Ho Chi Minh City University of Technology

Faculty of Computer Science and Engineering

Software Engineering (CO3001)

Student Smart Printing Service

Report 4 - Task 3

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

Instructors:

PhD. Truong Thi Thai Minh

Completion date: 10th November, 2023

Mục lục

1 Introduction

This Architectural Design Document for the HCMUT-SSPS system is a comprehensive blueprint that utilizes a layered architecture to ensure modularity, scalability, and maintainability. The document details the presentation strategy, data storage approach, and API management, while also incorporating a component diagram to illustrate the structural composition of the chosen modules.

2 Layered Architecture

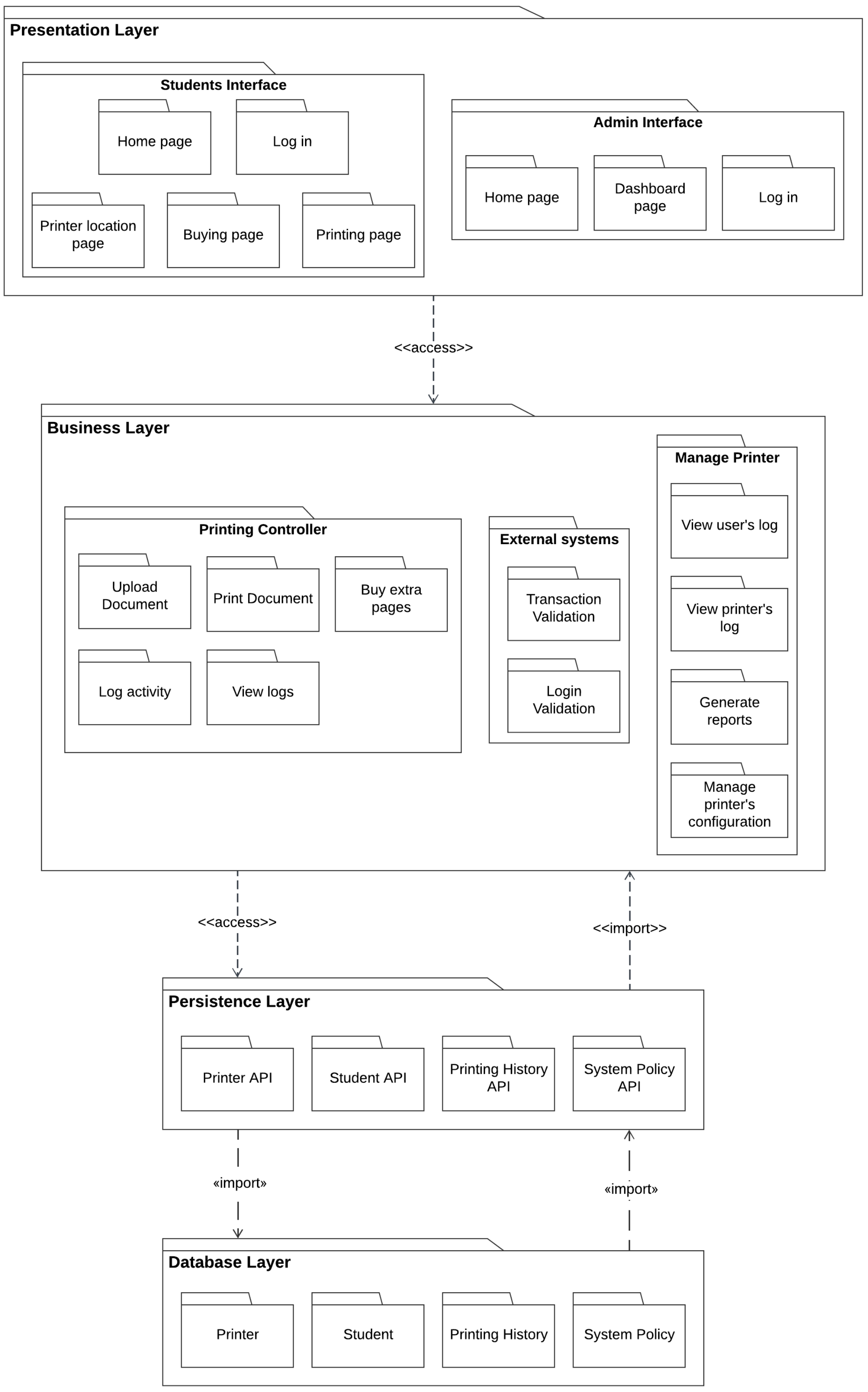
Layered architecture is a design pattern used in software engineering to organize a complex system into discrete, logical layers. Each layer has a specific set of responsibilities and interacts with the layers above and below it through well-defined interfaces or APIs (Application Programming Interface).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

