulting in long queues and extended waiting times during high-demand periods. Additionally, customizing print formats can be challenging due to miscommunication between students and shop owners. Security is also a concern, as there is no guarantee that sensitive information in the documents won't be disclosed when shared with print shops. Moreover, sending files through multiple platforms (e.g., email, Zalo, Messenger) increases the risk of errors and makes file management more complicated.

The HCMUT-SPSS resolves these issues by providing an online platform where students can upload documents, customize print settings, and submit jobs remotely. They only need to pick up their materials when ready, reducing waiting times and manual intervention. The platform's accessibility allows students to manage their printing anytime, improving control and reducing errors. This smart solution addresses key pain points while creating a more efficient and reliable printing experience.

1.2. Stakeholders and Needs

Students

Description: Students are the primary users of the system. They rely on it for printing academic and personal documents across campus using available printers. Each student has an account with a limited number of printing pages provided by the university, which they can manage through the system.

Needs:

- Upload documents and specify printing preferences (paper size, double-sided, etc.).
- Monitor and manage their page balance, and purchase more pages if needed.
- View their personal printing history and usage summary.
- Securely log in using the university's SSO system.

Student Printing Service Officer (SPSO)

Description: The SPSO manages the technical and operational aspects of the printing service. They are responsible for configuring system settings, managing printers, and overseeing system usage by students.

Needs:

- Manage printers (add, enable, disable) and system configurations (e.g., default page limits, accepted file types).
- Access and filter the printing logs of students by date or printer.
- View automated reports on system usage (monthly and yearly).
- Provide troubleshooting and technical support for system issues.

University Administration

Description: The administration oversees the system's alignment with university policies and ensures it operates smoothly to meet student needs. They are also concerned with financial and operational aspects of the service.

Needs:

- Ensure the printing service is effective and meets university policy standards.
- Monitor system efficiency and cost management (e.g., page allocation, student payments).
- Ensure compliance with privacy and data protection regulations.

BKPay Payment System Provider

Description: BKPay is the university's online payment platform integrated with HCMUT_SSPS. It allows students to purchase additional printing pages and ensures secure transactions.

Needs:

- Enable secure and seamless payment processing within the printing system.
- Maintain accurate transaction logs for both students and the system.
- Ensure payment security and prevent fraud.

Printer Manufacturers

Description: Printer manufacturers supply and maintain the printers located around the university's campuses. They ensure the hardware functions smoothly and integrates with the HCMUT_SSPS system.

Needs:

- Ensure printers are compatible with the system's requirements.
- Provide regular maintenance and troubleshooting services for the printers.

HCMUT_SSO Authentication Service

Description: HCMUT_SSO is the university's Single Sign-On authentication service that provides secure access to the HCMUT_SSPS system for both students and staff.

Needs:

- Provide secure, reliable login for all users accessing the printing system.
- Ensure smooth integration between the authentication system and HCMUT_SSPS.

1.3. Benefits of the System

Students:

- Convenient document printing with flexible options.
- Easy management of printing page balance and history.
- Secure access through HCMUT_SSO.

Student Printing Service Officer (SPSO):

- Full control over printer management and system settings.
- Access to detailed logs and automated reports.
- Streamlined oversight of system usage and student activity.
- Ensures system stability and quick troubleshooting.

University Administration:

- Ensures compliance with printing policies and privacy regulations.
- Improves cost management and operational efficiency.
- Data-driven insights from automated reports.

BKPay Payment System Provider:

- Increased transaction volume through student purchases.
- Accurate and secure payment tracking.
- Opens doors for potential future partnerships with the university for other services.

Printer Manufacturers:

- Promotes their printers as reliable and compatible with the university's system.

HCMUT_SSO Authentication Service:

- Provides secure and seamless login for all users.
- Simplified integration with the printing system.

2. Functional and Non - Functional Requirements (Task 1.2)

2.1. Functional Requirements

2.1.1. For Students:

User Registration and Authentication

- Students must be able to register and log in using their university credentials (student ID and password), ensuring only authorized users can access the service by enforcing **HCMUT Single Sign-On (SSO)** authentication.

Document Upload

- Students must be able to upload documents in various formats (PDF, DOCX, PPT, etc.). Specifying limits on formats could improve system performance.
- Batch upload functionality should be available for multiple documents to be uploaded simultaneously.

Printing Customization

- Students should have options to customize print settings, including:
  - Number of copies
  - Paper size (A4, A3, etc.)
  - Single-sided or double-sided printing

    - Color or black-and-white
    - Orientation (portrait or landscape)
    - Binding options (e.g., stapling)
- A real-time print preview should be provided to verify the job before submission. It should also offer **cost estimations** based on customization.

Payment Integration

- Students must be able to make payments through integrated options like university accounts, e-wallets, or credit/debit cards.
- The system should display an itemized cost breakdown based on the selected print settings.

Print Job Status Tracking

- Students should be able to track the progress of their print jobs, including statuses like "submitted," "in progress," "completed," and "ready for pickup."
- The system must send notifications (via email or app) when their print job is ready for pickup.

Document Security and Privacy

- Students must have the assurance that their uploaded documents are securely stored and accessible only by authorized personnel.
- Encryption should be used to ensure secure file uploads and storage.

Error Handling and Reprinting

- Students must have the option to request reprints in case of *system* errors (e.g., formatting mistakes, incomplete prints).
- The system should assist in preventing *user* errors (e.g., wrong settings) through warning messages.
- The system should facilitate free reprinting if errors are due to the system or print service, without charging students again.

Document History and Management

- Students must have access to their document history, allowing them to view previous jobs, reprint documents, or download receipts.
- The system should offer the option to automatically delete files post-printing after a certain number of days to maintain security and privacy.

2.1.2. For SPSO:

Printer Management

- The system must allow the SPSO to add, enable, disable, and manage printers across campus.

Printer Configuration

- The system must be able to modify the default number of printing pages assigned to students and configure the specific dates for distributing these page credits each semester.
- The system also has the ability to manage and update the types of file formats allowed for printing, ensuring compliance with system constraints.

System Maintenance and Troubleshooting

- The system must provide tools for SPSO staff to diagnose and fix printer issues or malfunctions quickly. Remote diagnostics could be useful for large, distributed campuses like HCMUT.
- The system should enable maintenance tasks like printer calibration and updating software without interrupting active print jobs.

Report Generation and Usage Insights

- The system must generate reports detailing print job volume, total revenue, and print service usage, allowing SPSO staff to monitor service performance. These reports can be exported in various formats (CSV, PDF) or integrated with other university systems for analytics.
- Usage statistics should provide insights for optimizing printing services, identifying trends, and ensuring resource efficiency.

Student Printing History Access

- The system must provide the SPSO with the ability to view the printing history of all students or a specific student for a selected time period.

Efficient Resource Management

- The system must provide real-time data on printer resources, such as paper levels and ink/toner status.
- Alerts should be generated when resources (paper, ink) are running low to ensure timely replenishment.

2.1.3. For University Administration:

Access to Aggregate Printing Reports

- The system must provide university administrators with access to comprehensive, aggregate reports on printing activities. This includes tracking overall usage trends, print volume per department, system performance, and identifying patterns in student printing behaviors (e.g., preferred printing times, document types).

Revenue Monitoring

- Administrators must be able to monitor the revenue generated from paid printing services, with clear financial reports. The system should offer insights into total revenue, revenue breakdown by user groups (students, faculty), and cost analysis to help inform pricing strategies and resource allocation.

University-Wide Policy Management

- The system must offer tools for administrators to set and enforce university-wide printing policies, such as:
  - **Printing quotas** for students (e.g., free prints per semester or paid prints beyond a certain limit).
  - **Printing fees** for different types of printing (e.g., black-and-white vs. color, single-sided vs. double-sided).
  - **Content restrictions**, ensuring that only permissible content can be printed, in line with university policies.

Global System Configuration

- The system must allow administrators to configure global system settings, including:
  - **Default printer page allocations** for students, staff, and faculty.
  - **Printer maintenance schedules**, ensuring that printers are regularly serviced and have minimal downtime.
  - **User access levels**, determining which users have access to certain printers or settings (e.g., high-quality or specialty printers).

Peak Time Reporting and Optimization

- The system must provide detailed reporting on peak printing times (e.g., during exams or project submission periods) to help optimize printer distribution and resource allocation across campus. This could include:

    - Data on which printers are used most frequently.
    - Recommendations for redistributing resources or adding printers in high-demand areas.
    - Insights for adjusting maintenance schedules during non-peak times to avoid disruptions.
    - Dynamic resource allocation (moving jobs to available printers) to further improve overall performance.

2.1.4. For HCMUT_SSO Authentication Service:

- The system must be integrated with the HCMUT_SSO authentication system.
- All user login operations must go through the HCMUT_SSO authentication system.
- The system should offer multi-factor authentication for added security.

2.1.5. For BKPay Payment System Provider :

- The system must integrate with at least one electronic payment method.
- The system must send a payment receipt to students via email.
- Transaction logging/history is available.

2.2. Non - Functional Requirements

2.2.1. Security

- Account recovery options such as password resets should also be addressed.
- Sensitive data, such as payment transactions and personal student information, must be encrypted both in transit and at rest to ensure confidentiality and data integrity.

2.2.2. Reliability

- The system should guarantee high uptime (e.g., 99.9% availability) with minimal downtime for maintenance or updates.
- Maintenance (downtime) should be scheduled during non-peak hours to avoid disruptions, ensuring the system is available during critical periods such as exam

seasons, thesis submissions, or other peak student usage times. It is handled through notice periods or compensation.

2.2.3. Usability

- The system must feature a user-friendly interface on both web and mobile platforms, allowing for easy navigation and functionality.
- The system should cater to users of varying technical proficiency, providing clear instructions, intuitive design, and support for common tasks such as document uploading, print customization, and payment processing.
- Predictive analytics for HCMUT SPSO is included to anticipate when supplies will run out.
- Refund policies and transaction failure handling should be implemented for smoother user experience.

2.2.4. Compatibility

- The system should be compatible with a wide range of devices and operating systems, allowing students to access the service from laptops, desktops, smartphones, and tablets.
- It must support modern web browsers (e.g., Chrome, Firefox, Safari) and mobile operating systems (iOS and Android), ensuring that students can print from different platforms with consistent performance.

2.2.5. Scalability

- The system must be capable of scaling to accommodate increased user load, particularly during peak periods like exam times or major academic events (e.g., thesis submissions, project deadlines), without performance degradation or delays. Automatically prioritize print jobs based on urgency.
- It should dynamically adjust to handle a high number of concurrent users and large file uploads to maintain system efficiency and user satisfaction.

3. Use-case Diagrams and Scenarios (Task 1.3)

3.1. Use-case table

ID	Use case Name	Description
UC001	Login	The user login to the system
UC002	Print documents	The user uses the printing service
UC003	Buy printing pages	The user buys more printing pages
UC004	Document upload	The user uploads a document
UC005	Customize printing options	The user configuring the printing options
UC006	Track printing status	The user tracks the printing status of the documents
UC007	Manage printer	SPSO manages the printer
UC008	Add printer	SPSO adds a printer
UC009	Enable printer	SPSO enables a printer
UC010	Disable printer	SPSO disables a printer

3.2. Use-case Diagram for the Whole System

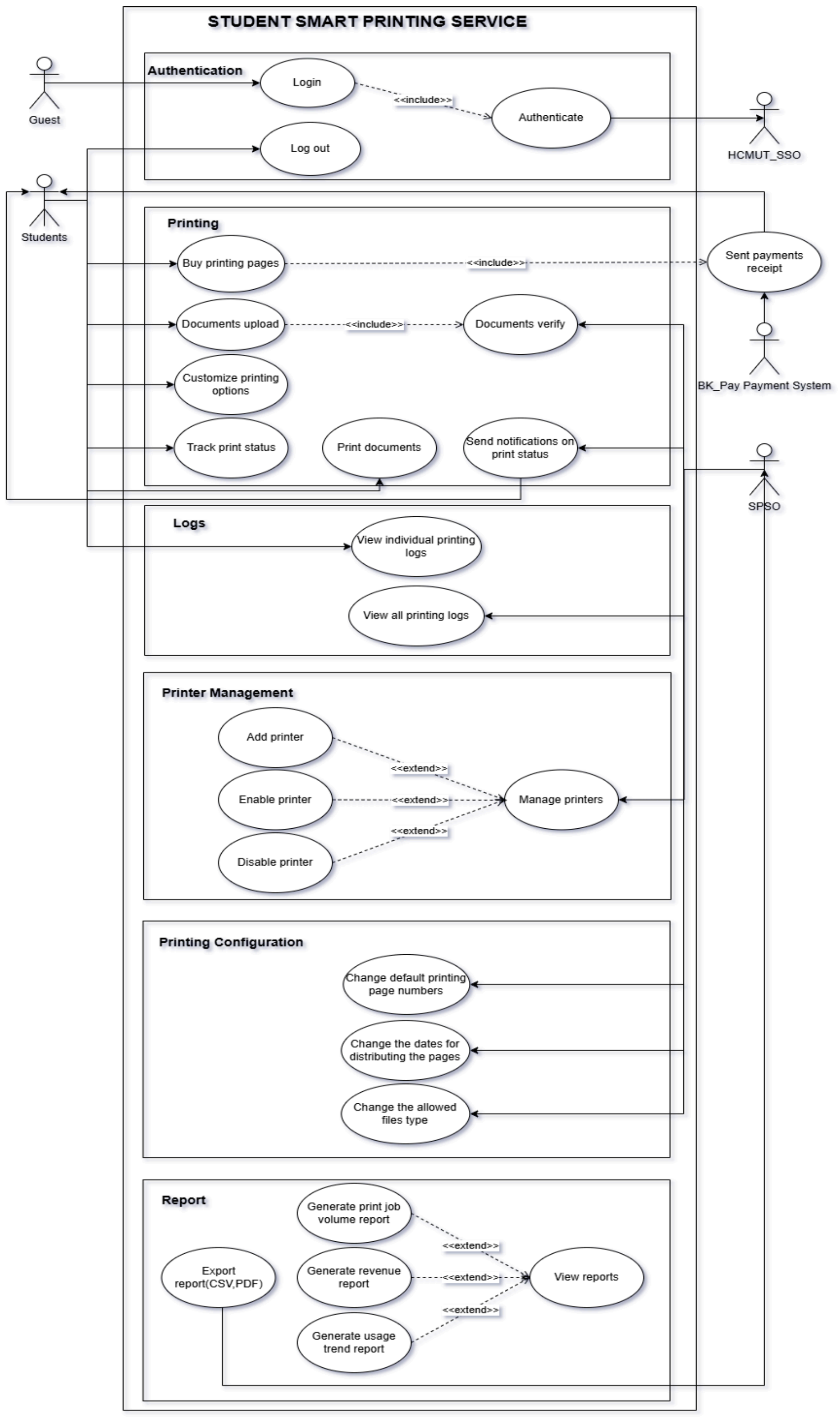


Figure 1: Use case diagram for the whole system

3.3. Module: Authentication

3.3.1. The Use Case Diagram in Authentication Module

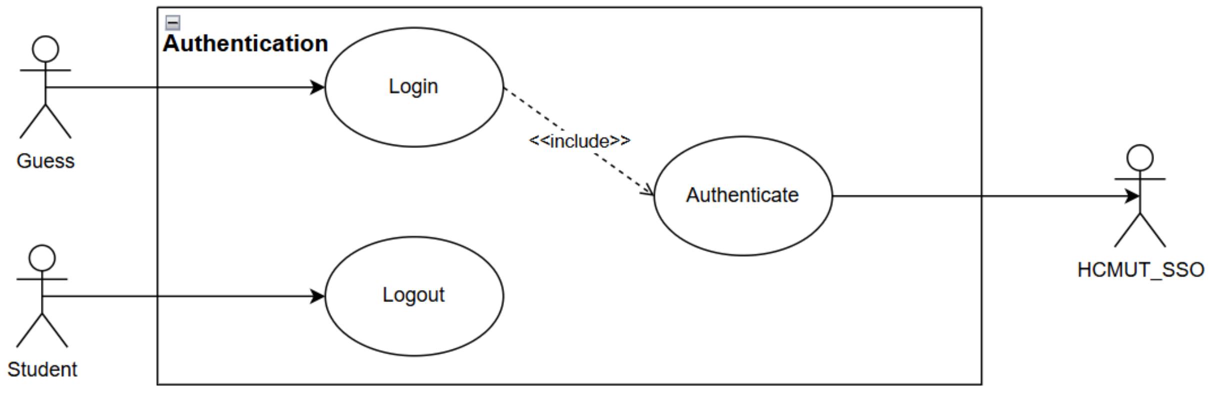


Figure 2: Use case diagram for the Authentication module

3.3.2. The Use Case Scenario: Authentication/Login

Use Case ID	**UC001**		
Use case name:	**Login**		
Created by:	Nguyen Quang Vinh	**Lasted updated by:**	Nguyen Quang Phu
Dated created:	25/09/2024	**Dated last updated:**	30/09/2024
Actors:	Guest (Unauthenticated User), Student (Authenticated User), HCMUT_SSO (Single Sign-On System)		
Description:	This use case allows an unauthenticated guest to log into the system using their HCMUT_SSO credentials. Users can log in with their accounts to access the system's features.		
Trigger:	The user (guess or student) initiates the login process by clicking the "Login" button on the main interface of the website or application.		
Preconditions:	- The guest has not logged into the system.		

	- The user possesses valid credentials for HCMUT_SSO. - The login account has been assigned student permissions. - The user's device is connected to the internet. - The system is online and available for login attempts.
Postconditions:	- The user is successfully authenticated. - The system has recorded the successful application login. - The system has created a session for the user. - The user gains access to their personalized features based on their role (guest or student).
Normal Flows:	1. The user accesses the application (web). 2. The user selects the login method using an account. 3. The system prompts the user to enter their username and password. 4. The user enters the username & password, then clicks the login button. 5. The system successfully verifies the login information and grants the user access to the application. 6. The system logs the successful application login activity. 7. The system updates the interface according to the information of the Guest account.
Alternative Flows:	**A1: At step 4** 4.1. The user selects the option to save the login information before clicking the login button, and the system automatically saves the username and password for future logins. 4.2. Return to step 5.
Exceptions:	**E1: At step 6** ***6.1. Invalid Credentials*** If the user enters an invalid username or password: - The system displays a generic error message (e.g., "Invalid username or

	password"). + This avoids revealing whether the username or password was incorrect for security reasons. ***6.2. Options for the User*** The user has the following options: - <u>Retry</u>: They can go back and re-enter their username and password. The system returns to Step 3 in the normal flow. - <u>Forgot Password</u>: The system should provide a link for users to reset their password if they've forgotten it. - <u>Cancel</u>: The user can cancel the login process and return to the main page.
Note and issues:	**Note**: Implement strong session management, enforce 2FA, allow users to monitor active sessions, and notify users of new logins. **Issues**: Multiple active sessions could increase the chance of unauthorized access if one device is compromised.

3.4. Module: Printing

3.4.1. The Use Case Diagram in Printing Module

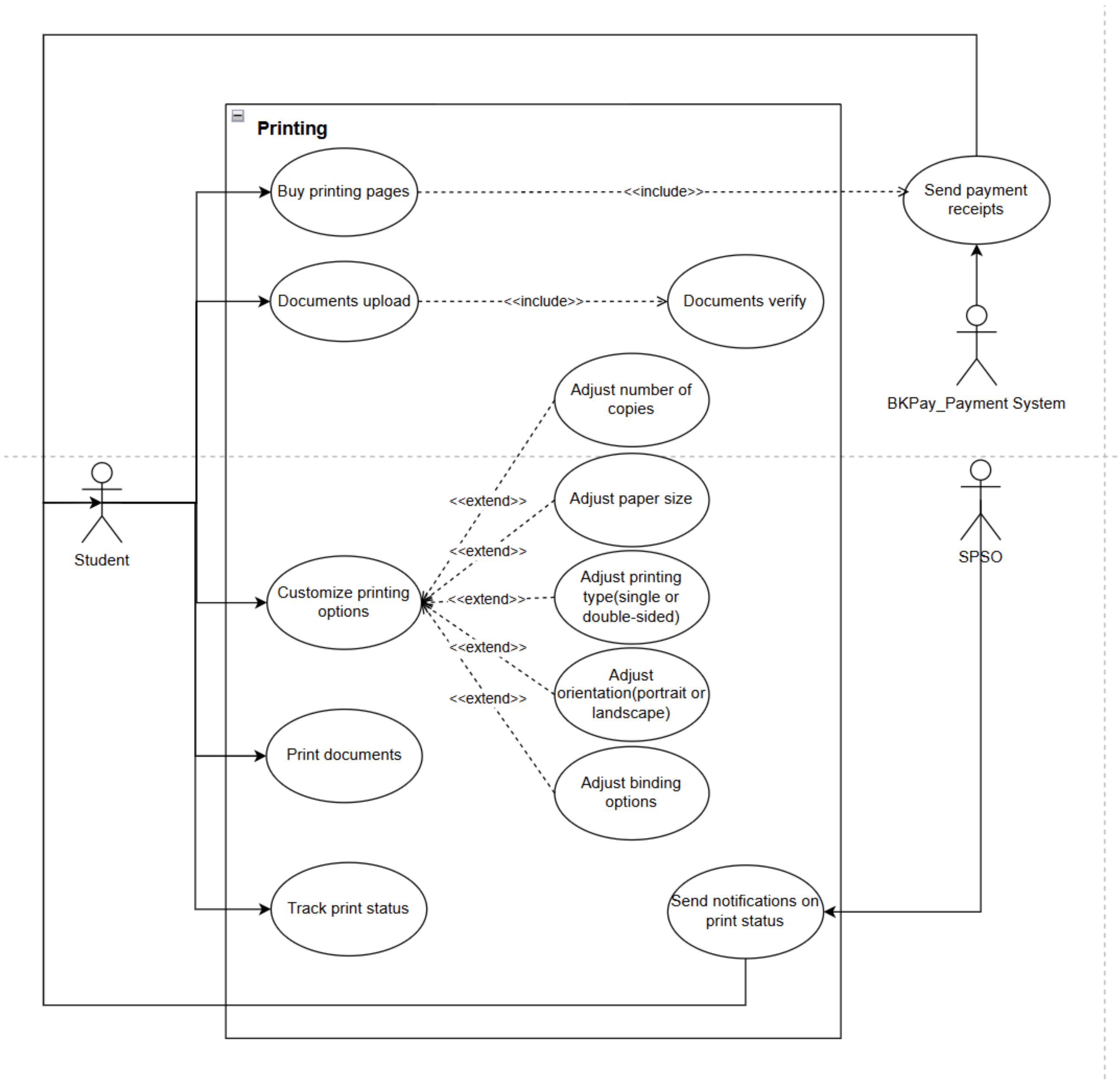


Figure 3: Use case diagram for the Printing module

3.4.2. The Use Case Scenario: Printing Documents

Use Case ID	**UC002**
Use case name:	**Print documents**

Created by:	Nguyen Quang Vinh	**Lasted updated by:**	Nguyen Quang Phu
Dated created:	25/09/2024	**Dated last updated:**	30/09/2024
Actors:	Student		
Description:	This use case allows users to upload documents and print them after approval.		
Trigger:	The user clicks the "Print Now" button on the navigation bar.		
Preconditions:	- The user must have an account on the system. - The document must be successfully uploaded and approved. - The user must have sufficient print credits. - The user selects the time, location, and printing configurations before submitting the print request.		
Postconditions:	The user will be placed in the print queue for the selected printer.		
Normal Flows:	1. The user clicks "Print Now" on the navigation bar. 2. The system displays the interface for selecting a printer (building, floor, room). 3. The system filters and displays available printers with details like location and number of pending print jobs. 4. The user selects an available printer based on capacity and preferences. The system allocates the job to the selected printer. 5. The user uploads the file for printing, if not done previously. 6. The system begins to verify the uploaded file for formatting and print requirements. 7. The user selects the time and method for printing (e.g., single-sided or double-sided). 8. The user confirms and submits the print job.		
Alternative Flows:	**A1: At Step 1:**		

	1.1: If the user is not logged into the system, the system redirects them to the login page. 1.2: The user enters their credentials and logs in. 1.3: Continue with Step 2 in the Normal Flow. **A2: At Step 4:** 4.1: If no printers are available (e.g., under maintenance), the system notifies the user of the unavailability. 4.2: The user is notified to select another printer or return later.
Exceptions:	**E1: At Step 4:** - If all printers do not meet the filtering criteria due to maintenance or other reasons: + The system notifies the user. **E2: At Step 7:** - The system detects that the uploaded file does not meet the verification criteria (file format, content issues, etc.): + The system notifies the user of the issue and requests a re-upload. **E3: At Step 8:** - The system detects that the user's remaining print credits are insufficient for the print job: + The system notifies the user of the issue and redirects them to the "Buy Print Credits" use case to purchase additional credits.
Note and issues:	No notes or issues identified.

