

Assignment 3: MVC, MVP and MVVM Architectural Pattern

Team member information:

Team Name:	idk
Member 1:	Kritsana Khankaew 6030026021
Member 2:	Kongtap Arunlakvilart 6030033321
Member 3:	Kantorn Chitchuen 6030038521
Member 4:	Melvin Macaranas 6030478021
Member 5:	Yodsavee Lertthirathanaphong 6030482421
Member 6:	David Tumcharoen 6031015221

Objective:

1. To understand the concept of patterns for achieving the separation of concerns in software design
 2. To understand the concept of Model-View-Controller pattern
 3. To understand the concept of Model-View-Presenter pattern
 4. To understand the concept of Model-View-ViewModel pattern
-

Requirement:

1. Python 3.7 or greater
2. wxPython for UI development (<https://www.wxpython.org/>)
3. RxPY for reactive programming (<https://rxpy.readthedocs.io/en/latest/index.html>)

P.S. The program in this assignment is designed to run on Windows, macOS and Linux.

How to submit:

1. Create your new group repository in the class organization with all of your source code
 2. Answer each question in this document
 3. Submit the document with your answers and your repository link in myCourseVille
-

Before we start:

When developing software, usually, the presentation layers (GUI/CLI/etc.) and business logic layers are included. There are many ways to communicate between these layers. The easiest way for a presentation layer is to access business logic directly. Alternatively, you can introduce another layer between these layers which may be better for separation of concerns design principle.

Q1: What is separation of concerns?

หลักการออกแบบที่มีการแบ่งโปรแกรมออกเป็นส่วน ๆ ตาม Concern หรือก็คือการพิจารณาการแบ่งตามหน้าที่การทำงาน เพื่อวางแผนการพัฒนาฟังก์ชันต่าง ๆ ในระบบ เพื่อให้ขั้นตอนการออกแบบและการพัฒนาโปรแกรมต่าง ๆ มีส่วนการทำงานที่ซ้ำซ้อนกันเกิดขึ้นน้อยที่สุดเท่าที่จะเป็นไปได้ ซึ่งอาจทำได้หลายวิธี เช่น Information Hiding คือการ Encapsulate ข้อมูลไว้ภายในอ็อบเจกต์แล้วเรียกการทำงานผ่านเมทอดของคลาสนั้น การออกแบบโปรแกรมโดยอาศัยหลักการ Separation of concerns จะทำให้

การแก้ไขบางส่วนของโปรแกรม เช่น การแก้ไขเฉพาะบางเมทอดของคลาส เกิดผลกระทบต่อเมทอดและคลาสอื่น ๆ น้อยที่สุด

Q2: Do you think that we should access the business logic layers directly from presentation layers? Why?

ไม่ควรทำ ควรมีการเพิ่มชั้นระหว่างกันอย่างน้อย 1 ชั้นเพื่อเป็นการรองรับการเปลี่ยนแปลงของชั้นต่าง ๆ เนื่องจากหากต้องการทำการเปลี่ยนแปลงในชั้นแสดงผล (presentation layer) เช่น เปลี่ยนมุมมอง (view) ใหม่ จะทำให้จำเป็นต้องมีการแก้ไขโปรแกรมในส่วนชั้นตรรกะ (business logic layer) ด้วย เพื่อให้รองรับการทำงานของชั้นแสดงผลที่ถูกแก้ไขไป อีกทั้งยังทำให้เกิดความซับซ้อนของระบบอันเป็นผลมาจากการซ้ำซ้อนของโค้ดด้วย เนื่องจากโค้ดในชั้นตรรกะจะเพิ่มขึ้น จากการที่ต้องเปลี่ยนแปลงไปตามโค้ดของชั้นแสดงผลแต่ละมุมมองที่ต่างกัน

Now, we will setup the development environment for this assignment

1. Install wxPython

```
# if you are using Windows of macOS
$ pip install -U wxPython

# if you are using Linux

# Method 1: build from source
$ pip install -U wxPython

# Method 2: Find binary suited for your distro
# For example with Ubuntu 16.04
$ pip install -U -f
https://extras.wxpython.org/wxPython4/extras/linux/gtk3/ubuntu-16.04
wxPython

# Method 3: Find the package in your distro repository
```

2. Install RxPY

```
$ pip install rx
```

P.S. In some OS, “pip” command is pointed to the pip for Python 2. Anyway, we used Python 3 here so please make sure to use the correct “pip” command since it might be “pip3” in some environment such as macOS.

Next, clone the provided git repository

```
# Change directory to your desired directory
[your directory] $: git clone
https://github.com/2110521-2563-1-Software-Architecture/Assignment-3-MVC-MVP-
MVVM.git
```

MVC: Model-View-Controller

Firstly, we start with MVC pattern. In this pattern, three components are presented including Model, View and Controller. The model includes all of your business logic, the view includes all of your program presentations and the controllers are places where your views interact with the models. Note that the implementation of MVC has many variants in addition to the way used in this assignment.