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

Below are detailed specifications of each layer in the architecture.

2.1 Presentation strategy (Presentation layer)

This layer is responsible for user interface and interaction. It handles user's command, display information to the user. There are two separate components: ***Student Interface*** and ***Administration Interface***. Both interfaces share some common elements, such as a ***Home page***, which serves as the initial point of entry for users upon accessing or logging into BKPrint. Additionally, there is a ***Login page***, which varies depending on whether the user is a student or an administrator. The ***Student Interface***, specifically, offers a range of features that cater to the needs of students. It includes a ***printer location page***, enabling students to view the available printers and their current status. This feature is invaluable in helping students find nearby, operational printers, saving time and enhancing convenience. The ***Printing page*** within the Student Interface allows users to upload documents and specify the desired printer and printing properties, ensuring that their printing needs are met efficiently. In the event that a student's paper supply is running low, the interface smoothly transitions to the ***Buying page***, where students can effortlessly top up their paper balance to meet their specific requirements. On the other hand, the ***Administration Interface*** is tailored for administrators, equipping them with powerful tools to manage the BKPrint system. The ***Dashboard page*** is the



Hình 1: Layered architecture of the whole system

command center for administrators, offering a comprehensive overview of user activities through print history, providing insights into printer quantity and status, and allowing administrators to make real-time changes to printer settings as needed. Furthermore, administrators can export periodic reports from this interface, which is invaluable for tracking system performance and usage trends. In summary, both User Interface and Interaction layer acts as a bridge between users and the system's functionality, enhancing the overall efficiency and effectiveness of printing services.

2.1.1 Students interface

The students interface contains 5 pages:

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

2.1.2 Admin interface

The admin interface contains 3 pages:

- Home page
- Log in
- Dashboard page

Details of the UI are in this section: User Interface

2.2 Business logic layer

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

2.2.1 Printing controller

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

- Upload documents: When students upload their documents, the **Printing controller** will accesses the **Persistent Layer** and asks **System Policy API** to import **System Policy** from **Database** to get the list of permitted file types. Then, it will check if the type of the document file is in that list and decide whether to accept or reject the file.

- Print documents: The **Printing controller** will get the list of printers from **Database layer** through **Printer API** from **Persistence layer** to let student choose a printer for printing document. Then, it will let student specify printing properties such as paper size, the number of pages to be printed, one/double sided and the number of copies. The **Printing controller** also maintains a queue of print jobs, which allows multiple users to send their print requests. It manages the order in which documents are printed. After a print job is done, the **Printing controller** will update the page balance of **Student** and the printing log in **Printing History** in **Database layer** through **Student API** and **Printing History API**, respectively, in the **Persistence layer**.

- Buy extra pages: When student's page balance is not enough for printing document and student is required to buy more pages, or when student want to buy more pages although their page balance is still enough for printing, the **Printing controller** take the number of pages student need to buy, calculating the price and then delegating the transaction to online payment services, which are external systems.

- Log activity: When a printing job or a transaction occurs, it is saved to the database by the **Printing controller** through the APIs in **Persistence layer**. For example, after the transaction is finished, the **Printing controller** will update the transaction log of **Student** in **Database layer** through **Student API** in **Persistence layer**.

- View logs: **Printing controller** gets the printing logs and transaction logs from **Student** and **Printing History** in **Database layer** through **Student API** and **Printing History API** in **Persistence layer** and haves that data display to user from the **Presentation layer**.