3.4.3. The Use Case Scenario: Buy printing pages

Use Case ID	**UC003**		
Use case name:	**Buy printing pages**		
Created by:	Nguyen Quang Vinh	**Lasted updated by:**	Nguyen Quang Phu
Dated created:	26/09/2024	**Dated last updated:**	30/09/2024
Actors:	Student		
Description:	Allows the user to purchase additional print pages for their account through the integrated BK Pay payment system. The purchased pages will be available for printing.		
Trigger:	The user clicks the "Buy More Pages" button in the Printing Module interface.		
Preconditions:	- The user is logged into the system. - The user's account is linked with BK Pay. - The user has selected a printer or a print job requiring additional pages.		
Postconditions:	- The user's print page balance is increased, and a payment receipt is sent to both the system and the BK Pay Payment System. - The user can now proceed with printing.		
Normal Flows:	1. The user accesses the Printing Module and clicks the "Buy More Pages" button. 2. The system checks whether the user has a linked BK Pay account. 3. The system displays the Buy More Pages interface, allowing the user to input the desired number of pages for purchase. 4. The user selects the number of pages to purchase and confirms the payment via BK Pay.		

	5. The system processes the payment and sends a receipt to the user's account and to the BK Pay Payment System. 6. Once the payment is confirmed, the system updates the user's print page balance and redirects them back to the printing options.
Alternative Flows:	**A1: BK Pay Account Not Linked:** - If the user does not have a linked BK Pay account: + The system prompts the user to link their account to BK Pay. + The user follows the instructions to link the payment method and is then redirected back to complete the transaction.
Exceptions:	**E1: Payment Failure:** - If the payment fails due to insufficient funds, expired account details, or a system error: + The system notifies the user of the payment failure and provides the option to retry the payment. + The system maintains the print job in the queue but does not proceed until sufficient funds are available in the BK Pay account. **E2: System Unavailability:** - If BK Pay is temporarily unavailable: + The system notifies the user that the payment cannot be processed at the moment. + The user is asked to try again later when BK Pay is back online.
Note and issues:	- The system should ensure that sensitive payment information is securely processed using BK Pay. - The user should receive a notification if their print job is delayed due to lack of pages or failed payment. - A clear record of page purchases and payment history should be available in the user's account for tracking purposes.

3.4.4. The Use Case Scenario: Document upload

Use Case ID	**UC004**		
Use case name:	**Document upload**		
Created by:	Nguyen Quang Vinh	**Lasted updated by:**	Nguyen Quang Phu
Dated created:	26/09/2024	**Dated last updated:**	30/09/2024
Actors:	Student		
Description:	Allows the user to upload documents for printing through the system. The uploaded document will be verified before proceeding to print.		
Trigger:	The user clicks the "Upload Document" button in the Printing Module interface.		
Preconditions:	- The user is logged into the system. - The user has access to a document ready for upload. - The document meets the acceptable file format criteria.		
Postconditions:	- The document is uploaded and verified for printing. - The user can proceed to customize printing options once the document is verified.		
Normal Flows:	1. The user clicks "Upload Document" in the Printing Module. 2. The system opens a file selection dialog for the user to choose a document. 3. The user selects a file and confirms the upload. 4. The system verifies the document format, size, and content. 5. If the document is verified successfully, the system confirms the upload. 6. The user can now proceed to customize printing options.		

Alternative Flows:	**A1: Unsupported File Format:** - If the uploaded document is in an unsupported format: + The system notifies the user and suggests supported formats (e.g., PDF, DOCX). + The user is prompted to upload a compatible document. **A2: Document Too Large:** - If the document exceeds the allowed file size: + The system notifies the user of the size limit and prompts them to upload a smaller document.
Exceptions:	**E1: Upload Failure:** - If the upload process fails due to network or system issues: + The system displays an error message and suggests retrying the upload or checking the network connection.
Note and issues:	- The system should allow the user to see a preview of the uploaded document to ensure accuracy. - The document verification process should be quick to avoid user delays. - Any failed uploads should be logged, and the user should receive feedback on why the upload was unsuccessful.

3.4.5. The Use Case Scenario: Customize printing options

Use Case ID	**UC005**		
Use case name:	**Customize printing options**		
Created by:	Nguyen Quang Vinh	**Lasted updated by:**	Nguyen Quang Phu
Dated created:	26/09/2024	**Dated last updated:**	30/09/2024
Actors:	Student		

Description:	Allows the user to adjust printing preferences such as paper size, number of copies, and binding options before submitting the document for printing.
Trigger:	The user clicks the "Customize Printing Options" button after uploading a document in the Printing Module.
Preconditions:	- The user is logged into the system. - The document has been successfully uploaded and verified. - The user has available print pages for the selected options.
Postconditions:	The user's printing preferences are saved, and the document is ready for printing.
Normal Flows:	1. The user selects "Customize Printing Options" after uploading a document. 2. The system displays a set of available printing options, including: - Adjust the number of copies. - Select paper size (A4, A3, etc.). - Choose the printing type (single or double-sided). - Set page orientation (portrait or landscape). - Adjust binding options (if available). 3. The user selects the desired options for their print job. 4. The system calculates the total number of print pages and verifies that the user has enough pages available. 5. The user confirms the settings and proceeds to print or save the configuration for later use.
Alternative Flows:	**A1: Insufficient Print Pages:** - If the user does not have enough print pages to accommodate the selected options: + The system notifies the user of the shortage and provides the option to purchase more pages via BK Pay.

Exceptions:	**E1: System Error During Customization:** - If a system error occurs while the user is selecting or confirming options: + The system notifies the user of the issue and suggests retrying the customization or saving the settings for later.
Note and issues:	- The system should provide a real-time cost estimate based on the selected printing options to help users make informed choices. - Ensure that the user is notified if any selected options (e.g., paper size or binding) are unavailable at the selected printer. - The system should allow users to save their preferred printing configurations for future use.

3.4.6. The Use Case Scenario: Track print status

Use Case ID	**UC006**		
Use case name:	**Track printing status**		
Created by:	Nguyen Quang Vinh	**Lasted updated by:**	Nguyen Quang Phu
Dated created:	26/09/2024	**Dated last updated:**	30/09/2024
Actors:	Student		
Description:	Allows the user to track the status of their submitted print job in real-time through the system. This helps the user stay informed of their print job's progress.		
Trigger:	The user clicks the "Track Print Status" option after submitting a print job in the Printing Module interface.		
Preconditions:	- The user is logged into the system. - The user has submitted a print job for printing.		

	- The system is connected to the printers to receive real-time updates.
Postconditions:	- The user can view the current status of their print job (e.g., in queue, printing, completed). - The user receives a notification when the print job is completed.
Normal Flows:	1. The user submits a document for printing and selects the "Track Print Status" option. 2. The system retrieves the current status of the print job from the printer. 3. The system displays the print job's status, which can include: - <u>In Queue</u>: The print job is waiting in line. - <u>Printing</u>: The document is currently being printed. - <u>Completed</u>: The print job has finished. - <u>Error</u>: There is an issue with the print job (e.g., paper jam, insufficient ink). 4. The user can choose to receive notifications on the progress of the print job. 5. Once the print job is completed, the system sends a notification to the user.
Alternative Flows:	**A1: Print Job Delay:** - If the print job is delayed due to high volume or printer maintenance: + The system notifies the user of the expected wait time or delay. + The user can choose to cancel the job or continue waiting.
Exceptions:	**E1: Printer Error:** - If a printer error occurs (e.g., paper jam, ink error): + The system notifies the user of the issue and suggests possible solutions (e.g., contact SPSO for support). + The print job status is updated accordingly (e.g., paused or canceled). **E2: Network/Connection Failure:**

	- If the system fails to retrieve the print status due to a network issue: + The system notifies the user and suggests trying again later.
Note and issues:	- The user should be able to track multiple print jobs simultaneously if they have submitted more than one. - Notifications should be timely and accurate to avoid confusion. - Ensure that the user can view detailed error messages if the print job fails, and provide actionable steps for resolution.

3.5. Module: Printer Management

3.5.1. The Use Case Diagram in Printer Management Module

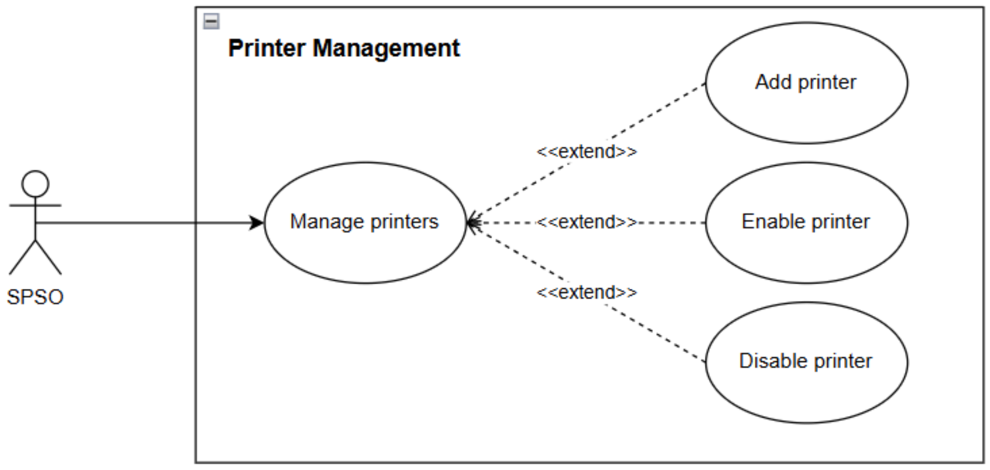


Figure 4: Use case diagram for the Printing Configuration module

3.5.2. The Use Case Scenario: Manage printers

Use Case ID	**UC007**		
Use case name:	**Manage printers**		
Created by:	Nguyen Nhat Khoi	**Lasted updated by:**	Nguyen Quang Phu
Dated created:	25/09/2024	**Dated last updated:**	30/09/2024
Actors:	SPSO		
Description:	Allows SPSO to manage printers around the campus		
Trigger:	The user clicks on the “Manage printers” on the navigation bar		
Preconditions:	- SPSO must be authenticated or authorized within the system - SPSO already login to the website - SPSO’s device connected to the internet		
Postconditions:	The printer management interface pops up for SPSO.		
Normal Flows:	1. SPSO chooses “Manage printers” on the navigation bar. 2. The printer management interface pops up with the list of all the printers around the campus and the options to add, enable and disable a printer.		
Alternative Flows:	None		
Exceptions:	None		
Note and issues:	No notes or issues identified.		

3.5.3. The Use Case Scenario: Add Printer

Use Case ID	**UC008**		
Use case name:	**Add printer**		
Created by:	Nguyen Nhat Khoi	**Lasted updated by:**	Nguyen Quang Phu
Dated created:	25/09/2024	**Dated last updated:**	30/09/2024
Actors:	SPSO		
Description:	Allows SPSO to add an additional printer to the system		
Trigger:	The user clicks on the “Add printer” printer management interface		
Preconditions:	- SPSO must be authenticated or authorized within the system - SPSO already login to the website - SPSO’s device connected to the internet		
Postconditions:	The data of the new printer is added to the system database and the list of printers		
Normal Flows:	1. SPSO choose the option “Add printer” on the printer management interface 2. The system display the information of the new printer and SPSO fill in the new printer’s details 3. The system asked for confirmation 4. SPSO confirms the operation 5. The system updates the changes to the database		
Alternative Flows:	None		
Exceptions:	**E1: At step 5**		

	- SPSO does not confirm the operation - The system does not save the changes that has been made by the SPSO
Note and issues:	No notes or issues identified.

3.5.4. The Use Case Scenario: Enable printer

Use Case ID	**UC009**		
Use case name:	**Enable printer**		
Created by:	Nguyen Nhat Khoi	**Lasted updated by:**	Nguyen Quang Phu
Dated created:	25/09/2024	**Dated last updated:**	30/09/2024
Actors:	SPSO		
Description:	Allows SPSO to enable a printer		
Trigger:	The user clicks on the “Enable printer” printer management interface		
Preconditions:	- SPSO must be authenticated or authorized within the system - SPSO already login to the website - SPSO’s device connected to the internet		
Postconditions:	The data of the printer is updated to the system database and the list of printers		
Normal Flows:	1. SPSO choose the option “Enable printer” on the printer management interface 2. The system asked for confirmation 3. SPSO confirms the operation		

	4. The system updates the changes to the database
Alternative Flows:	None
Exceptions:	**E1: At step 3** - SPSO does not confirm the operation - The system does not save the changes that has been made by the SPSO
Note and issues:	No notes or issues identified.

3.5.5. The Use Case Scenario: Disable printer

Use Case ID	**UC010**		
Use case name:	**Disable printer**		
Created by:	Nguyen Quang Vinh	Last updated by:	Nguyen Quang Phu
Dated created:	26/09/2024	Date last updated:	30/09/2024
Actors:	SPSO		
Description	Allows SPSO to disable a printer, making it temporarily unavailable for users via the printer management interface.		
Trigger:	The SPSO clicks the "Disable" button on a printer in the printer management interface.		
Preconditions:	- SPSO has a valid account with appropriate permissions. - SPSO has successfully logged into the system. - The SPSO's device is connected to the internet. - The printer is registered and online.		
Postconditions:	The printer is marked as disabled in the system, and it is no longer available for print jobs until re-enabled.		

Normal Flows:	1. The SPSO selects "Disable" for a specific printer. 2. The system prompts the SPSO for confirmation. 3. The SPSO confirms the action. 4. The system updates the printer's status in the database, marking it as disabled. 5. The printer is removed from the list of available devices in the user interface.
Alternative Flows:	**A1: SPSO Cancels Action:** - The SPSO chooses not to confirm the action. - The system cancels the disable request, and the printer remains active and available.
Exceptions:	**E1: Network or Database Failure:** - If the system fails to update the printer status due to a network or database error: + The system displays an error message to the SPSO and logs the failure. + The printer’s status remains unchanged.
Note and issues:	No notes or issues identified.

II. System Modeling

1. Activity Diagrams (Task 2.1)

1.1. Module: Authentication

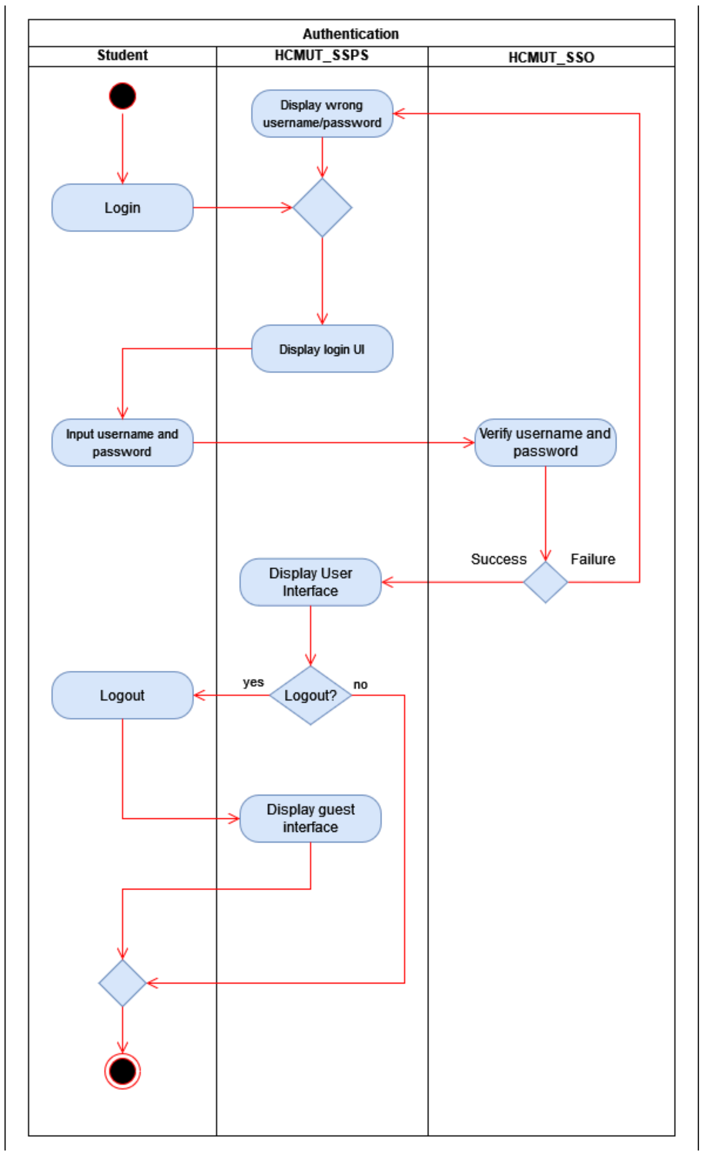


Figure 5: Activity diagram for authentication

Description:

First, the Student (initially as Guest) clicks on the "*Login*" button, then the system will display the login interface. Here, the Student can enter their username and password, then the system will verify these to see whether they are valid or not. If so, the system will display the User Interface to Student, otherwise it'll display the wrong username/password message and return to the login interface. Next if the user wants to logout via the "*Logout*" button, the system will display the Guest interface.

1.2. Module: Printing

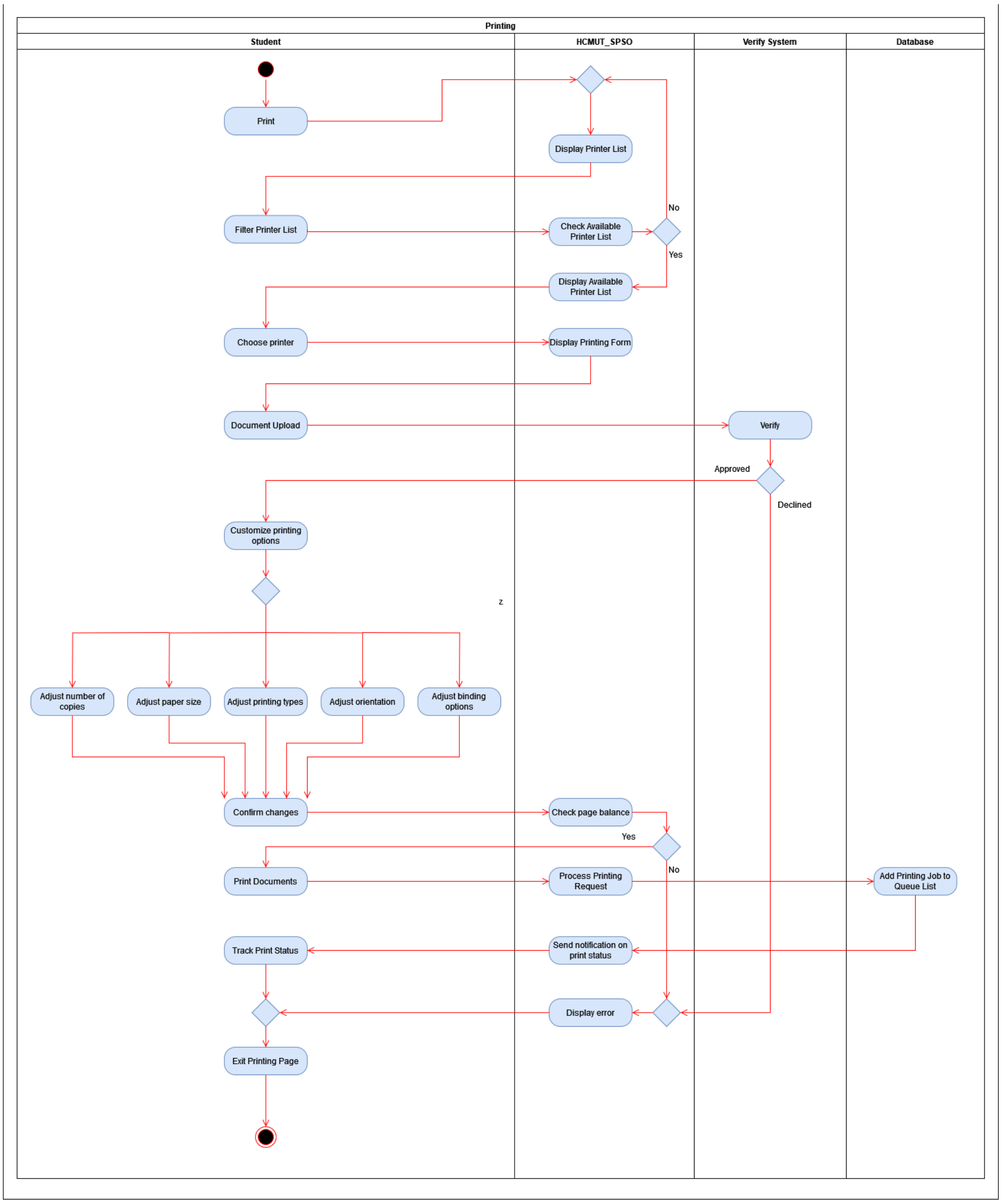


Figure 6: Activity Diagram for Printing module

Description:

First, the Student chooses the "*Print*" option on the navigation menu, then the system will display the list of printers. After that the Student will filter the printer list by filtering the facility, building and room.