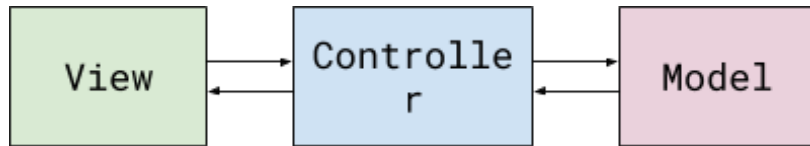


Figure: The MVC pattern and the interaction between each layer

We will create the simple note taking application as shown below

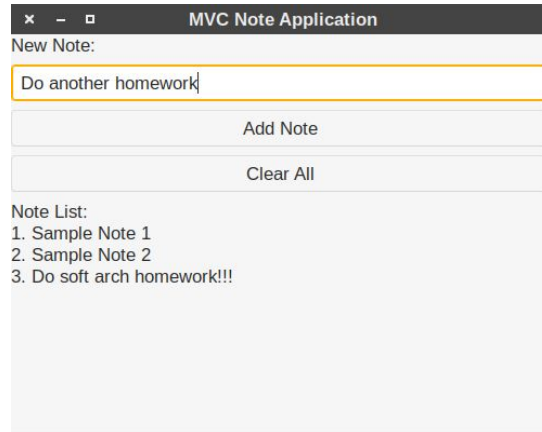


Figure: The simple MVC Note Application which we are going to create

Change directory to the “mvc” folder

```
# From assignment root
$ cd mvc
```

You will see these structure

```

mvc
├── main.py
├── mvc
│   ├── controllers
│   │   ├── __init__.py
│   │   └── main_controller.py
│   ├── __init__.py
│   ├── models
│   │   ├── entities
│   │   │   ├── __init__.py
│   │   │   └── note.py
│   │   ├── __init__.py
│   │   ├── repositories
│   │   │   ├── __init__.py
│   │   │   └── note_repository.py
│   └── views
│       ├── base_view.py
│       ├── __init__.py
│       └── main_view.py
  
```

What we already implemented for you is all the presentation parts and business logic parts, the objective task is to implement the controller and connect the views and models through the controller.

To run the application

```
$ python main.py
```

Also make sure that the “python” command pointed to Python 3. If it pointed to Python 2, use the command “python3” instead.

Open main_controller.py and implement all missing methods.

Hint: Use an object of type NoteRepository to interact with the business logic.

Q3: How did you make the controller work?

ใน main_controller.py เพิ่ม code ส่วนนี้

```
class MainController(NoteRepository):  
    def __init__(self):  
        super().__init__()  
        pass  
  
    def get_all_notes(self):  
        return NoteRepository.get_all_notes(self)  
        pass  
  
    def add_note(self, note: str):  
        NoteRepository.add_note(self, note)  
        pass  
  
    def clear_all(self):  
        NoteRepository.clear_all_notes(self)  
        pass
```

Next, we will connect our views to the controller. Open main_view.py and implement all missing methods.

Hint: Use an object of type MainController which you implemented in the previous step.

Q4: How did you make the view work?

ใน main_view.py เพิ่ม code ส่วนนี้

```
def on_clear_all_button_clicked(self, e):  
    self.controller.clear_all()  
    self.update_view(self.controller.get_all_notes())  
    pass  
  
def on_add_note_button_clicked(self, e):  
    content = self.note_input.GetValue()  
    self.note_input.SetValue("")  
    self.controller.add_note(content)  
    self.update_view(self.controller.get_all_notes())  
    pass
```