2.2.2 External systems

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

- Transaction validation: This belongs to online payment services which make transactions when student buys more pages.

- Login validation: This belongs HCMUT_SSO authentication service which authenticates users before they use the HCMUT_SSPS system.

2.2.3 Manage Printers

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

- View user's log: The SPSO adminstrator can view the printing history of all students. This package will get the printing logs from **Student** in **Database layer** through **Student API** in **Persistence layer**.

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

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

2.3 Data storage approach (Database layer)

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

2.3.1 Printer

The printer schema contains the following attributes

Attribute	**Data Type**	**Description**
Printer ID	String	This represents a unique code for the printer
Printer Brand	String	This represents the brand of the printer
Printer Name	String	This represents the name of the printer
Location	Object	An object containing **Building** and **Room Number** attributes
Building	String	The building where the printer is located
Room	String	The room number where the printer is located
Status	Boolean	State of the printer (enable or disable)
Printed Pages	Integer	This field represents the number of printed pages

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

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

2.3.2 Student

The student schema contains the following attributes

Attribute	**Data Type**	**Description**
Student ID	String	This represents a unique identifier for the student
Student Name	String	The name of the student
Student Email	String	The email of the student
Student Faculty	String	The faculty of the student
Remaining Pages	Integer	The number of remaining pages for the student
Transaction History	Object	A list of object which containing the price, purchased pages and the time of the transaction
Time	DateTime	The time that the transaction was conducted
Price	Float	The cost associated with the purchase
Purchased pages	Integer	The number of pages purchased
Printing History	Object	A list of object which containing printing details
Filename	String	The name of the file printed
Time	DateTime	The time of the printing job

Printed pages	Integer	Show the number of pages that was used for the printing job
Paper type	String	Type of papers that was used for the printing job (A3, A4, A5, etc)
Location	String	Represents the building of the used printer

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

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

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

```
"student":{
    StudentID: "2152788,
    StudentName: "Ta Ngoc Nam",
    Email: "nam.ta8989@hcmut.edu.vn",
    Faculty: "Ky Thuat May Tinh",
    RemainingPages: 20,
    TransactionHistory: {
        Time:
        Price:
        PurchasedPages:
    },
    PrintingHistory: [
        {
            Time: 00:00, 22/10/2023,
            FilenName: "file1.pdf",
            PrintedPages: 15,
            PaperType: "A4",
            Location: "A4",
        },
        {
            Time: 07:00, 23/10/2023,
            FileName: "file2.pdf",
            PrintedPages: 20,
            PaperType: "A5",
            Location: "B1",
        }
    ]
    TransactionHistory: [
        {
            Time: 09:00, 22/10/2023,
            Price: 50000
            PurchasedPages: 100
        },
        {
            Time: 10:00, 23/10/2023,
            Price: 20000
            PurchasedPages: 40
        }
    ]
}
```

2.3.3 Printing History

The printing history schema contains the following attributes

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

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

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

2.3.4 System Policy

The system policy history schema contains the following attributes

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

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

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

2.4 API management (Persistence layer)

API stands for application programming interface, which is a set of definitions and protocols for building and integrating application software. APIs let your product or

service communicate with other products and services without having to know how they're implemented.
According this HCMUT_SPSS system, there are 3 main API services are used

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

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

2.4.1 Printer API

This object API has the following methods

Method	**Passing Parameter**	**Description**
getPrinterBrand	String (ID)	Pass the printer ID to get the information of printer's brand from the database
getPrinterName	String (ID)	Pass the printer ID to get the information of printer's name from the database
getPrinterStatus	String (ID)	Pass the printer ID to get the information of printer's status from the database
getPrinterBuilding	String (ID)	Pass the printer ID to get the information of printer's building location from the database
getPrinterRoomNumber	String (ID)	Pass the printer ID to get the information of printer's room number location from the database
getPagePrinted	String (ID)	Pass the printer ID to get the information of how many page has been printed by this printer from the database
addNewPrinter	Object (Printer Information)	Create a new printer and add it to the database

changePrinterStatus	String (ID)	Pass the printer ID to change the printer status and also update that status to the database