The system will check if there are any printers (not under maintenance) that satisfy these criteria. If there are no available printers satisfying the criteria, the system will return to displaying the full printer list, otherwise the system will display the list of printers that satisfy the criteria.

Here, the Student will choose a printer, then the system will display a printing form and the Student will upload the document file needed to print into the system. The system will then verify if the document is valid or not. If not, the system will display an error message onto the screen. Otherwise, the Student will customize the printing options by adjusting the number of copies to print, adjusting the paper size, adjusting the printing type (single or double sided), adjusting the orientation of the document (portrait or landscape), and adjusting the binding options, and confirm the changes..

Then the system will check if the number of pages Student needed to print is enough or not. If not, it'll display the error message to Student, and Student must either reduce the number of pages or buy extra pages to print. Otherwise, the system will add the Student's printing request to the queue list. Once the printing is done, the system will send a notification on print status to Student, or Student can track the print status whenever they want.

1.3. Submodule: Buying pages

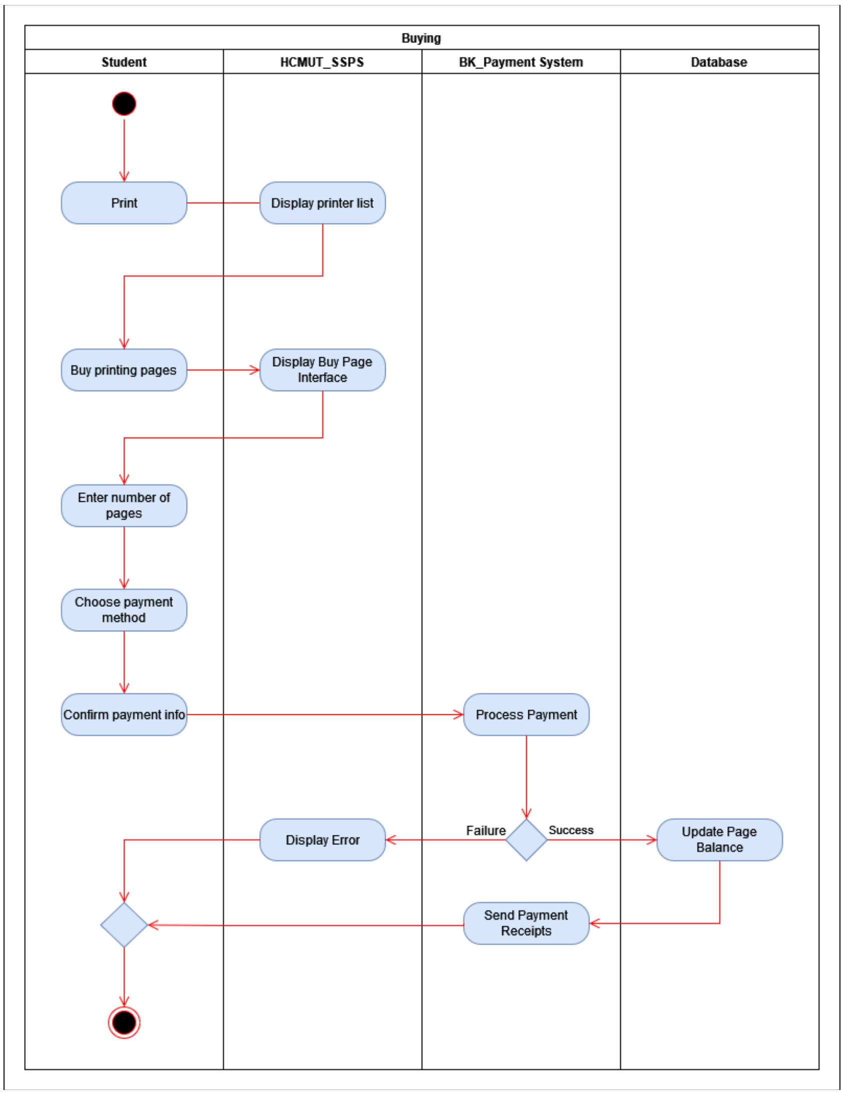


Figure 7: Activity Diagram for Buying pages

Description:

At the "*print*" interface, the Student chooses "*Buy printing pages*", the system will display the buy printing pages interface. Next, the Student will enter the number of pages needed to purchase and choose the payment method.

Afterwards, the payment system will process the payment, if unsuccessful, the system will display an error message onto the screen, otherwise, Student's page balance will be updated.

1.4. Module: Printer Management

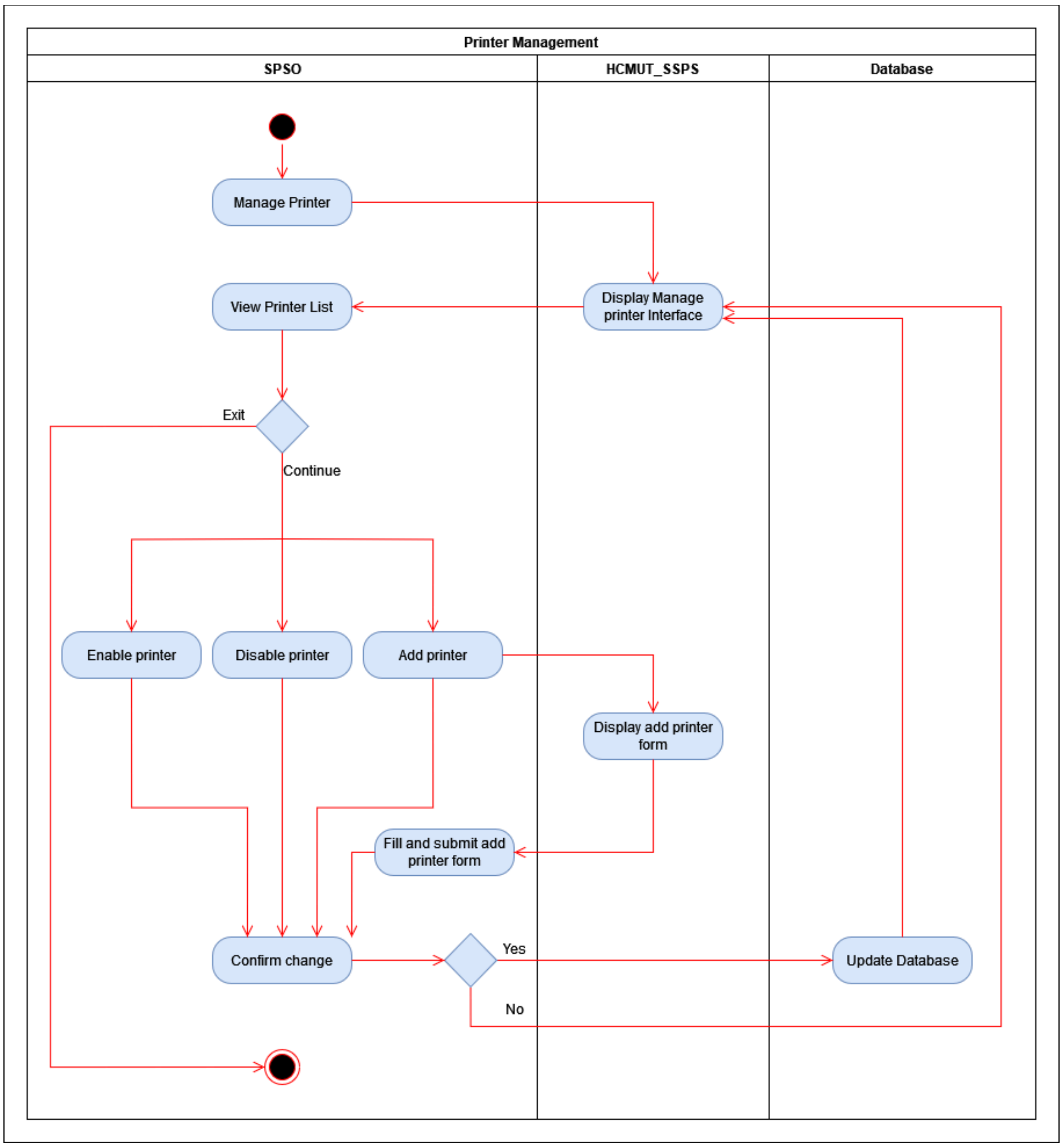


Figure 8: Activity Diagram for Printer Management

Description:

First, SPSO chooses "*Manage Printer*", the system will display the "*Manage Printer*" interface and SPSO will be able to view the list of printers.

Here, SPSO can choose "*Enable Printer*" to license a printer, "*Disable Printer*" to disable a printer, or add a printer via "*Add Printer*". If SPSO chooses "*Add Printer*", the system will display the new printer form that SPSO must fill in and save. After the above tasks, SPSO must confirm the changes, if yes then the database will be updated and return to displaying the list of printers, otherwise it'll discard the changes and return to displaying the printer list.

Besides that, SPSO can also choose "*View printer*" to see the information of a printer. Then, the system will display the information of that printer, if SPSO chooses "*Back*" the system will return to displaying the printer list. All options return to the display printer list interface that the user can see. If the user wants to continue with changing the printers, they may, otherwise the operation ends.

2. Sequence Diagram (Task 2.2)

2.1. Module: Authentication

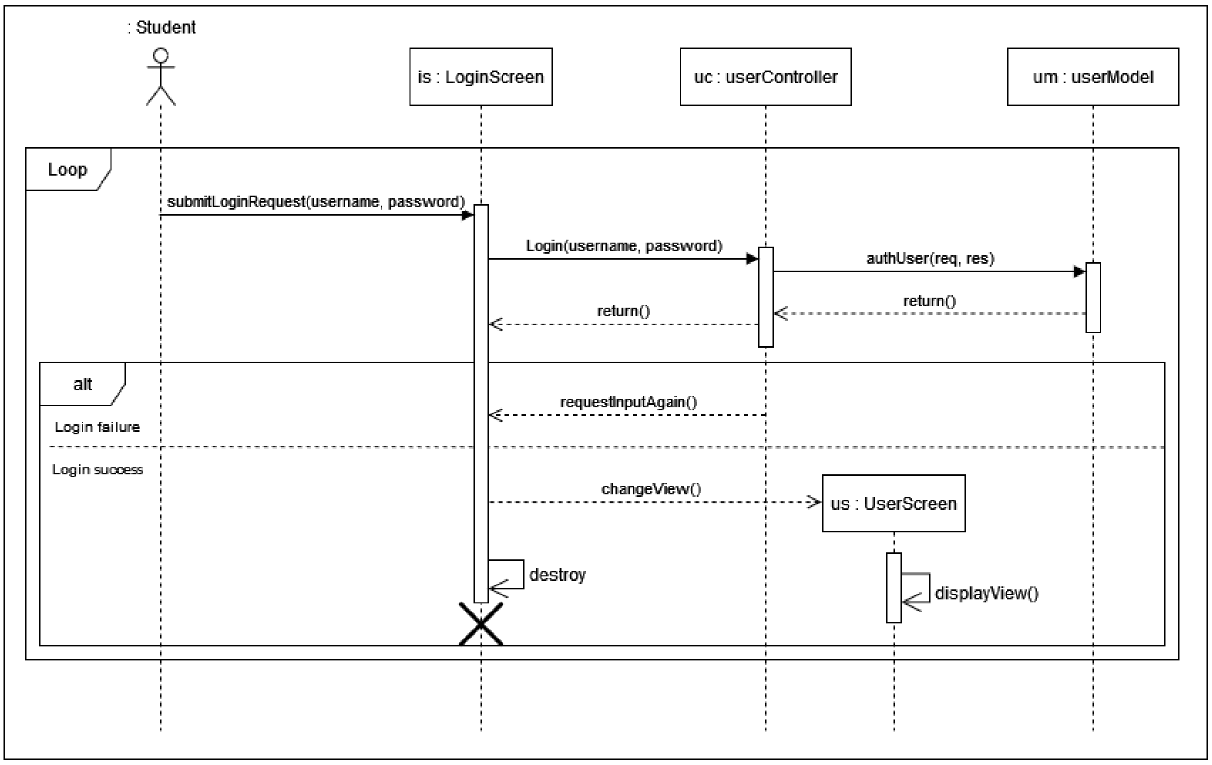


Figure 9: Sequence Diagram for Authentication

Description:

First, Student enters their username and password via the *submitLoginRequest(username, password)* function, **uc (userController)** will then call the *Login(username, password)* function to receive the values for username and password attributes, **uc** will then request the **um (userModel)** to call the *authUser(req, res)* function to authenticate username and password and return the value to **ls (loginScreen)**. If the values are invalid, **uc** will request Student via **ls** to input the username and password again, otherwise change view to **us**'s display view for the user.

2.2. Module: Printing

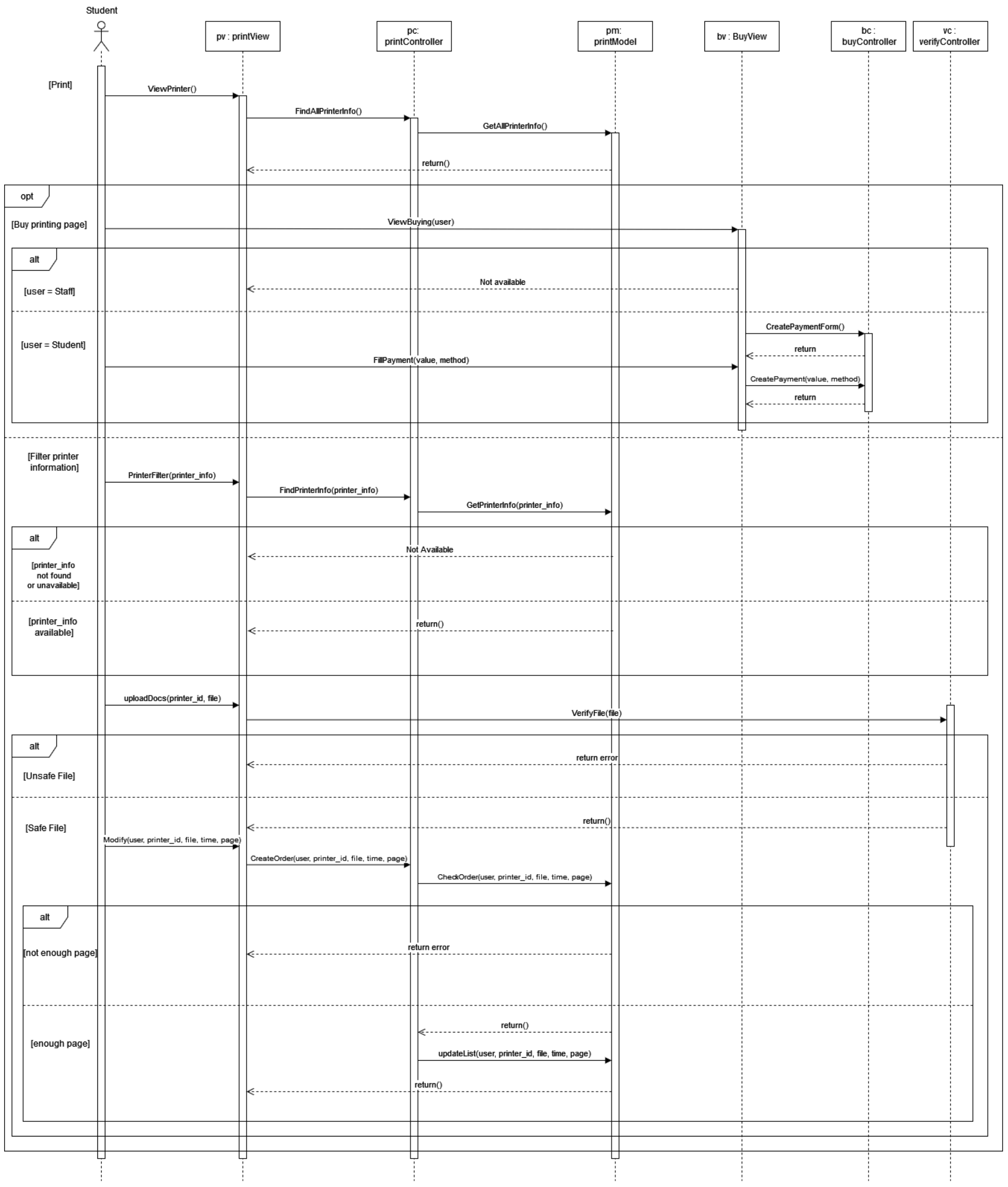


Figure 10: Sequence Diagram for Printing

Description:

First, the Student will open **pv (printView)** via *ViewPrinter()* function. Then **pc (printController)** calls the *FindAllPrinterInfo()* function to retrieve all printers present in the **pv**. **pc** then requests **pm (printModel)** to call the function *GetAllPrinterInfo()* to retrieve the list of all printers from the database, after that the list of printers will be returned and displayed at the **pv**. Next Student will have 2 parallel choices:

1. Student chooses "*Buy printing pages*", opens **bv (buyView)** via *ViewBuying(user)* function.
    a. If the inputted user is a Staff then return an error message to **pv** and display onto the screen.
    b. If the inputted user is a Student, then **bv** requests **bc (buyController)** to call the function *CreatePaymentForm()* to create a form with payment information, then return to the screen. Then Student provides payment information including the number of pages and the payment method and call *FillPayment(value, method)* function from **bv**, after that **bv** requests **bc** to call *CreatePayment(value, method)*, process the transaction via the payment system and returns the transaction result to the screen.
2. Student filters the printer information, **pv** will call *PrinterFilter(printer_info)* function based on the filtered information, then **bc** calls *FindPrinterInfo(prtiner_info)* function to find the appropriate printers to display onto the screen. Next **pc** requests **pm** to call *GetPrinterInfo(printer_info)* function to retrieve the list of printers from the database according to the provided information.
    a. If there are no printers satisfying the filtered information or if there are but the printer(s) are under maintenance then return an error message to **pv** and display onto the screen.
    b. Otherwise, return the result to **pv** and display the list of appropriate printers onto the screen.

   Student then chooses a printer to print and upload the document. After that **pv** will call *uploadDocs(printer_id, file)* function and requests **vc (viewController)** to call *VerifyFile(file)* to verify the contents of the uploaded document.

    a. If the uploaded file contains unsafe content, **vc** will send an error message to **pv** and display the error message onto the screen.
    b. Otherwise, **vc** will return the successful verification result and display onto the screen. After that, Student will modify and provide the printing information (time, number of pages), **pv** will call *Modify(user, printer,id, file, time, page), then* ***pc*** *will create an order via the function CreateOrder(user, printer_id, file, time, page)* and requests **pm** to call *CheckOrder(user, printer_id, file, time, page)* to export into the database in order to verify Student's information.
        i. If the Student's number of pages is not enough, **pm** will send an error message to **pv** and display onto the screen.

ii. Otherwise, **pm** returns the result to **pc**, then **pc** will proceed to request **pm** to call the function *updateList(user, printer_id, file, time, page)* to update the information into the database. Finally **pm** returns the result to **pv** and displays onto the screen.

2.3. Module: Printer Management

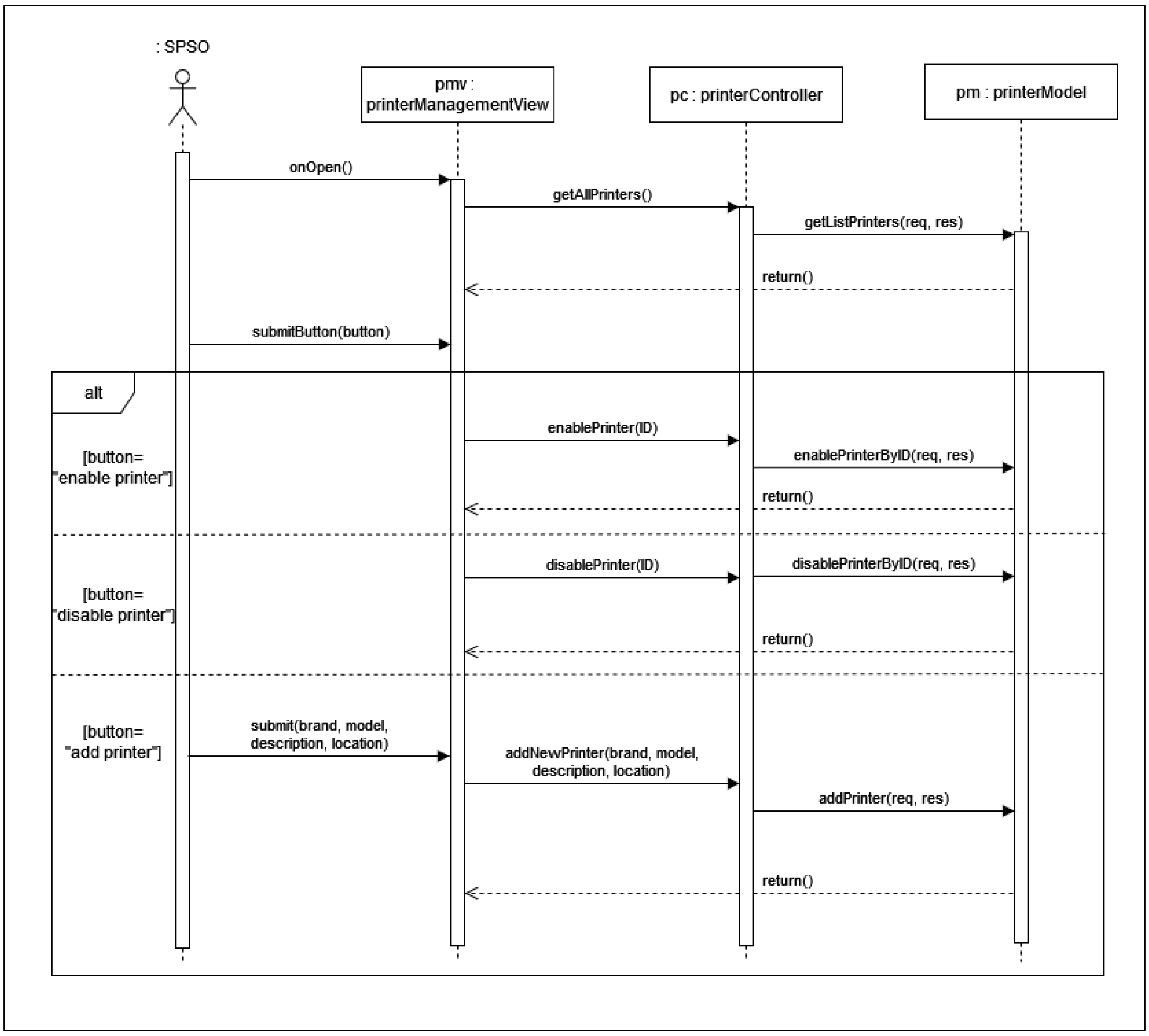

: SPSO
pmv : printerManagementView
pc : printerController
pm : printerModel
onOpen()
getAllPrinters()
getListPrinters(req, res)
return()
submitButton(button)
alt
[button= "enable printer"]
enablePrinter(ID)
enablePrinterByID(req, res)
return()
[button= "disable printer"]
disablePrinter(ID)
disablePrinterByID(req, res)
return()
[button= "add printer"]
submit(brand, model, description, location)
addNewPrinter(brand, model, description, location)
addPrinter(req, res)
return()

Figure 11: Sequence Diagram for Printer Management

Description:

1. First, **pmv (printerManagementView)** is opened by SPSO via the function *onOpen()*.
2. **pc (printerController)** calls *getAllPrinter()* function to retrieve all printers to display onto **pmv.**

3. **pc** requests **pm (printerModel)** to call *getListPrinters()* function to retrieve the list of printers from the database.
4. The printer list is returned via *return()* and displayed at **pmv**

At step 5, there are 3 choices;

A1:

5. SPSO chooses "*Enable printer*", then **pmv** will call *submit("enable printer").*
6. **pc** calls *enablePrinter(ID)* according to the ID of the chosen printer to activate that printer.
7. **pm** calls *enablePrinterByID(req, res)* from **pc**'s request to set the state value of printer to 1.
8. Update the database, *return()* and display at **pmv.**

A2:

5. SPSO chooses "*Disable printer*", then **pmv** calls *submit("disable printer").*
6. **pc** calls *disablePrinter(ID)* according to the ID of the chosen printer to deactivate that printer.
7. **pm** call *disablePrinterByID(req, res)* from **pc**'s request to set the state value of printer to 0.
8. Update the database, *return()* and display at **pmv.**

A3:

5. SPSO chooses "*Add printer*", then **pmv** calls *submit("add printer").*
6. SPSO fills in the information of the printer and **pmv** calls *submit(brand, model, description, location).*
7. **pc** calls *addNewPrinter(brand, model, description, location)* to add the new printer with the attributes provided by SPSO.
8. **pm** calls *addPrinter(req, res)* from **pmc**'s request to add the new printer to the database
9. Update the database, *return()* and display at **pmv.**

3. Class Diagram (Task 2.3)

In the report, our group decided to design the class diagram using the **MVC architecture**. **MVC (Model - View - Controller)** is a software design pattern used for developing user interfaces and organizing application logic. It divides an application into three interconnected components:

1. **Model**: This represents the application's data and business logic. It manages the data, responds to requests for information, and updates when necessary.
2. **View**: The View is responsible for displaying the data from the Model to the user. It defines how the user interface looks and interacts but doesn't contain logic for manipulating the data itself.
3. **Controller**: The Controller acts as an intermediary between the Model and the View. It handles user input, processes it, and sends instructions to the Model to update, while also determining which View to display.