This API will be called in some following cases

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

2.4.2 Student API

This object API has the following methods

Method	**Passing Parameter**	**Description**
getStudentName	String	Pass the student ID to get the information of the student's name from the database.
getStudentEmail	String	Pass the student ID to get the information of the student's email from the database.
getStudentFaculty	String	Pass the student ID to get the information of the student's faculty from the database.
addPrintingActivity	Object (Printing Activity)	Create a new printing activity and add it into the printing history list.
addTransactionActivity	Object (Transaction Activity)	Create a new transaction activity and add it into the transaction history list.
getPrintingTime	String (ID), Integer (Index)	Pass the student ID and the index of the chosen printing activity to get the time of that printing activity.

getPrintingFileName	String (ID), Integer (Index)	Pass the student ID and the index of the chosen printing activity to get the filename of that printing activity.
getPrintingPaperNumber	String (ID), Integer (Index)	Pass the student ID and the index of the chosen printing activity to get the information about how many pages have been printed by this printer from the database
getPrintingPaperType	String (ID), Integer (Index)	Pass the student ID and the index of the chosen printing activity to get the type of paper have been printed in this printing work (A3, A4, A5)
getPrintingLocationBuilding	String (ID), Integer (Index)	Pass the student ID and the index of the chosen printing activity to get the information of the printer's building location from the database.
getTransactionTime	String (ID), Integer (Index)	Pass the student ID and the index of the chosen transaction activity to get the time of that transaction.
getTransactionCost	String (ID), Integer (Index)	Pass the student ID and the index of the chosen transaction activity to get the price that student have to pay for this transaction.
getTransactionPage	String (ID), Integer (Index)	Pass the student ID and the index of the chosen transaction activity to get the number of page that student purchase in this transaction. In default, the paper type is A4 paper.

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

1. Student name (Ta Ngoc Nam)
2. Student ID (2152788)

3. Student email (nam.ta8989@hcmut.edu.vn)
4. Student Faculty (Ky Thuat May Tinh)
5. List of all printing activity
6. List of all transaction

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

2.4.3 Printing History API

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

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

2.4.4 System Policy API

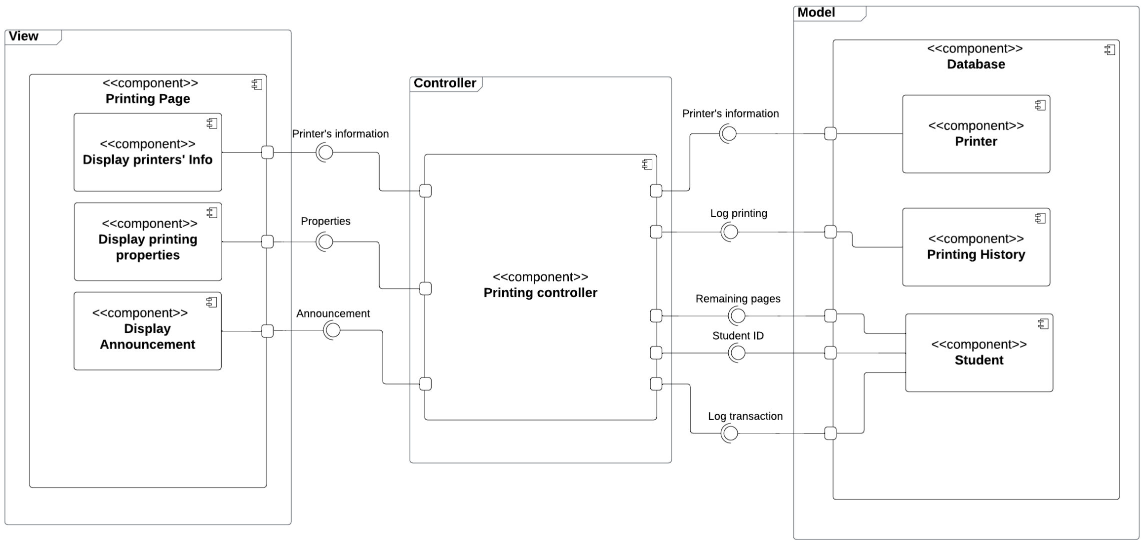
This object API has some following methods