By separating these concerns, MVC helps create more maintainable, scalable, and testable applications, making it a popular choice in web and software development frameworks. In the class diagram, each class type is represented by a different color: controllers are red, models are blue, views are yellow, and external components are purple, following the MVC pattern.

3.1. Authentication Module

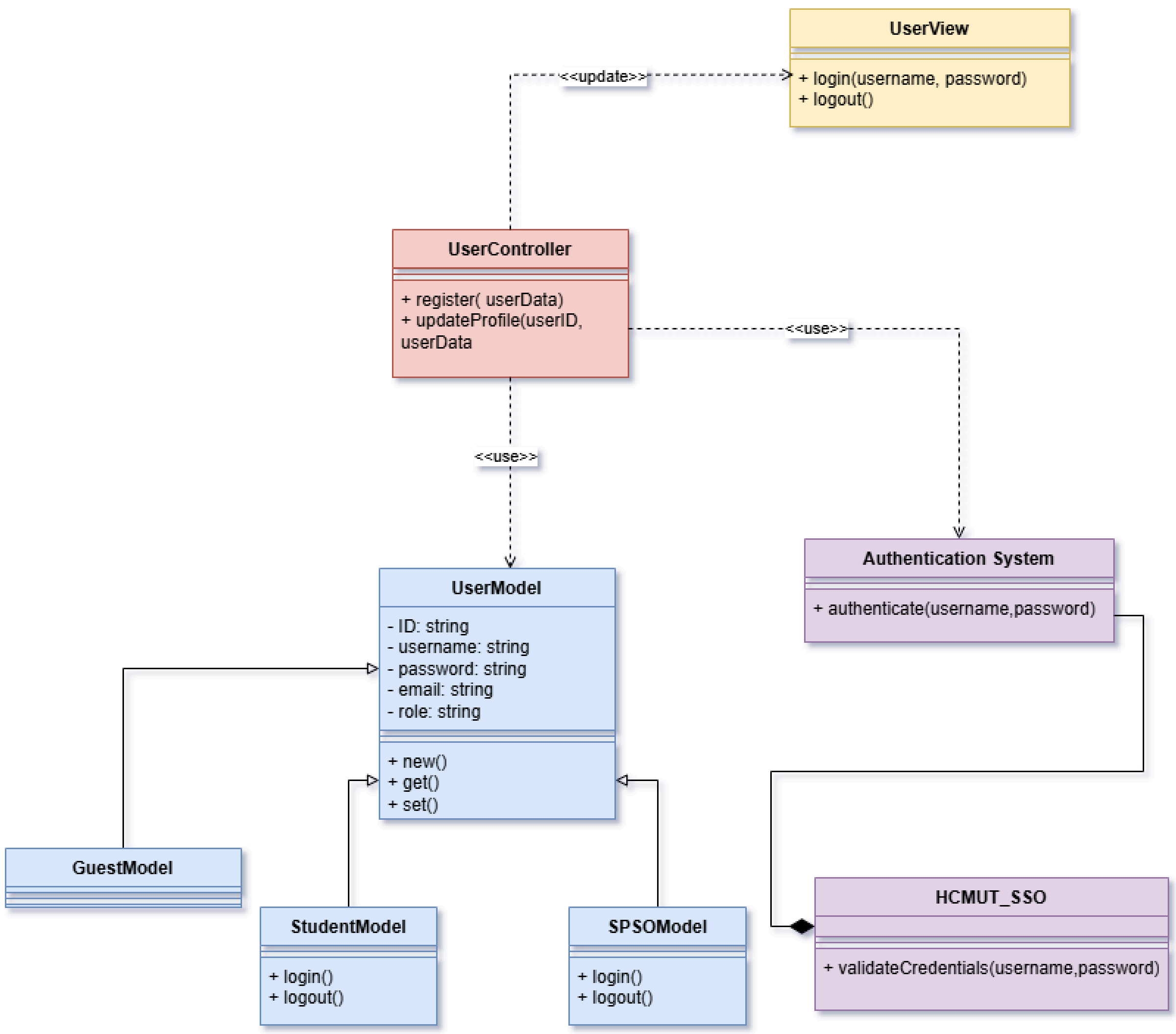


Figure 12: Authentication Module Class Diagram

Description:

UserController

- Responsible for handling requests from the view and interacting with the model.
- **Key methods:**
  + ***register(userData)***: Registers a new user.
  + ***updateProfile(userID, userData)***: Updates the profile of a user.

UserView

- This class is part of the view, responsible for interacting with the user interface.
- **Key methods:**
  + ***login(username, password)***: Handles user login.
  + ***logout()***: Handles user logout.

UserModel

- Represents the data and business logic related to users.
- **Attributes:**
  + ***ID***, ***username***, ***password***, ***email***, ***role***: Fields to store user information.
- **Key methods:**
  + ***new()***: Creates a new user.
  + ***get()***: Retrieves user information.
  + ***set()***: Updates user information.

Authentication System

- Handles authentication logic separate from the user model.
- **Key method:**
  + ***authenticate(username, password)***: Authenticates the user's credentials.

GuestModel, **StudentModel**, **SPSOModel**

- These are specialized models inheriting from ***UserModel***.
- Both ***StudentModel*** and ***SPSOModel*** contain:
  + ***login()***: Logs in a specific type of user.
  + ***logout()***: Logs out the user.

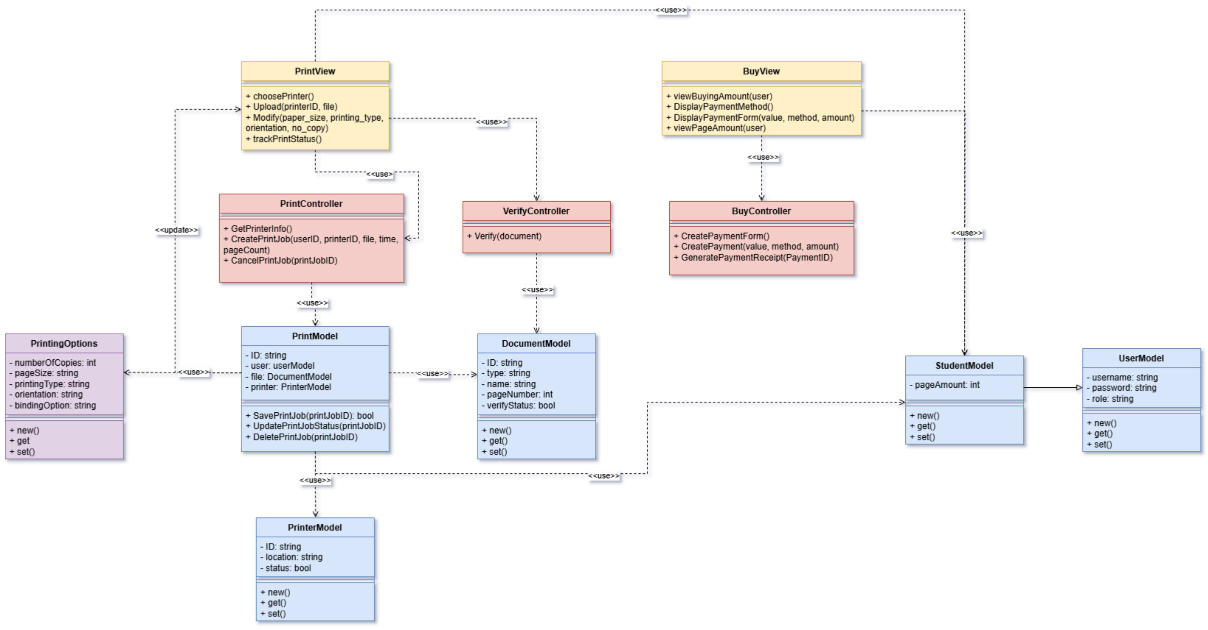
HCMUT_SSO

- Likely part of an external single sign-on (SSO) system.
- **Key method:**
  + ***validateCredentials(username, password)***: Validates the user's credentials using the SSO system.

Key Relationships:

- **UserController** interacts with **UserModel**, using it to manage user data (e.g., registration and profile updates).
- **UserView** communicates with **UserController** to display user data and handle actions like login and logout.
- **UserController** also uses an **Authentication System** for user authentication.
- The **Authentication System** integrates with **HCMUT_SSO** to validate credentials.
- **UserModel** serves as the parent class for **GuestModel**, **StudentModel**, and **SPSOModel**, which inherit login and logout functionality.

3.2. Printing Module



<u>Figure 13:</u> Printing Module Class Diagram

<u>Description:</u>

PrintView

- This class is responsible for the user interface related to printing.
- **<u>Key methods:</u>**
    + ***choosePrinter()***: Allows the user to select a printer.
    + ***Upload(printerID, file)***: Uploads the file to be printed.
    + ***Modify(paper_size, printing_type, orientation, no_copy)***: Modifies printing settings.
    + ***trackPrintStatus()***: Tracks the status of the print job.

PrintController

- Manages user requests related to printing, interacting with the model to create or update print jobs.
- **<u>Key methods:</u>**
    + ***GetPrinterInfo()***: Retrieves information about the printer.
    + ***CreatePrintJob(userID, printerID, file, time, pageCount)***: Creates a new print job.
    + ***CancelPrintJob(printJobID)***: Cancels an existing print job.

PrintModel

- Represents the business logic and data for print jobs.

- **Attributes:**
  + ***ID***, ***user*** **(from** ***UserModel*****)**, ***file*** **(from** ***DocumentModel*****)**, ***printer*** **(from** ***PrinterModel*****)**.
- **Key methods:**
  + ***SavePrintJob(printJobID)***: Saves a new print job.
  + ***UpdatePrintJobStatus(printJobID)***: Updates the status of a print job.
  + ***DeletePrintJob(printJobID)***: Deletes a print job.

PrinterOptions

- Represents various settings related to a print job..
- **Attributes:**
  + ***numberOfCopies (integer):*** Specifies how many copies to print
  + ***pageSize (string):*** Indicates the size of the paper (e.g., A4, Letter*)*
  + ***printingType (string):*** Likely represents the type of printing (e.g., one-sided)
  + ***orientation (string):*** Specifies page orientation (e.g., portrait, landscape)
  + ***bindingOption (string):*** Indicates any binding preferences
- **Key Methods**:
  + ***new()***: Initializes a new printer instance.
  + ***get()*****:** Retrieves the details of a specific printer.
  + ***set()*****:** Updates the printer's details.

PrinterModel

- Represents the data related to the printer itself.
- **Attributes:**
  + ***ID***, ***location***, ***status***: Stores information about the printer and its availability.
- **Key Methods**:
  + ***new()***: Initializes a new printer instance.
  + ***get()*****:** Retrieves the details of a specific printer.
  + ***set()*****:** Updates the printer's details.

DocumentModel

- This class models the document to be printed.
- **Attributes:**
  + ***ID***, ***type***, ***name***, ***pageNumber***, ***verifyStatus***: Describes the document's type, name, number of pages, and whether it has been verified.
- **Key Methods**:
  + ***new()***: Initializes a new printer instance.

  + ***get()*****:** Retrieves the details of a specific printer.
  + ***set()*****:** Updates the printer's details.

VerifyController

- This controller handles verification logic for documents before printing.
- **Key method:**
  + ***Verify(document)***: Verifies the file before allowing it to be printed.

BuyView

- Responsible for presenting the purchase information and payment forms to the user.
- **Key methods:**
  + ***viewBuyingAmount(user)***: Displays the total amount for a user to pay.
  + ***DisplayPaymentMethod()***: Shows the available payment methods.
  + ***DisplayPaymentForm(value, method, amount)***: Shows a payment form based on the selected value, method, and amount.
  + ***viewPageAmount(user):*** Display the remaining amount of printing pages the user had left in the account.

BuyController

- Manages user requests related to purchasing and payments.
- **Key methods:**
  + ***CreatePaymentForm()***: Creates a form for the payment process.
  + ***CreatePayment(value, method, amount)***: Processes the payment.
  + ***GeneratePaymentReceipt(PaymentID)***: Generates a receipt after successful payment.

StudentModel

- Contains the data related to the student user..
- **Attributes**:
  + ***pageAmount***: Stores the page amount left in the student's account..
- **Key Methods**:
  + ***new()***: Initializes a new printer instance.
  + ***get()*****:** Retrieves the details of a specific printer.
  + ***set()*****:** Updates the printer's details.

UserModel

- Contains the data related to the user.
- **Attributes**:
  + ***username***, ***password***, ***role***: Stores user credentials, roles for tracking purposes.

- <u>**Key Methods**</u>:
  + ***new()***: Initializes a new printer instance.
  + ***get()*****:** Retrieves the details of a specific printer.
  + ***set()*****:** Updates the printer's details.

<u>**Key Relationships:**</u>

- **PrintView** interacts with **PrintController** to modify printing settings, upload files, and track print job status, also retrieving information from the **StudentModel**.
- **PrintController** communicates with **PrintModel** to create, update, or delete print jobs, while also retrieving information from the **PrinterModel**.
- **PrintModel** links to **StudentModel** (representing the user who submitted the print job), **PrintingOptions**(representing the configurations of a print job) and **DocumentModel** (representing the document to be printed).
- **VerifyController** is responsible for verifying documents before they are printed, interacting with the **Document** class.
- **BuyView** and **BuyController** manage payment interactions, where the view handles displaying payment-related information and the controller processes payments and receipts while updating the page amount from the **StudentModel**.
- **UserModel** serves as the parent class for **StudentModel**.

3.3. Printer Management Module

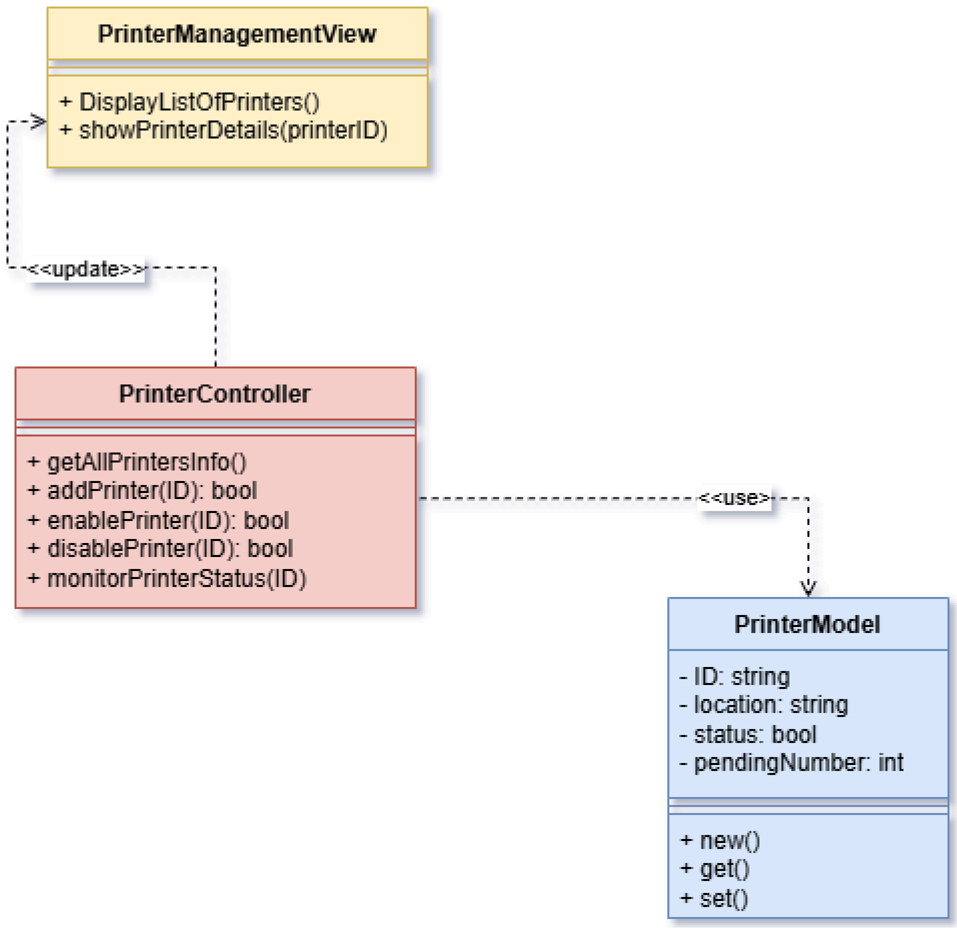


Figure 14: Printer Management Class Diagram

Description:

PrinterManagementView

This class is responsible for the user interface related to managing printers.

- **Key Methods**:
  + ***DisplayListOfPrinters()***: Displays the list of all printers available in the system.
  + ***showPrinterDetails(printerID)***: Shows detailed information for a specific printer identified by its ID.

PrinterController

Handles user requests related to managing printers and interacts with the model to perform printer operations.

- **Key Methods**:

- ***getAllPrintersInfo()***: Retrieves the information of all the printers from the system.
- ***addPrinter(ID): bool*:** Adds a new printer to the system by its ID, returning a boolean indicating success or failure.
- ***enablePrinter(ID): bool***: Enables or activates a printer using its ID, returning a success boolean.
- ***disablePrinter(ID): bool***: Disables or deactivates a printer using its ID, returning a success boolean.
- ***monitorPrinterStatus(ID)***: Monitors the status of a printer based on its ID (e.g., active, idle, offline).

PrinterModel

Represents the data and attributes related to a printer.

- **Attributes**:
  - ***ID***: A unique identifier for each printer.
  - ***location***: The physical location of the printer.
  - ***status***: A boolean indicating whether the printer is enabled or disabled.
  - ***pendingNumber****:* The number of requests on each printer.
- **Key Methods**:
  - ***new()***: Initializes a new printer instance.
  - ***get()*:** Retrieves the details of a specific printer.
  - ***set()*:** Updates the printer's details.

Key Relationships:

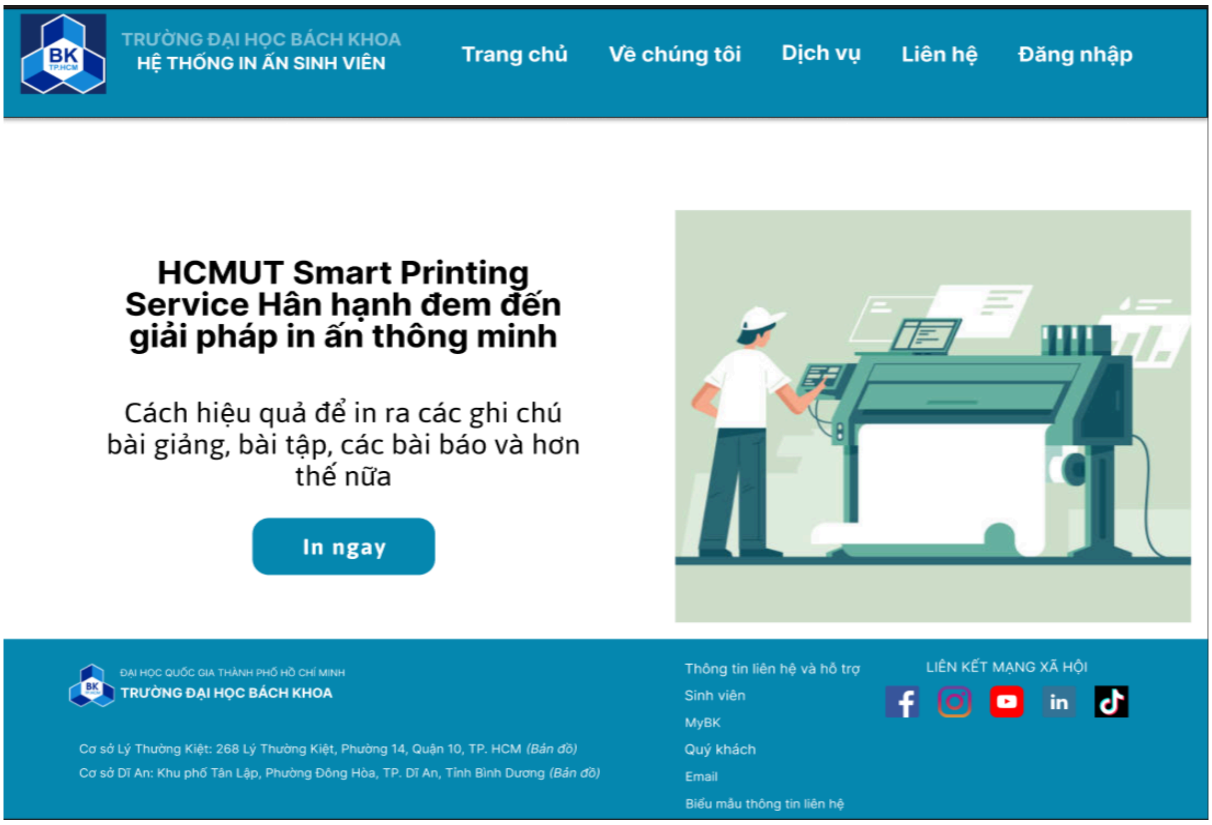
- **PrinterManagementView** interacts with **PrinterController** to display each printer details.
- **PrinterController** communicates with **PrinterModel** to add, enable, remove, and monitor printers, as well as assigning print jobs to printers.

4. User Interface (Task 2.4)

In this project, our group decided to choose a web application to implement. We use figma to implement our app interface. For further view here are the prototype and figma links:

- **Figma :** View here
- **Prototype:** View here

4.1. Log in



TRƯỜNG ĐẠI HỌC BÁCH KHOA
HỆ THỐNG IN ẤN SINH VIÊN

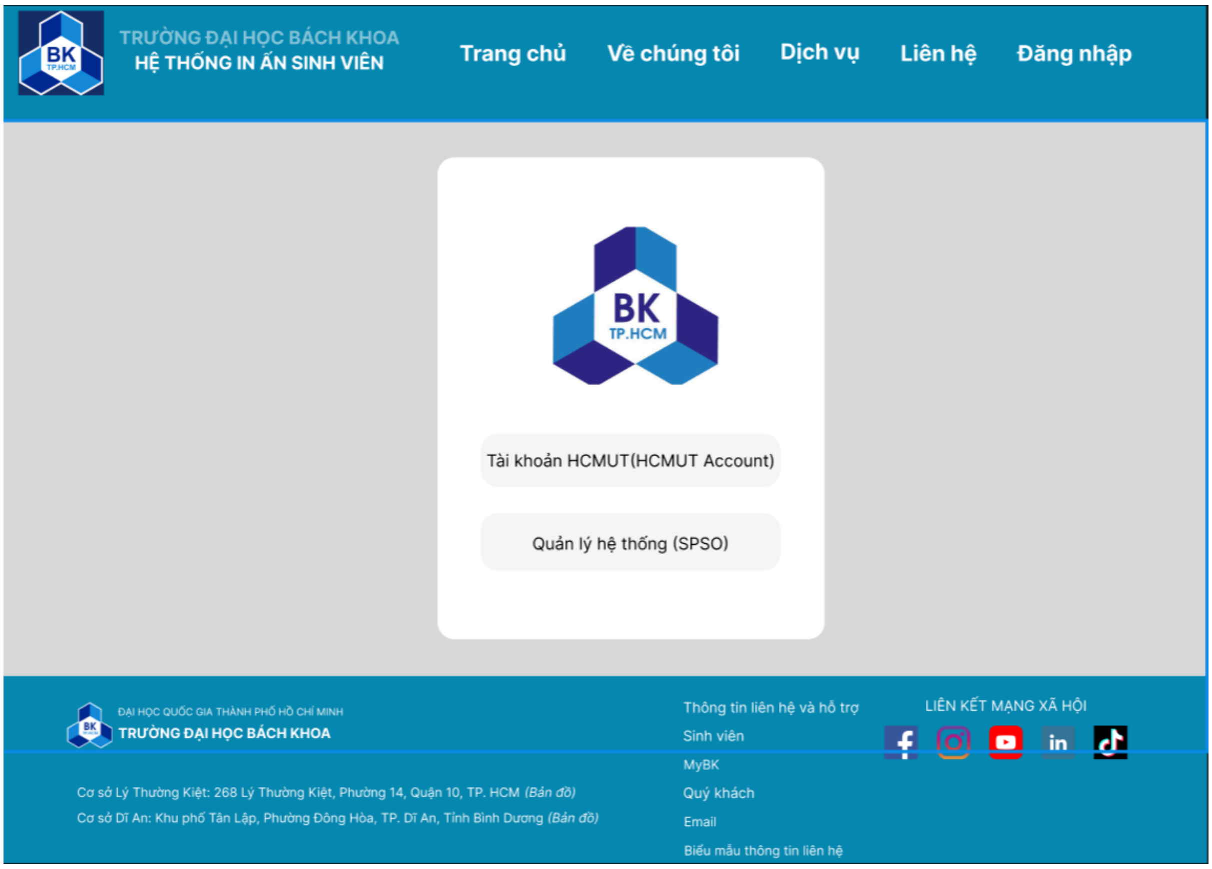
HCMUT Smart Printing Service Hân hạnh đem đến giải pháp in ấn thông minh

Cách hiệu quả để in ra các ghi chú bài giảng, bài tập, các bài báo và hơn thế nữa

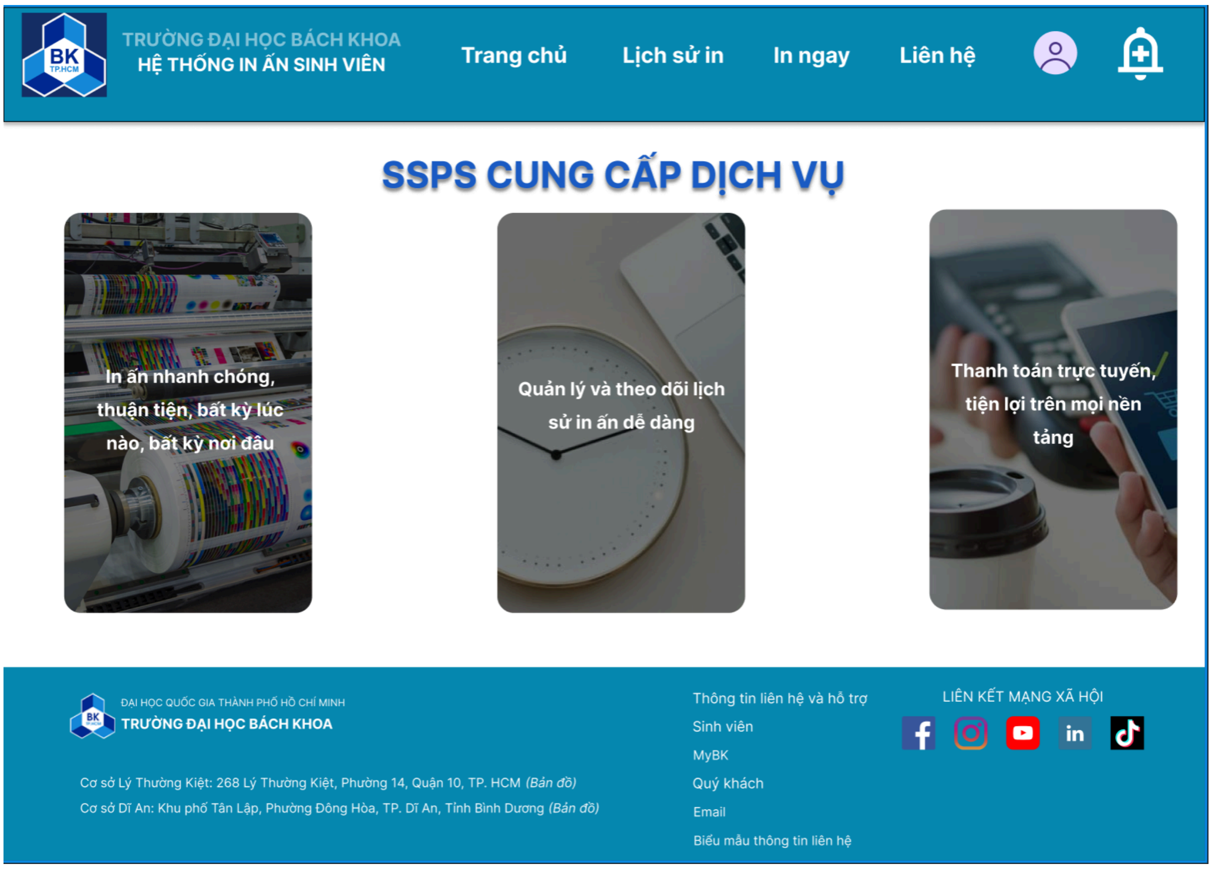
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Cơ sở Lý Thường Kiệt: 268 Lý Thường Kiệt, Phường 14, Quận 10, TP. HCM *(Bản đồ)*
Cơ sở Dĩ An: Khu phố Tân Lập, Phường Đông Hòa, TP. Dĩ An, Tỉnh Bình Dương *(Bản đồ)*

Figure 15: Dashboard


TRƯỜNG ĐẠI HỌC BÁCH KHOA
HỆ THỐNG IN ẤN SINH VIÊN
Trang chủ
Về chúng tôi
Dịch vụ
Liên hệ
Đăng nhập
Tài khoản HCMUT(HCMUT Account)
Quản lý hệ thống (SPSO)
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
Cơ sở Lý Thường Kiệt: 268 Lý Thường Kiệt, Phường 14, Quận 10, TP. HCM (Bản đồ)
Cơ sở Dĩ An: Khu phố Tân Lập, Phường Đông Hòa, TP. Dĩ An, Tỉnh Bình Dương (Bản đồ)
Thông tin liên hệ và hỗ trợ
Sinh viên
MyBK
Quý khách
Email
Biểu mẫu thông tin liên hệ
LIÊN KẾT MẠNG XÃ HỘI

<u>Figure 16:</u> Log in screen



SSPS CUNG CẤP DỊCH VỤ

In ấn nhanh chóng, thuận tiện, bất kỳ lúc nào, bất kỳ nơi đâu

Quản lý và theo dõi lịch sử in ấn dễ dàng

Thanh toán trực tuyến, tiện lợi trên mọi nền tảng

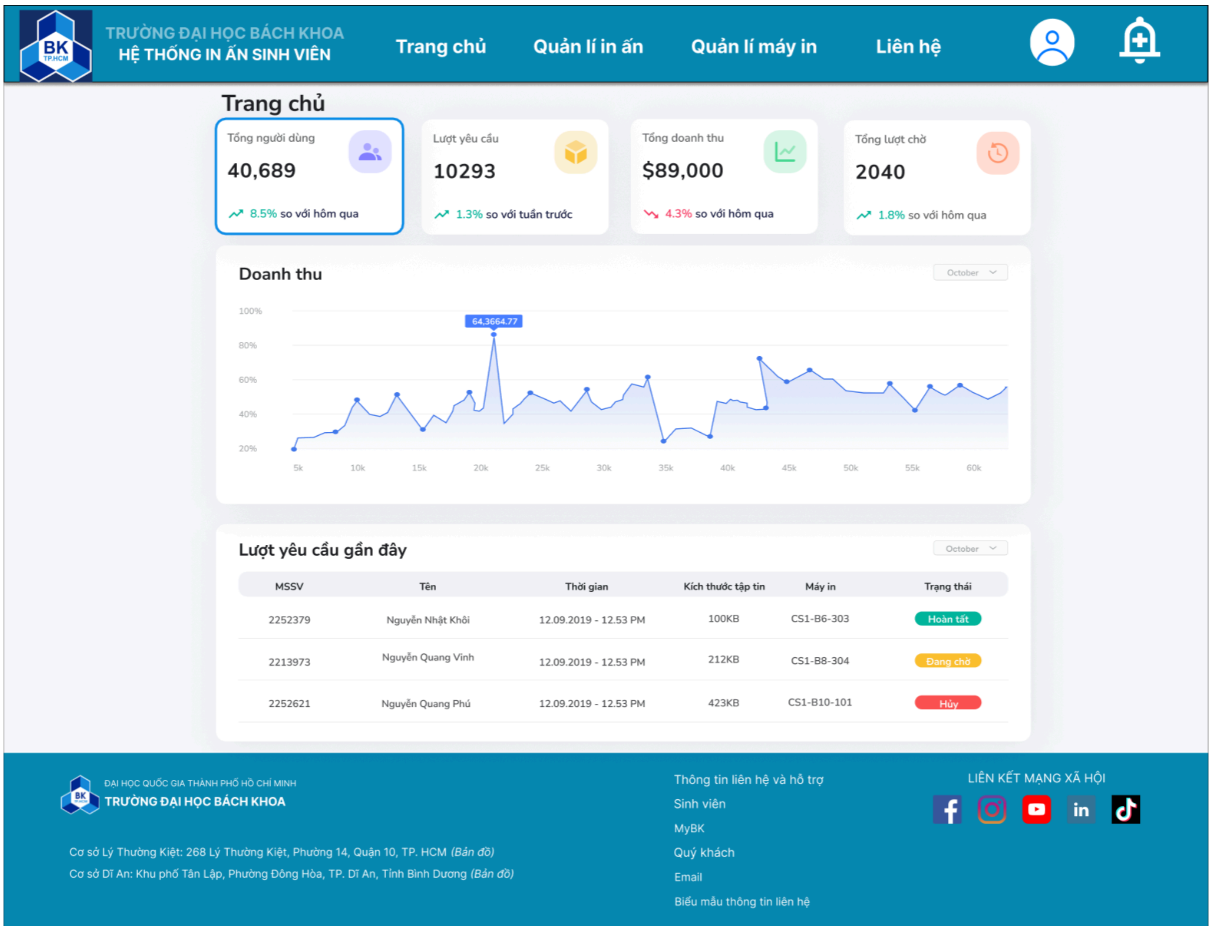
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Cơ sở Lý Thường Kiệt: 268 Lý Thường Kiệt, Phường 14, Quận 10, TP. HCM *(Bản đồ)*
Cơ sở Dĩ An: Khu phố Tân Lập, Phường Đông Hòa, TP. Dĩ An, Tỉnh Bình Dương *(Bản đồ)*

Thông tin liên hệ và hỗ trợ
Sinh viên
MyBK
Quý khách
Email
Biểu mẫu thông tin liên hệ

LIÊN KẾT MẠNG XÃ HỘI

Figure 17: Student's dashboard



Trang chủ

Tổng người dùng
40,689
8.5% so với hôm qua

Lượt yêu cầu
10293
1.3% so với tuần trước

Tổng doanh thu
$89,000
4.3% so với hôm qua

Tổng lượt chờ
2040
1.8% so với hôm qua

Doanh thu

Lượt yêu cầu gần đây

MSSV	Tên	Thời gian	Kích thước tập tin	Máy in	Trạng thái
2252379	Nguyễn Nhật Khôi	12.09.2019 - 12.53 PM	100KB	CS1-B6-303	Hoàn tất
2213973	Nguyễn Quang Vinh	12.09.2019 - 12.53 PM	212KB	CS1-B8-304	Đang chờ
2252621	Nguyễn Quang Phú	12.09.2019 - 12.53 PM	423KB	CS1-B10-101	Hủy

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

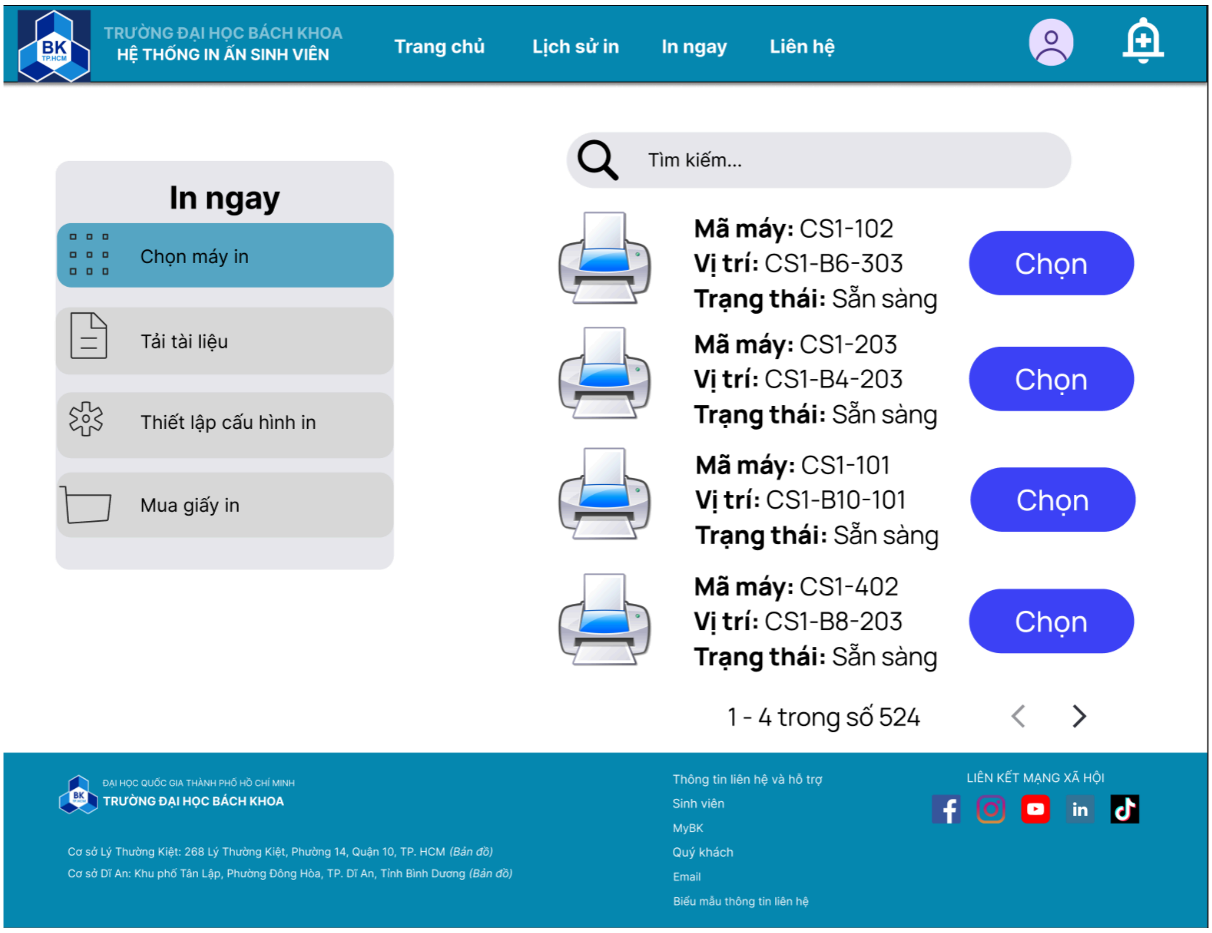
Cơ sở Lý Thường Kiệt: 268 Lý Thường Kiệt, Phường 14, Quận 10, TP. HCM *(Bản đồ)*
Cơ sở Dĩ An: Khu phố Tân Lập, Phường Đông Hòa, TP. Dĩ An, Tỉnh Bình Dương *(Bản đồ)*

Thông tin liên hệ và hỗ trợ
Sinh viên
MyBK
Quý khách
Email
Biểu mẫu thông tin liên hệ

LIÊN KẾT MẠNG XÃ HỘI

<u>Figure 18:</u> SPSO's dashboard

4.2. Printing



TRƯỜNG ĐẠI HỌC BÁCH KHOA
HỆ THỐNG IN ẤN SINH VIÊN

Trang chủ | Lịch sử in | In ngay | Liên hệ

In ngay

- Chọn máy in
- Tải tài liệu
- Thiết lập cấu hình in
- Mua giấy in

Tìm kiếm...

Mã máy: CS1-102
Vị trí: CS1-B6-303
Trạng thái: Sẵn sàng
Chọn

Mã máy: CS1-203
Vị trí: CS1-B4-203
Trạng thái: Sẵn sàng
Chọn

Mã máy: CS1-101
Vị trí: CS1-B10-101
Trạng thái: Sẵn sàng
Chọn

Mã máy: CS1-402
Vị trí: CS1-B8-203
Trạng thái: Sẵn sàng
Chọn

1 - 4 trong số 524

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Cơ sở Lý Thường Kiệt: 268 Lý Thường Kiệt, Phường 14, Quận 10, TP. HCM *(Bản đồ)*
Cơ sở Dĩ An: Khu phố Tân Lập, Phường Đông Hòa, TP. Dĩ An, Tỉnh Bình Dương *(Bản đồ)*

Thông tin liên hệ và hỗ trợ
Sinh viên
MyBK
Quý khách
Email
Biểu mẫu thông tin liên hệ