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

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

3 Component Diagram

3.1 "Print Document" module



Hình 2: The *Print Document* Component Diagram

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

- **Display printers' info** component: the section which displays a list of printers and their information such as printer model and location for student to choose. This requires **Printer's information** interface from the component **Printing controller**.
- **Display printing properties** component: the section which allows student to specify printing properties such as paper size, the number of pages to be printed, one/double sided and the number of copies. This requires **Properties** interface from the component **Printing controller**.

- **Display announcement** component: requires **Announcement** interface from the component **Printing controller** and consists of
  - The page which notifies student that their document is printing and reminds student to go to the printer to take the document.
  - The page which notifies student that his/her page balance is not enough and requires he/she to buy more pages.

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

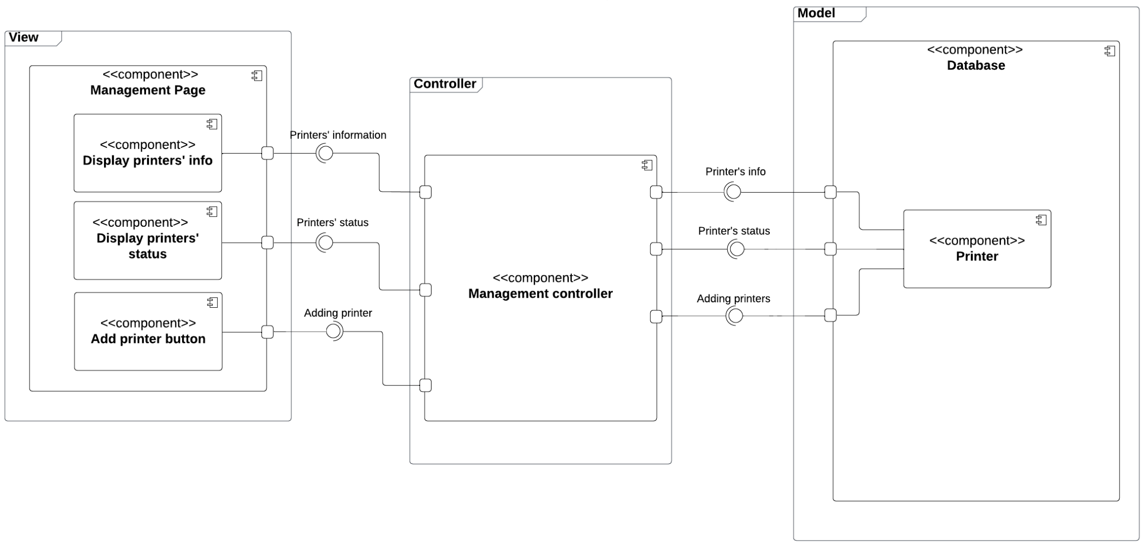
- **Printing controller** component requires **Printer's information** interface from **Printer** component in **Database** to provide for **Display printer's info** component.
- **Printing controller** component requires **Log printing** interface from **Printing History** component in **Database**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

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

3.2 "Manage Printers" module

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

- **Display printers' info** component: the section which displays a list of printers and their information such as printer ID, brand name, printer model, and location for the SPSO administrator to manage. This requires **Printer's information** interface from the component **Management controller**.



Hình 3: The *Manage Printers* Component Diagram

- **Display printers' status** component: the section which displays the current status of each printer and a column which allows the SPSO administrator to change the printer's status. This requires **Printers' status** interface from the component **Management controller**.
- **Add printer button** component: provides **Adding printer** interface for the **Management controller** component to add a new printer to the printers' list.

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

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

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

4 Conclusion

Embracing a layered architecture, focusing on modern UI presentation, adopting robust data storage methods, and emphasizing seamless API management, the design ensures a strong foundation for system development. The inclusion of the component diagram provides a detailed view of the module's architecture, interactions, and dependencies, further enhancing the document.