LIÊN KẾT MẠNG XÃ HỘI

Figure 19: Choose printer

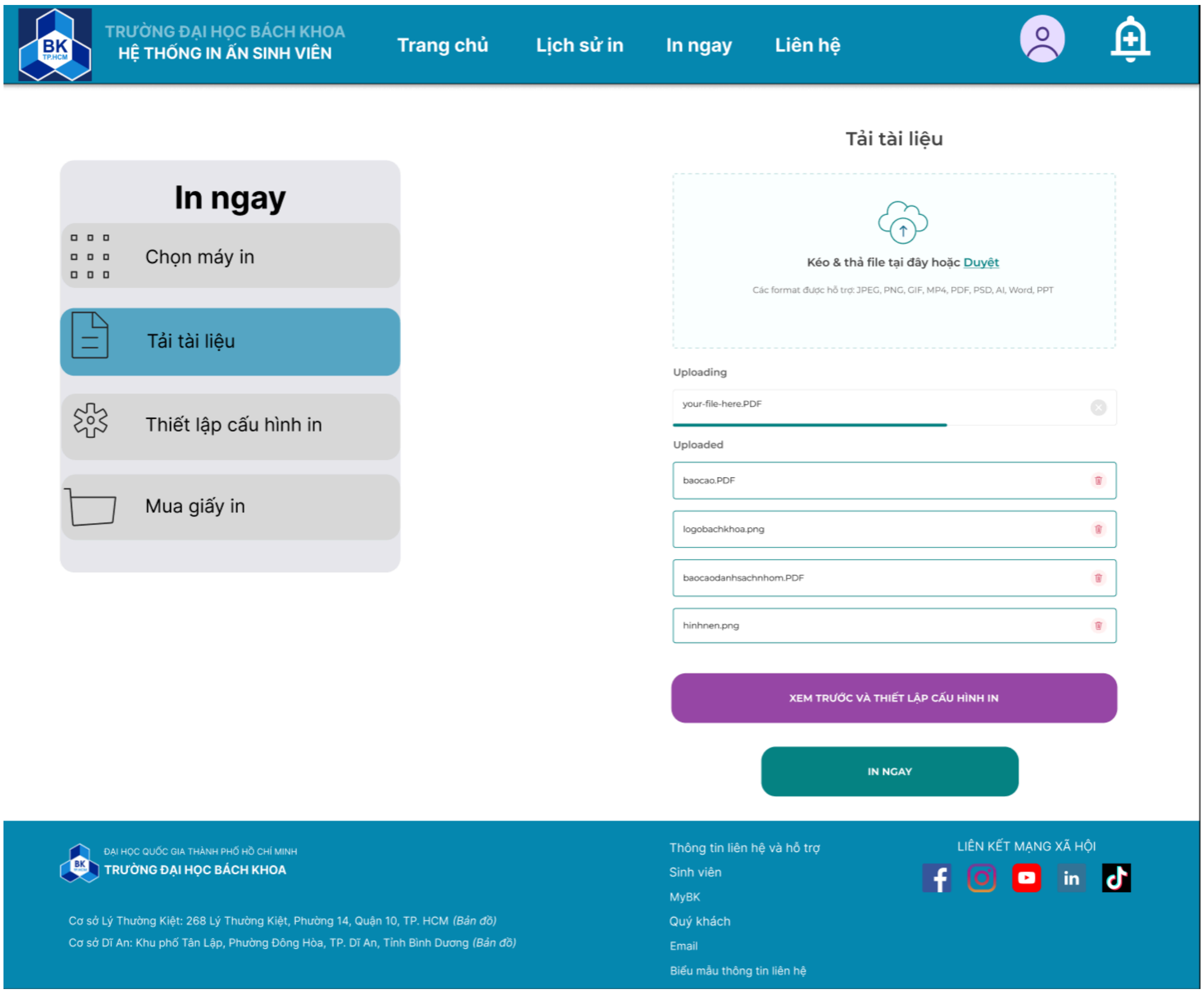
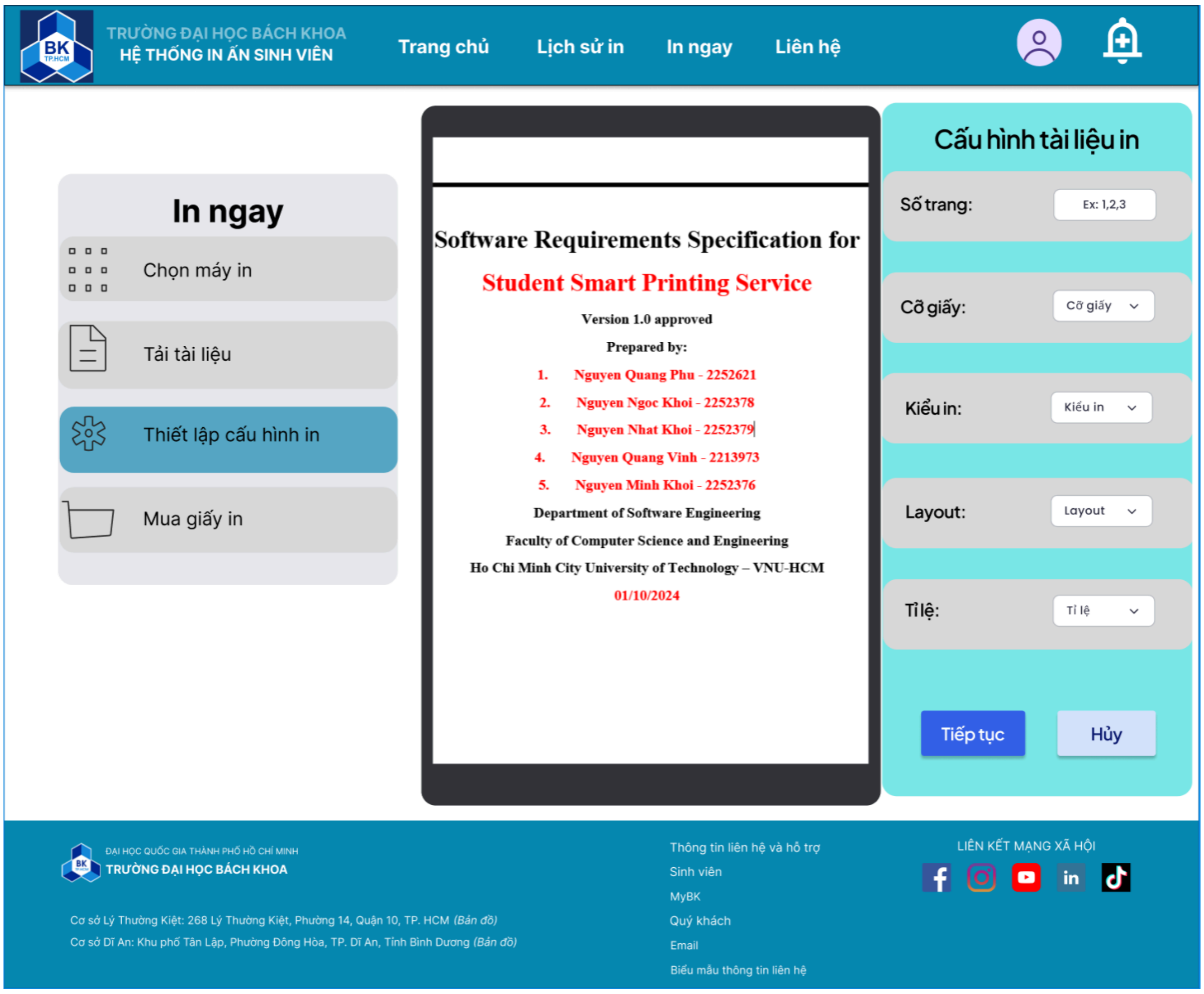

TRƯỜNG ĐẠI HỌC BÁCH KHOA
HỆ THỐNG IN ẤN SINH VIÊN
Trang chủ
Lịch sử in
In ngay
Liên hệ
In ngay
Chọn máy in
Tải tài liệu
Thiết lập cấu hình in
Mua giấy in
Tải tài liệu
Kéo & thả file tại đây hoặc Duyệt
Các format được hỗ trợ: JPEG, PNG, GIF, MP4, PDF, PSD, AI, Word, PPT
Uploading
your-file-here.PDF
Uploaded
baocao.PDF
logobachkhoa.png
baocaodanhsachnhom.PDF
hinhnen.png
XEM TRƯỚC VÀ THIẾT LẬP CẤU HÌNH IN
IN NGAY
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
Cơ sở Lý Thường Kiệt: 268 Lý Thường Kiệt, Phường 14, Quận 10, TP. HCM (Bản đồ)
Cơ sở Dĩ An: Khu phố Tân Lập, Phường Đông Hòa, TP. Dĩ An, Tỉnh Bình Dương (Bản đồ)
Thông tin liên hệ và hỗ trợ
Sinh viên
MyBK
Quý khách
Email
Biểu mẫu thông tin liên hệ
LIÊN KẾT MẠNG XÃ HỘI

Figure 20: Upload files



Software Requirements Specification for
Student Smart Printing Service
Version 1.0 approved
Prepared by:
1. Nguyen Quang Phu - 2252621
2. Nguyen Ngoc Khoi - 2252378
3. Nguyen Nhat Khoi - 2252379
4. Nguyen Quang Vinh - 2213973
5. Nguyen Minh Khoi - 2252376
Department of Software Engineering
Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology – VNU-HCM
01/10/2024

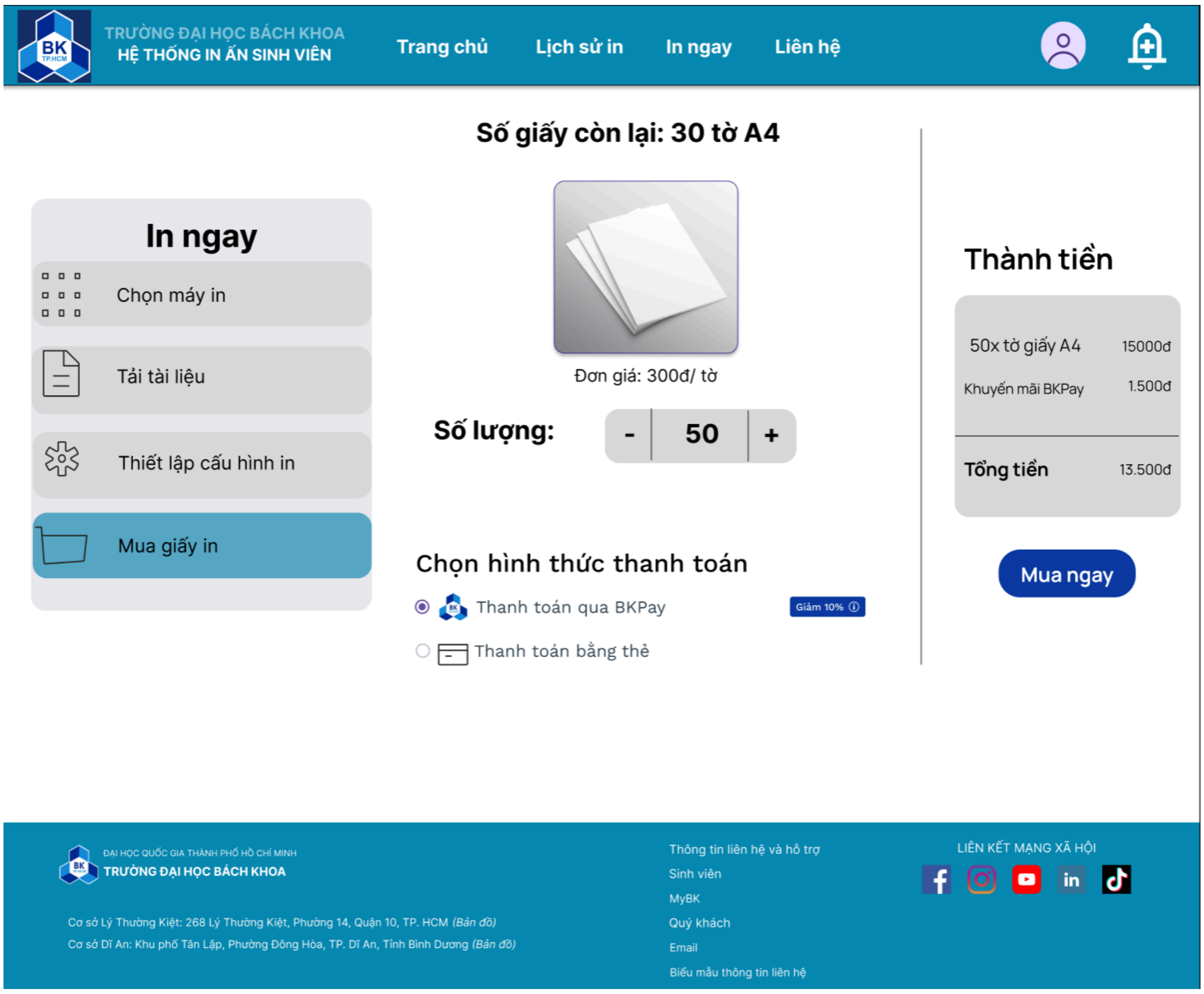
Số trang:	Ex: 1,2,3
Cỡ giấy:	Cỡ giấy
Kiểu in:	Kiểu in
Layout:	Layout
Tỉ lệ:	Tỉ lệ

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
Cơ sở Lý Thường Kiệt: 268 Lý Thường Kiệt, Phường 14, Quận 10, TP. HCM *(Bản đồ)*
Cơ sở Dĩ An: Khu phố Tân Lập, Phường Đông Hòa, TP. Dĩ An, Tỉnh Bình Dương *(Bản đồ)*

Thông tin liên hệ và hỗ trợ
Sinh viên
MyBK
Quý khách
Email
Biểu mẫu thông tin liên hệ

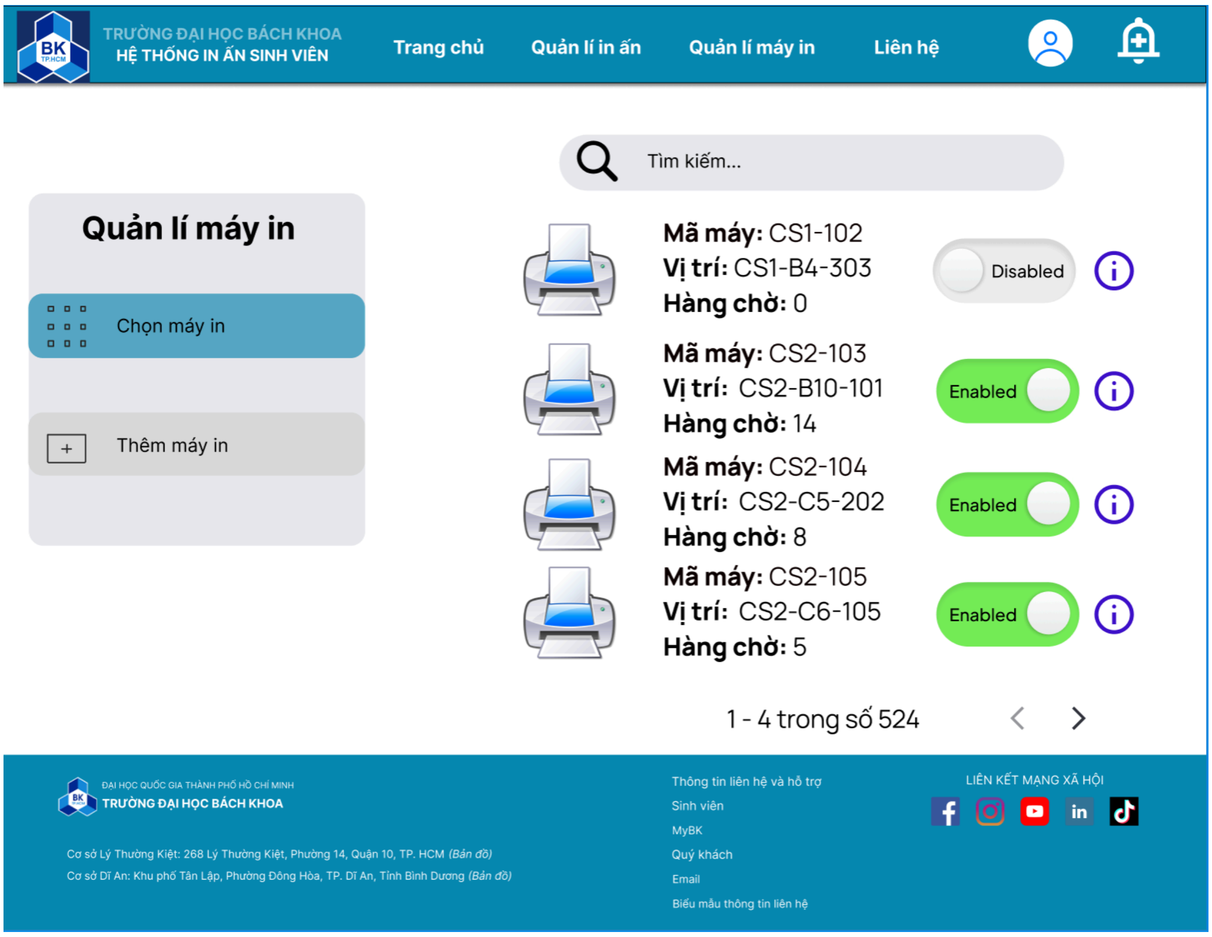
LIÊN KẾT MẠNG XÃ HỘI

Figure 21: Printing configuration



<u>Figure 22:</u> Buy printing paper

4.3. Printer Management


TRƯỜNG ĐẠI HỌC BÁCH KHOA
HỆ THỐNG IN ẤN SINH VIÊN

Trang chủ
Quản lí in ấn
Quản lí máy in
Liên hệ

Tìm kiếm...

Quản lí máy in

Chọn máy in

Thêm máy in

Mã máy: CS1-102
Vị trí: CS1-B4-303
Hàng chờ: 0
Disabled

Mã máy: CS2-103
Vị trí: CS2-B10-101
Hàng chờ: 14
Enabled

Mã máy: CS2-104
Vị trí: CS2-C5-202
Hàng chờ: 8
Enabled

Mã máy: CS2-105
Vị trí: CS2-C6-105
Hàng chờ: 5
Enabled

1 - 4 trong số 524

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

Cơ sở Lý Thường Kiệt: 268 Lý Thường Kiệt, Phường 14, Quận 10, TP. HCM *(Bản đồ)*
Cơ sở Dĩ An: Khu phố Tân Lập, Phường Đông Hòa, TP. Dĩ An, Tỉnh Bình Dương *(Bản đồ)*

Thông tin liên hệ và hỗ trợ
Sinh viên
MyBK
Quý khách
Email
Biểu mẫu thông tin liên hệ

LIÊN KẾT MẠNG XÃ HỘI

Figure 23: Printer management

III. Architecture Design

1. MVC Architecture

The team has chosen the MVC (Model - View - Controller) architecture, one of the most popular architectures, to design the HCMUT_SPSS system.

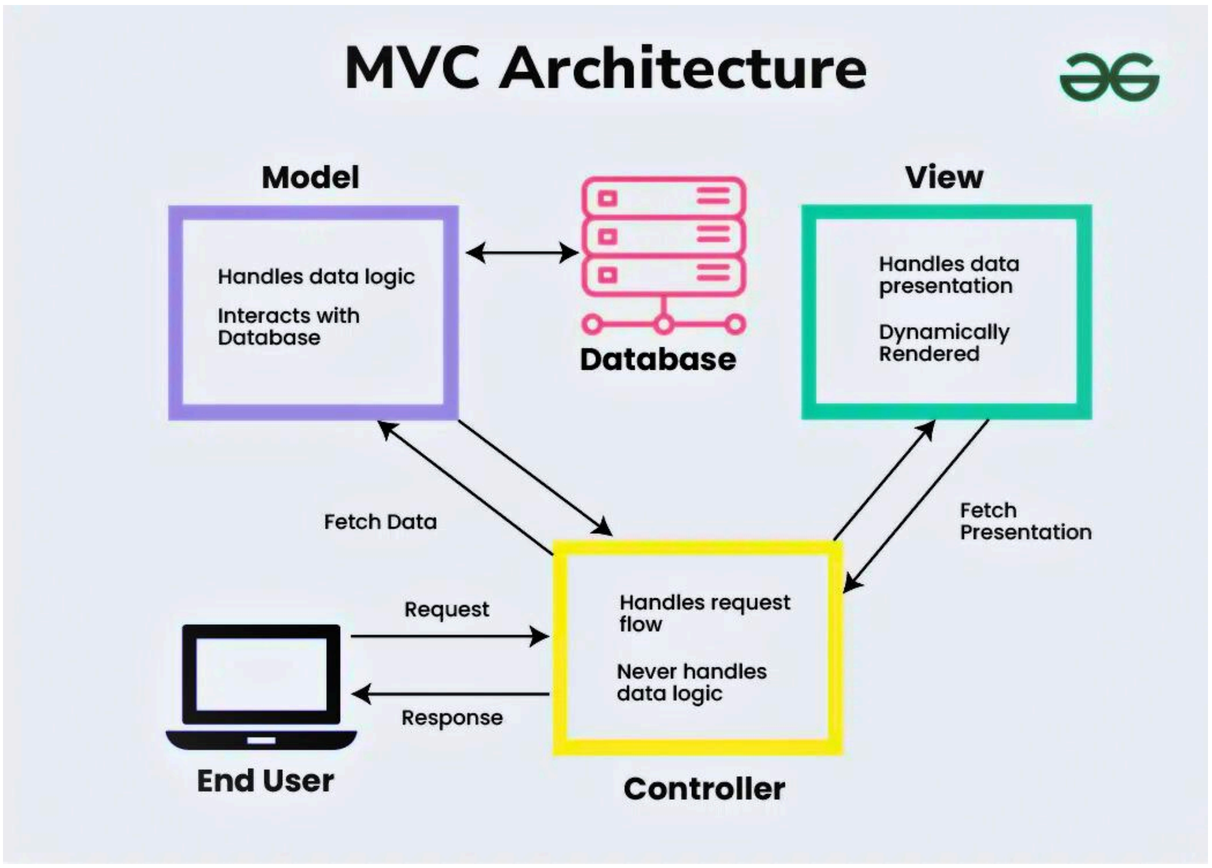


Figure 24: MVC Architectural Model (Source: GeeksforGeeks.org)

1.1. Advantages and Disadvantages of MVC

Advantages of MVC:

1. **Clear Separation of Responsibilities**: MVC divides the application into three distinct components - Model, View, and Controller - each with well-defined roles. This separation simplifies code organization, making it easier to manage and maintain.
2. **Enhanced Scalability and Maintainability**: By isolating each component, the MVC architecture allows for easy modifications to individual parts without impacting the entire system. This modularity promotes scalability and long-term maintenance as the application evolves.
3. **Parallel Development Capability**: The MVC pattern allows multiple developers to work on different layers simultaneously - one can work on the View while another focuses on the Controller, and a third works on the Model. This concurrent development reduces overall development time and prevents code conflicts.
4. **Flexible Integration of New Features**: MVC supports easy swapping or modification of components. Developers can add or replace features with minimal impact on the other parts of the application, facilitating smooth integration of new functionalities.
5. **Improved Testability**: MVC's modular structure enables testing of individual components without having to execute the entire application, which enhances testing efficiency and helps identify issues early.
6. **Reusable Components**: MVC encourages the development of reusable components. Views and Models, for instance, can often be repurposed in other applications, improving development efficiency and consistency across projects.

Disadvantages of MVC:

1. **Increased Complexity**: The MVC architecture can introduce additional complexity, as developers need to manage the relationships and data flow between Models, Views, and Controllers.
2. **Challenging for Large Applications**: As the application scales, the number of Model, View, and Controller files grows significantly. Without a well-structured organization, managing and navigating these files can become cumbersome, impacting maintainability.

3. **Higher Learning Curve for New Developers**: For developers unfamiliar with MVC, the structure may be challenging to grasp at first, especially in understanding how each component interacts and the conventions that must be followed.
4. **Potential for Performance Overhead**: In applications where each component frequently communicates with the others, there can be performance overhead due to the additional layers of abstraction. This may require optimization to maintain efficient performance.
5. **Increased Initial Development Effort**: Setting up an application in MVC involves creating and configuring separate components, which can require more initial development time and effort compared to simpler, monolithic architectures.

1.2. Reasons for Choosing MVC over Layered Architecture

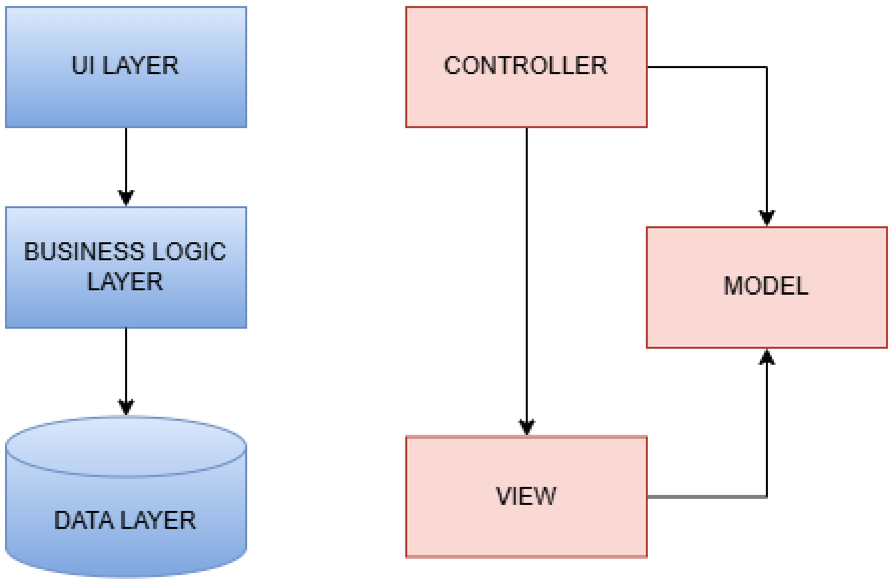


Figure 25: Difference between 3-tier and MVC architecture

After analyzing the MVC and Layered Architecture models, the team opted for MVC for the following reasons:

- ***Easy Task Allocation***: The MVC architecture divides the application into three main components - Model, View, and Controller. This separation simplifies task distribution within the team, enabling each member to focus on a distinct component without overlapping responsibilities. Unlike the Layered Architecture, where each layer depends sequentially on the lower layers, MVC allows for clear task delineation, reducing interdependence among team members.

- ***Concurrent Development***: MVC supports concurrent development as the Model, View, and Controller can be developed independently, promoting collaboration among multiple developers. This is especially advantageous for medium-scale projects like the SPSS system, where parallel workstreams can reduce development time. In contrast, the Layered Architecture's dependency chain between layers makes simultaneous work challenging, as testing upper layers often requires completed lower layers.
- ***Support from Frameworks and Libraries:*** MVC is widely supported by a variety of frameworks and libraries, such as ASP.NET, AngularJS, and Ruby on Rails, making it easier to adopt and implement for modern web applications. This broad support accelerates development and provides readily available resources, community support, and documentation. While the Layered Architecture has its advantages in enterprise-level applications, MVC's popularity makes it an efficient choice for smaller, adaptable projects like SPSS.
- ***Adaptability for Project Scale***: The SPSS project is a medium-sized system, where MVC's structure is particularly suitable. MVC offers flexibility by allowing components to be developed and tested individually, which simplifies management and reduces complexity. In a Layered Architecture, the tight dependency among layers could complicate updates or modifications, which is less desirable for a system that requires agility.

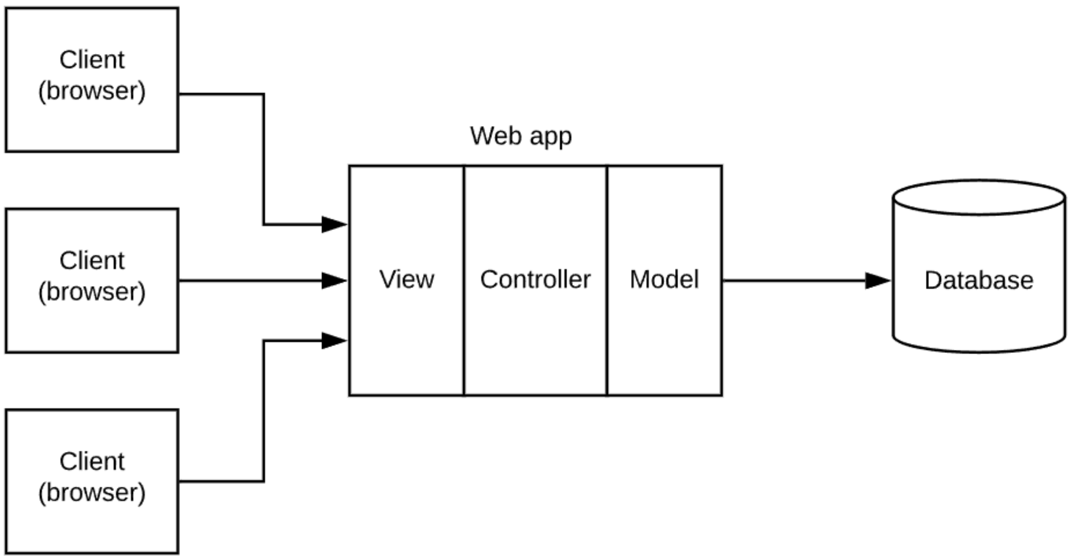


Figure 26: Web MVC Architecture

- ***Learning and Practical Application***: Given that MVC is a commonly used architecture pattern in web development, implementing it in the SPSS project is an excellent opportunity for the team to deepen their skills. This familiarity with MVC prepares team members for future projects, as many organizations utilize MVC for its balance of separation and integration of concerns.

- ***Scalability and Flexibility***: MVC allows for straightforward scalability and adaptation to new requirements. Each component (Model, View, Controller) can be modified independently as new features or changes are introduced. This is particularly beneficial for projects expected to evolve over time, as the SPSS system might. The Layered Architecture can also be scalable, but often at the cost of complex refactoring when requirements change.
- ***Efficient Handling of Data and User Interaction***: In MVC, the Controller acts as a mediator between the user input (View) and the data (Model), handling requests and passing data seamlessly. This clear interaction path enhances responsiveness and user experience. In a Layered Architecture, user requests might need to pass through multiple layers, which can introduce latency and increase processing complexity.

By selecting MVC over Layered Architecture, the team aligns the SPSS project's goals with an architecture that supports efficient development, task separation, adaptability, and real-time user interaction - qualities that are essential for a responsive and scalable application.

2. Architectural Diagram and Deployment Diagram

2.1. Architectural Diagram for the overall design of HCMUT-SSPS system

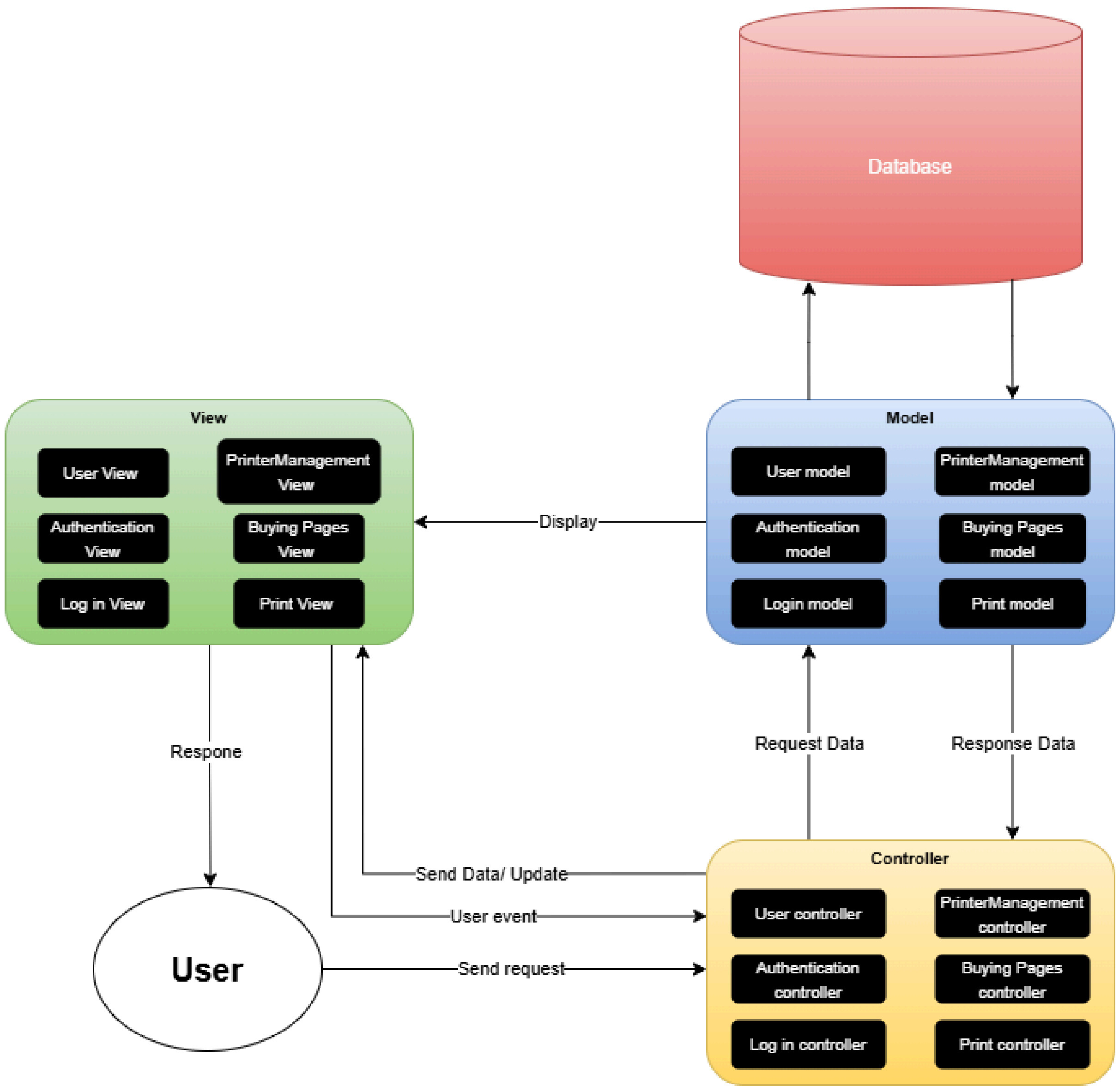


Figure 27: Architectural diagram

Presentation Strategy

The presentation strategy for the HCMUT-SSPS system emphasizes providing a user-friendly and intuitive interface through distinct View components. Core screens such as User View, Authentication View, Log in View, and Printer Management View allow students to easily navigate and access functions like account management, printing services, and page purchasing. By clearly delineating each interface, the system enhances user experience and reduces the time required to locate information. Using frontend frameworks like React or Angular would further streamline the development of these Views due to their flexibility and efficiency.

Data Storage Approach

In designing the data storage strategy for the HCMUT-SSPS system, the team is undertaking a comparative evaluation of **SQL** and **NoSQL** databases to identify the most suitable solution. The SQL option under consideration is **MySQL**, a well-established relational database management system (RDBMS), whereas **MongoDB** represents the NoSQL category as a document-oriented database. Below, we outline the primary advantages and limitations of each approach, followed by a conclusion based on their alignment with the system's data requirements.

SQL (MySQL)

- **Advantages**:
    + **Data Integrity and ACID Compliance**: MySQL ensures adherence to ACID properties (*Atomicity, Consistency, Isolation, and Durability*). Thereby providing a high degree of data reliability, essential for applications requiring precise transactional integrity.
    + **Structured Data Management**: As a relational database, MySQL is optimized for handling structured data within a well-defined schema. It is particularly effective for applications that require complex join operations and precise data relationships.
    + **Mature Ecosystem and Tooling**: MySQL, as a widely adopted SQL database, benefits from a robust ecosystem of tools, extensive documentation, and a supportive community. This maturity facilitates integrations, maintenance, and reliable performance monitoring.
- **Limitations**:

  + **Scalability Constraints**: MySQL typically relies on vertical scaling, which may not be optimal for applications that need to scale horizontally. This limitation can pose challenges when handling rapidly expanding data volumes or data that require distributed storage across multiple nodes.
  + **Schema Rigidity**: MySQL's strict schema enforcement can hinder flexibility, as altering table structures can be complex and time-consuming. This constraint may become a disadvantage in systems where data requirements are dynamic and subject to frequent changes.

NoSQL (MongoDB)

- **Advantages**:
  + **Horizontal Scalability and High Availability**: MongoDB is designed with horizontal scaling capabilities, allowing it to distribute data across multiple servers (sharding). This design is well-suited for handling large and continuously growing datasets in distributed environments.
  + **Schema Flexibility and Agility**: MongoDB's schema-less structure permits flexible document-based storage, making it ideal for storing heterogeneous data that may not conform to a rigid schema. This flexibility is advantageous in rapidly evolving systems where data models may change frequently.
  + **Optimized for Large Datasets and High-Performance Read/Write Operations**: MongoDB's document-oriented storage architecture enables it to efficiently handle high-velocity data and large datasets. Its architecture facilitates fast read and write operations, enhancing performance in applications that process substantial volumes of unstructured or semi-structured data.
  + **Alignment with Agile Development Practices**: MongoDB's adaptability to store complex, hierarchical data structures directly within documents aligns well with agile development methodologies, supporting rapid prototyping and iterative development cycles.
- **Limitations**:
  + **Potential for Reduced Consistency**: MongoDB, like many NoSQL databases, operates on the principles of eventual consistency, prioritizing availability and partition tolerance. As a result, it may not provide the same level of transactional integrity as MySQL, which could be a drawback for applications requiring strict data consistency.

+ **Limited Support for Complex Join Operations**: Unlike relational databases, MongoDB is less suited for complex querying involving multiple collections (equivalent to tables). This limitation could impact analytical workloads that require intricate data joins or aggregations.

Conclusion

Based on the comparative analysis, MongoDB appears more suitable for the HCMUT-SSPS system’s anticipated data needs. The system is expected to handle diverse, potentially unstructured data at a significant scale, making MongoDB’s schema flexibility, horizontal scalability, and efficient handling of large datasets advantageous. Consequently, the team is inclined to proceed with MongoDB in the subsequent stages of implementation, recognizing that its adaptability will support both the system's current requirements and its projected growth.

API Management

API management in the HCMUT-SSPS system will follow **RESTful standards**, ensuring seamless and structured communication between **Controller** and **Model** components, as depicted in the diagram. RESTful principles enable the APIs to provide well-defined endpoints that facilitate data retrieval and updates across different system components. Each endpoint is organized around specific resources, aligning with HTTP methods such as GET, POST, PUT, and DELETE, which handle operations like fetching user profiles, updating printer settings, or logging print activities.

To secure these APIs, the system will implement robust authentication protocols, such as **JWT (JSON Web Token)** or **OAuth2**, to ensure that only authorized users can access sensitive data and perform specific actions. This security layer prevents unauthorized access and enhances data integrity across the **Controller** and **Model** layers.

Furthermore, the RESTful standards promote **stateless interactions**, meaning each API request from the client to the server is independent, carrying all necessary information without relying on previous interactions. This approach allows for easier scalability and faster response times, as no session state is stored on the server. The APIs will be further optimized to minimize latency, employing techniques such as caching for frequently requested data and reducing payload sizes, which will enhance overall performance and user experience in real-time interactions.

List of Models in the Diagram:

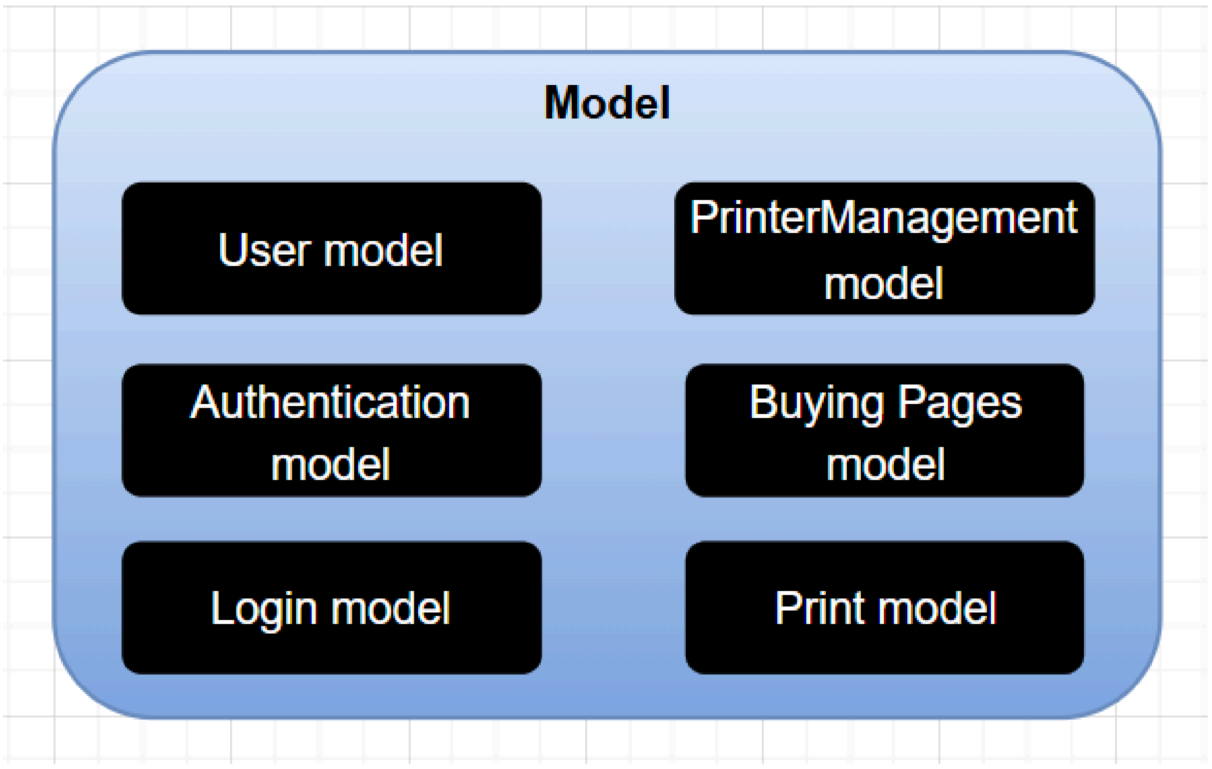


Figure 28: List of Models

- **User Model** - Manages information and data related to users.
- **Authentication Model** - Handles functions related to user authentication.
- **Log in Model** - Manages data associated with user login activities.
- **Printer Management Model** - Manages information and the status of printers.
- **Buying Pages Model** - Processes information related to purchasing additional printing pages.
- **Print Model** - Manages data related to printing requests.

These models support various functionalities of the HCMUT-SSPS system.

List of Views in the Diagram:

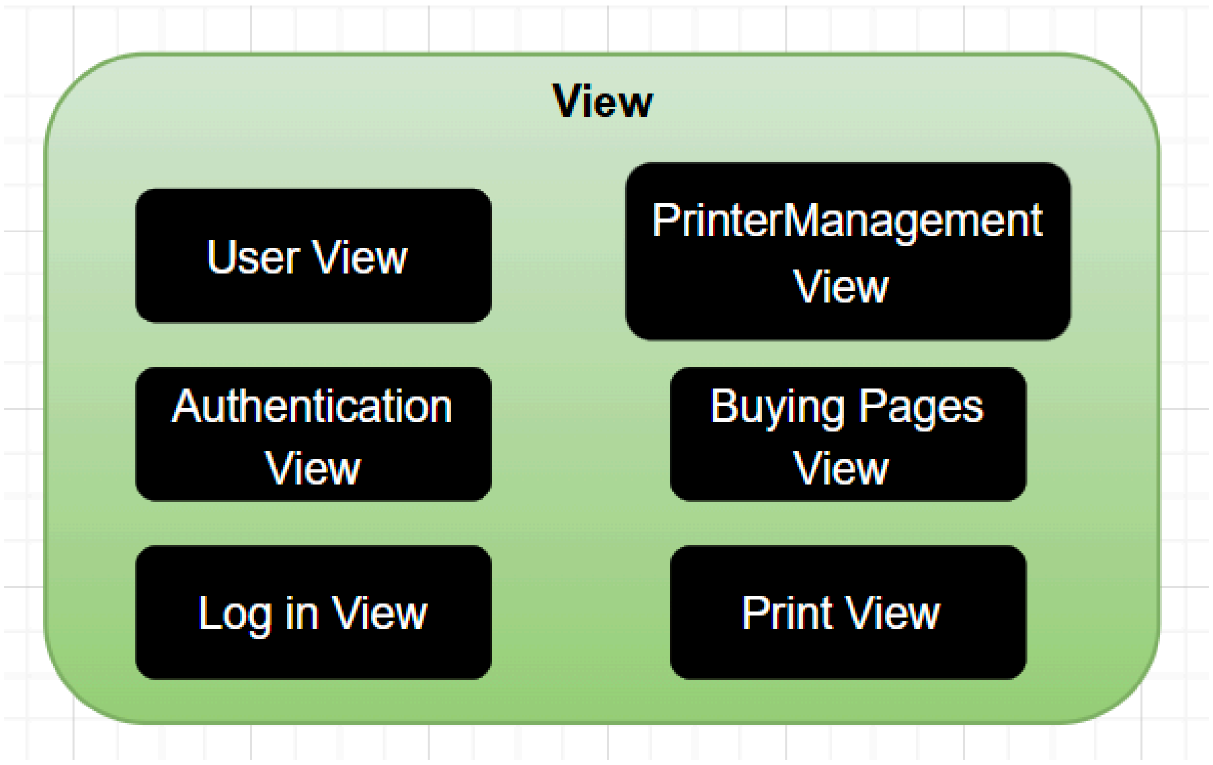


Figure 29: List of Views

- **User View** - Displays information and interface elements related to user data.
- **Authentication View** - Interface for user authentication.
- **Log in View** - Interface for user login to the system.
- **Printer Management View** - Interface for managing printer details and status.
- **Buying Pages View** - Interface for purchasing additional printing pages.
- **Print View** - Interface for executing and managing print requests.

Each of these Views is designed to provide user access to different functionalities of the HCMUT-SSPS system.

List of Controllers in the Diagram:

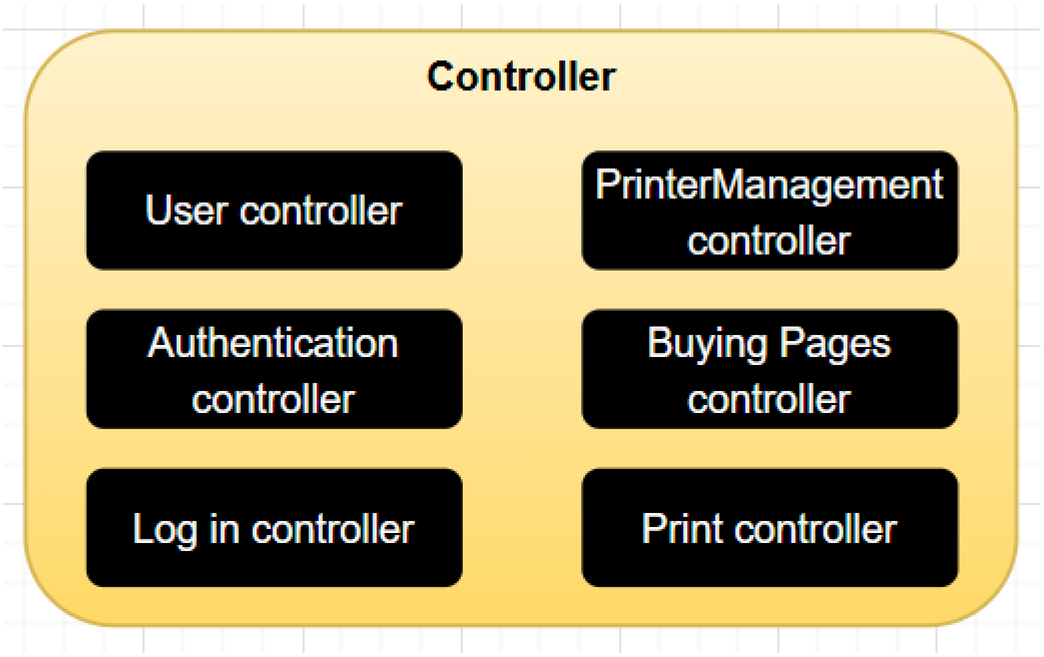


Figure 30: List of Controllers

- **User Controller** - Controls functionalities related to user management.
- **Authentication Controller** - Manages activities related to user authentication.
- **Log in Controller** - Handles requests related to user login.
- **Printer Management Controller** - Controls printer management processes.
- **Buying Pages Controller** - Manages requests related to the purchase of additional printing pages.
- **Print Controller** - Controls and manages print request operations.

The Controllers coordinate with the Views and Models to ensure smooth and efficient system operations within the HCMUT-SSPS.

2.2. Deployment Diagram for the overall design of HCMUT-SSPS system

2.2.1 Deployment Diagram

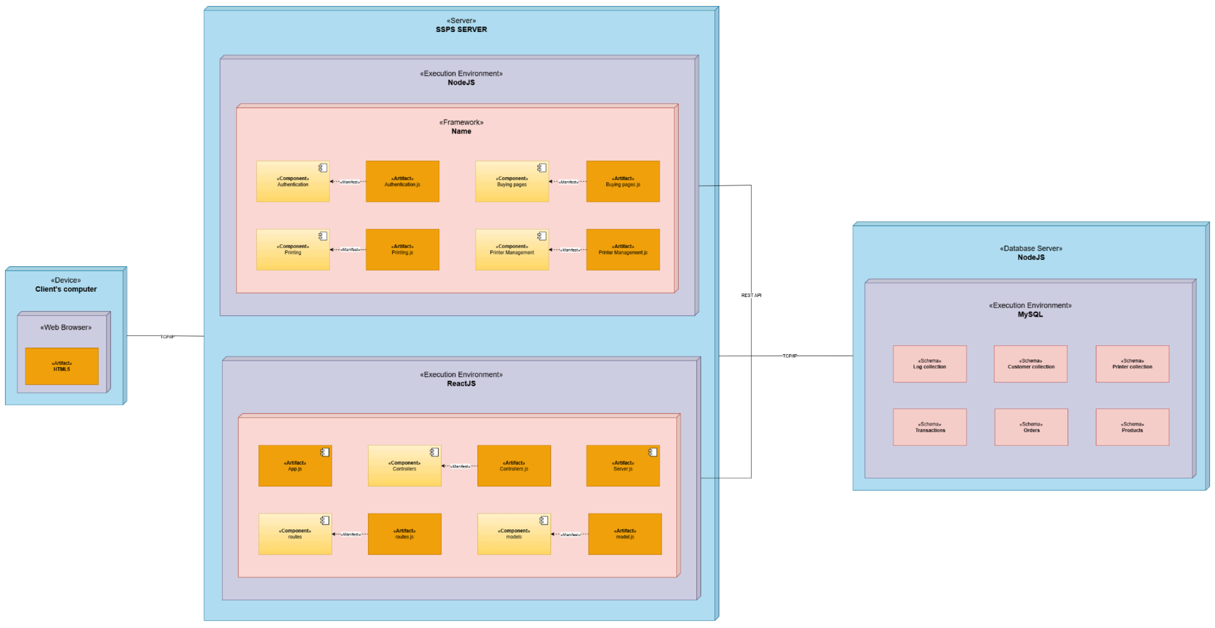


Figure 31: Deployment diagram

2.2.2. Deployment Diagram for description

The deployment diagram for the Student Smart Printing Service (SSPS) system illustrates the allocation of components across distinct hardware and software environments. This architecture is structured into three primary nodes: **Client's Computer**, **SSPS Server**, and **Database Server**. Each node hosts specific components and execution environments, contributing unique roles to fulfill the overall system functionality.

Client's Computer

Role: The Client's Computer serves as the user interface, facilitating interactions between the user and the SSPS system.

- **Components**:
  - **Web Browser**: Acts as the main access point, allowing users to connect to the SSPS system and initiate interactions such as logging in, managing print jobs, and purchasing additional pages.
  - **HTML5 Artifact**: Represents the front-end interface rendered within the web browser, providing users with a responsive and accessible application experience.

- **Communication Protocol**: The Client's Computer connects to the SSPS Server via the **TCP/IP** protocol. Through this setup, user requests (e.g., login or print commands) are transmitted to the server, which processes them and returns responses (e.g., printing status updates) for display on the client-side interface.

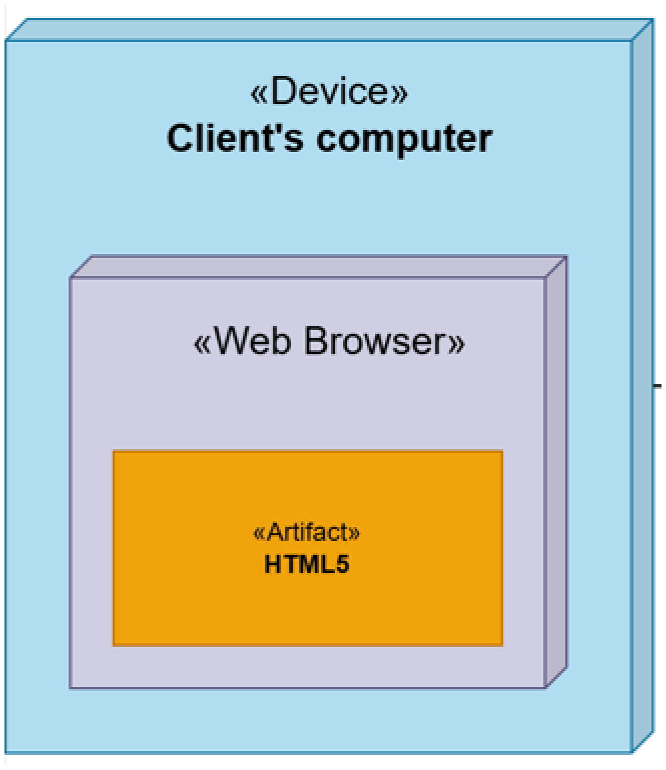


Figure 32: Client's Computer in SSPS system.

SSPS Server

Role: The SSPS Server is responsible for handling application logic, managing user interactions, and coordinating the system's printing functionalities. It is divided into two primary execution environments to support both back-end processing and front-end interface logic.

NodeJS Environment (Back-End)

- **Purpose**: This environment handles the core back-end functionalities, such as user authentication, print job management, and page purchasing processes.
- **Framework**: The back-end processing leverages the **Express.js** framework, which is instrumental in developing RESTful APIs and managing routing efficiently.
- **Components**:

    + **Authentication Component**: Implements user authentication processes, ensuring secure access to the system, and is represented by `Authentication.js.`
    + **Buying Pages Component**: Manages the purchasing of additional printing pages for users, implemented within `BuyingPages.js.`
    + **Printing Component**: Oversees the processing and management of print jobs, represented by `Printing.js`.
    + **Printer Management Component**: Handles configuration and monitoring of printer status, contained within `PrinterManagement.js`.
- **Implementation**: Each component in this environment is encapsulated in a JavaScript artifact (file) dedicated to handling specific back-end operations. This modular approach enhances code maintainability, allowing each component to perform discrete tasks without interdependency.

ReactJS Environment (Front-End)

- **Purpose**: The ReactJS environment manages client-side logic, including data routing and front-end data modeling, to support a dynamic and interactive user interface.
- **Framework**: The front-end environment utilizes **Material UI** alongside ReactJS to design a cohesive, accessible, and responsive user interface.
- **Components**:
    + **Controllers Component**: Encapsulates front-end control logic, providing business logic processing for each module, contained in `Controllers.js`.
    + **Routes Component**: Manages application routing, defining pathways between different views, implemented within `Routes.js`.
    + **Models Component**: Defines front-end data models to interact with and represent the server-side data, implemented as `Models.js`.
    + **App Component**: Acts as the main entry point for the ReactJS application, configuring Express and connecting routes, represented by `App.js`.
    + **Server Component**: Facilitates communication between the front-end and back-end environments, implemented in `Server.js`.
- **Implementation**: Each front-end component is implemented as a modular JavaScript file, providing a structured and organized approach to managing the client-side application logic.
- **Communication**: The SSPS Server establishes communication with the Database Server through **REST API** over the TCP/IP protocol. This RESTful architecture supports modular interactions between the server and the database, ensuring data consistency and efficiency in data handling operations.

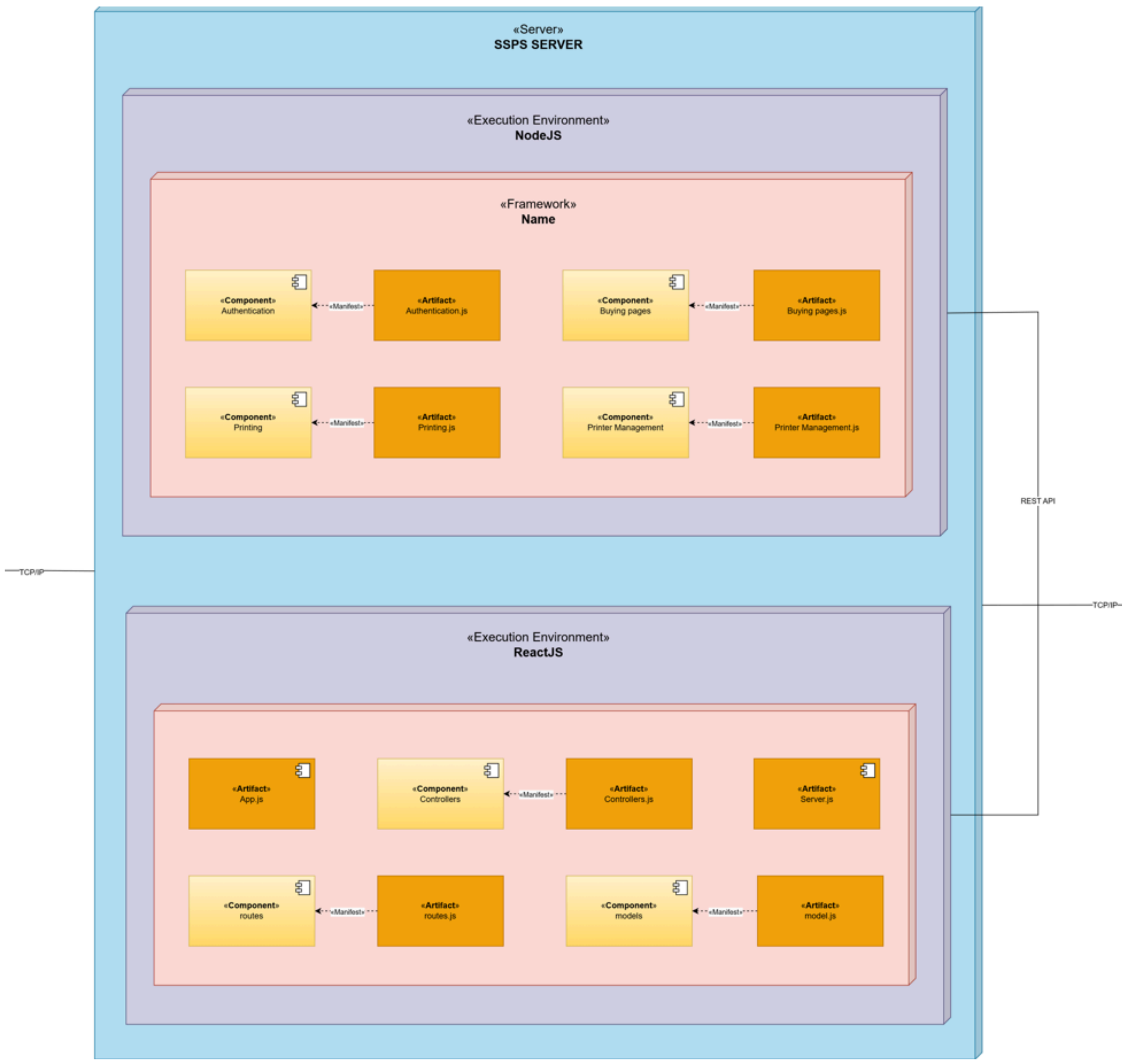


Figure 33: SSPS Server in SSPS system.

Database Server

Role: The Database Server is responsible for storing and managing all system data, including user profiles, printing logs, printer configurations, and transactional data.

- **Environment**: The SSPS system utilizes **MongoDB Atlas Cloud Database** as its primary data storage solution, chosen for its scalability, high availability, and remote accessibility.
- **Database Features**:
  + **Remote Connection**: The MongoDB Atlas cloud environment facilitates remote data management, removing the necessity for on-premises storage and enabling distributed access for enhanced scalability.

+ **Data Management**: MongoDB Atlas provides comprehensive data management tools, allowing administrators to view, query, and delete data directly from the online interface, supporting flexible data manipulation.
+ **Schemas**: The database organizes data into various collections, including `Log Collection`, `Customer Collection`, `Printer Collection`, `Transactions`, `Orders`, and `Products`. Each collection is structured to support specific data types required by the SSPS system and to maintain relational consistency among data elements.

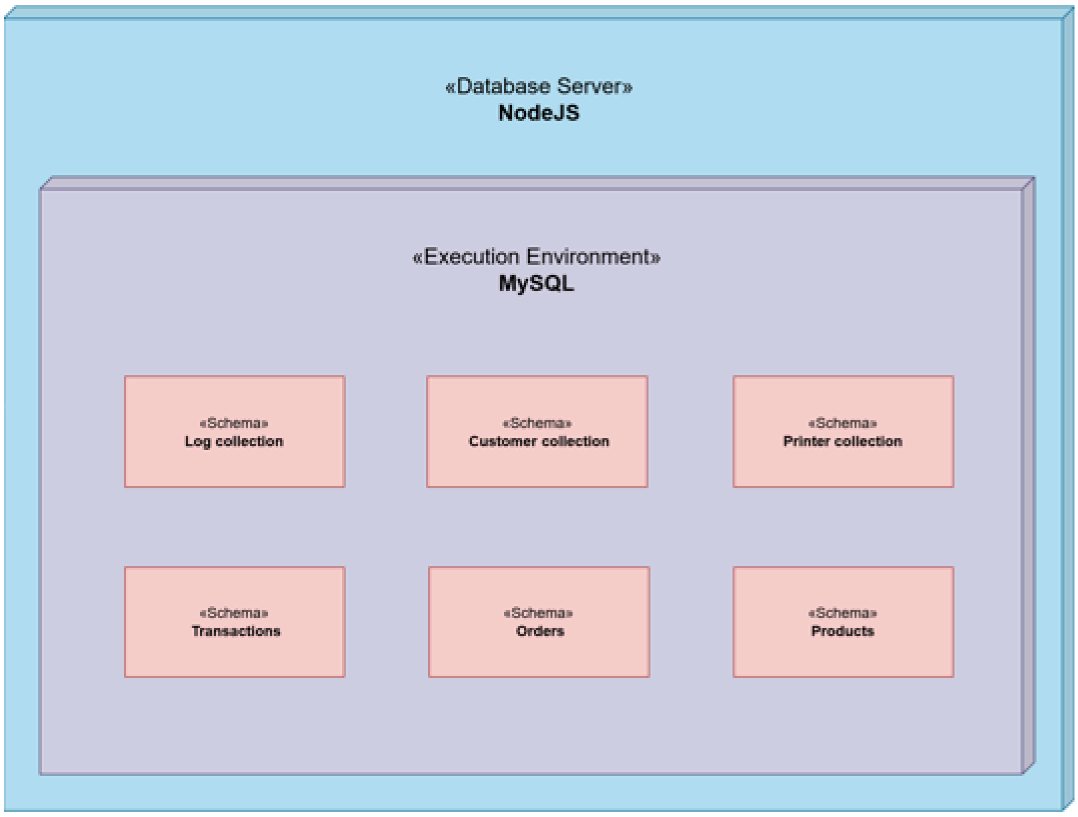


Figure 34: Database Server in SSPS system.

3. Component Diagram

3.1. Module: Authentication

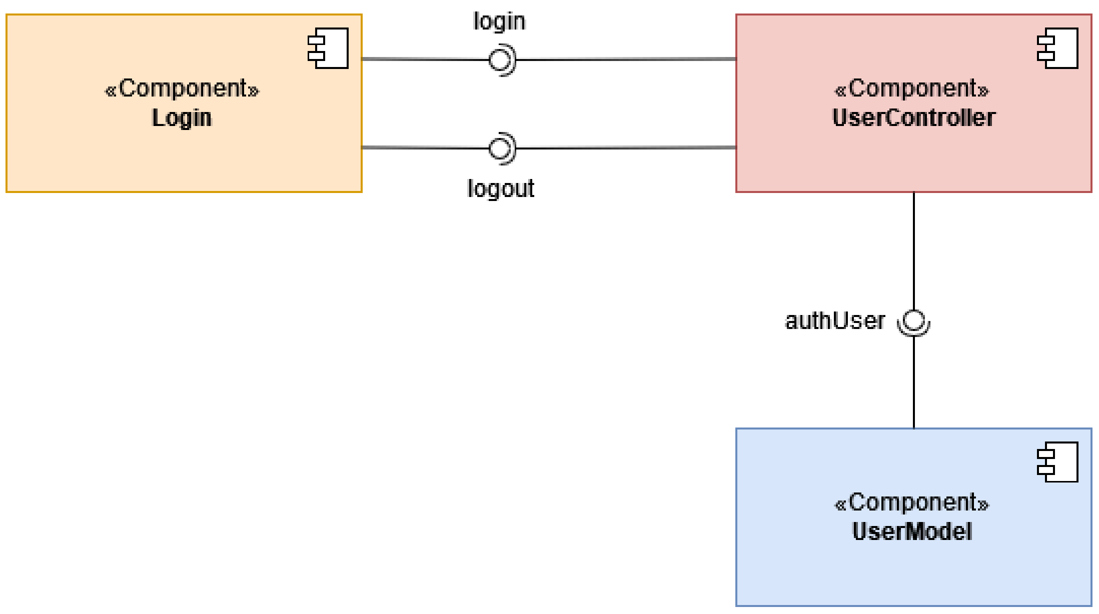


Figure 35: Component Diagram for Authentication Module

Description:

In the component diagram for the Authentication module, the view layer includes a component called **Login** providing the login interface for users to login to their accounts. The controller layer includes a component called **UserController** providing the methods for authenticating the accounts. The model layer includes a component called **UserModel** storing and performing queries on data related to user accounts (authentication).

3.2. Module: Printing

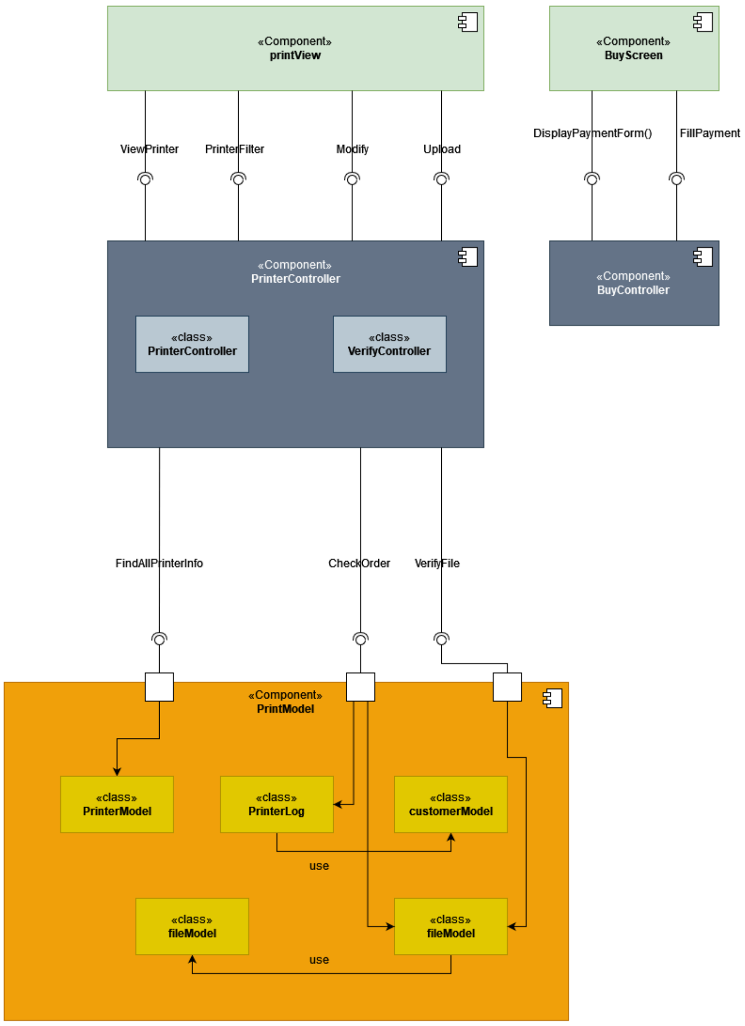


Figure 36: Component Diagram for Printing Module

Description:

In the component diagram of the printing module, the view layer includes two components called **printView -** which provides the interface for the user to perform the operations such as choosing a printer, switching to buying printing pages - and **BuyScreen -** which provides the interface for the user to purchase printing pages. The controller layer includes two components, **PrinterController** and **BuyController**. The **PrinterController** component contains two classes, **PrinterController** and **VerifyController** providing the methods for searching information and choosing the printer. The **BuyController** component provides the methods for creating the buying printing pages form. The model layer includes a component called **PrintModel** for storing and performing operations on printing such as get printer information, verify and store the document file for printing.

3.3. Module: Printer Management

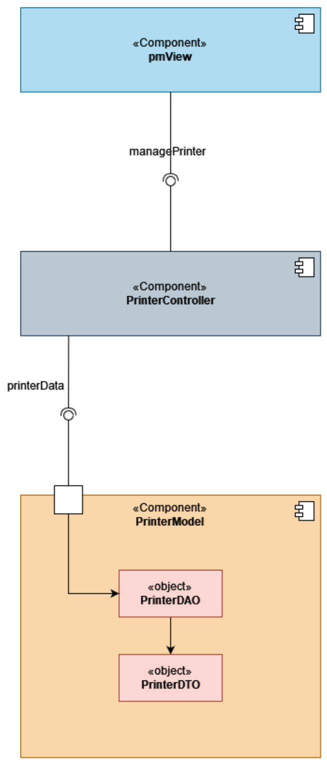


Figure 37: Component Diagram for Printer Management Module

Description:

In the component diagram of the Printer Management module, the view layer includes one component called **pmView** providing the interface for SPSO to manage the printers (add, enable, disable printers). The controller layer includes one component called **PrinterController** providing the methods for managing the printers such as add, enable, and disable printers. The model layer includes one component called **PrinterModel** for storing and performing actions related to managing printers such as add, enable and disable printers. Inside the **PrinterModel** contains 2 objects called **PrinterDAO** (to perform operations on the database) and **PrinterDTO** (to store the printer's data).

The interfaces include:

- **managePrinter**: provides the methods for managing the printers (add, enable, disable).
- **printerData**: provides the API to retrieve the printer's data for management (add, enable, disable).