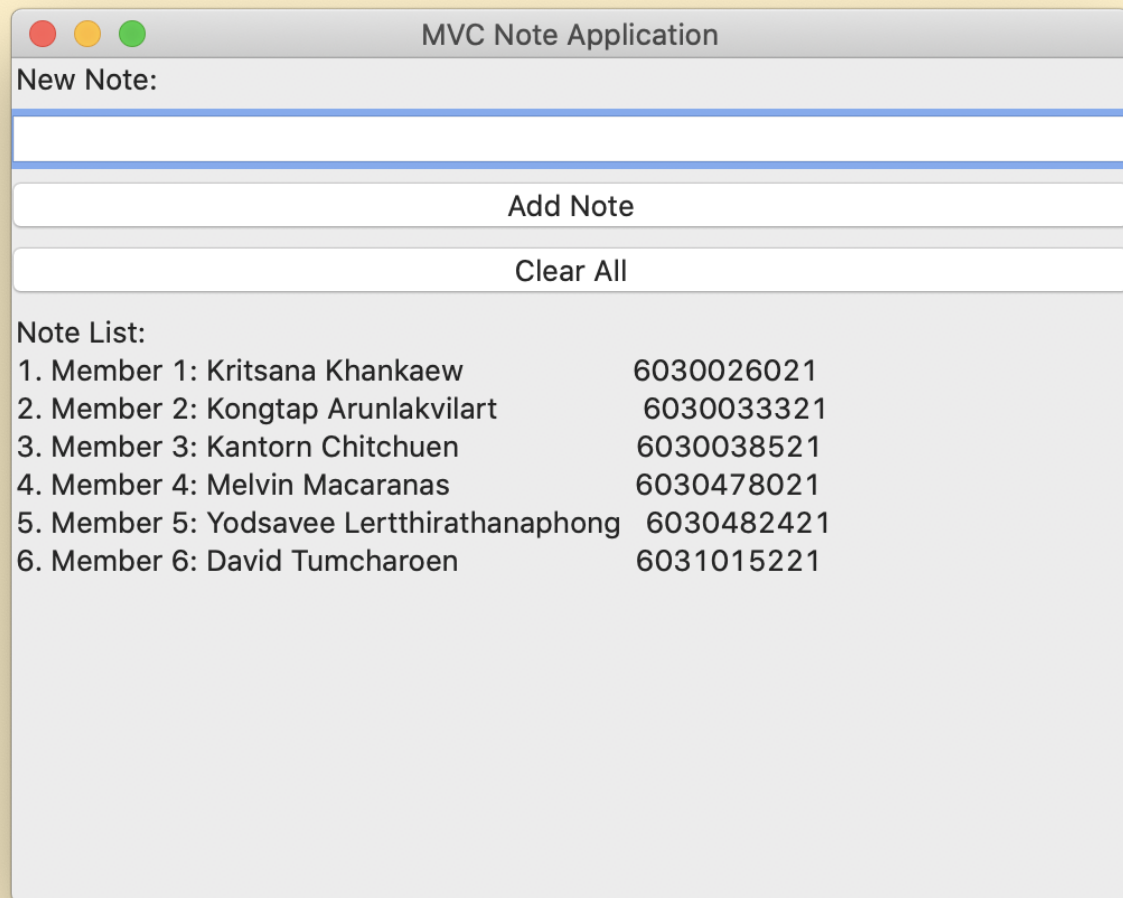
Q5: What is the role of the controller here? Explain it briefly.

หน้าที่ของ controller คือการรับ input มา แล้วเปลี่ยนเป็นคำสั่งสำหรับ model หรือ view

Q6: What are the advantages of MVC pattern?

มีการจัดการเป็นระบบ จึงสามารถทำงานร่วมกันได้ง่ายขึ้น และ ทำให้ develop ได้เร็วขึ้น และ ทำให้สามารถ debug และ update application ได้ง่ายขึ้นอีกด้วย

Q7: Put the screenshot of the MVC Note Application displaying your members' name in each note.



MVP: Model-View-Presenter

Now, we will look into another pattern called "MVP" which is considered as a variant of MVC pattern. In the MVC pattern, the view needs to update itself when the data changes which may not be convenient in a complex application. Instead, we will replace the controller with the presenter and change the way they communicate to each other. In MVP pattern, the presenter will be the object which updates the view instead of the view itself.

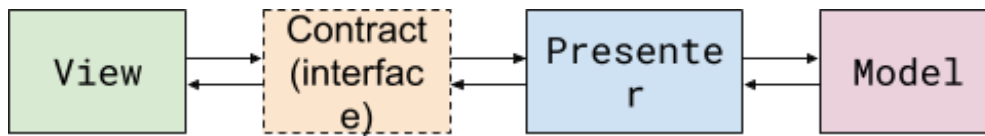


Figure: The MVP pattern and the interaction between each layer

According to the figure, notice that the view and presenter don't directly communicate to each other but through an interface (We use simple class here for this assignment since Python doesn't have the interface).

Q8: In your opinion, why does an interface need to be introduced between the view and the presenter?

เพื่อความสะดวกในการเรียกใช้งาน เมื่อ program มีความซับซ้อนมากขึ้น และง่ายต่อการทำ test อีกด้วย หากมีการเปลี่ยนแปลงตรงส่วนใดก็จะไม่ส่งผลกระทบต่ออีกฝั่งหนึ่งซึ่งถือว่าการลด coupling

Change directory to the "mvp" folder

```
# From assignment root
$ cd mvp
```

You will see these structure

```
mvp
├── main.py
├── mvp
│   ├── contracts
│   │   ├── __init__.py
│   │   └── main_contract.py
│   ├── __init__.py
│   ├── models
│   │   ├── entities
│   │   │   ├── __init__.py
│   │   │   └── note.py
│   │   ├── __init__.py
│   │   ├── repositories
│   │   │   ├── __init__.py
│   │   │   └── note_repository.py
│   ├── presenters
│   │   ├── base_presenter.py
│   │   ├── __init__.py
│   │   └── main_presenter.py
│   └── views
│       ├── base_view.py
│       ├── __init__.py
│       └── main_view.py
```

We will start by implementing the contract. Open the file main_contract.py

Our view needs to be updated by the presenter, to achieve this, the view needs an update method exposed through its interface.

Add these methods to the MainContract.View class

```
def update_view(self, items: List[Note]):
    pass
```

In the same way, the presenter is also accessed by the view therefore we also need to provide required methods in the MainContract.Presenter class

Add these methods to the MainContract.Presenter class

```
def add_note(self, note: str):  
    pass  
  
def get_all_notes(self):  
    pass  
  
def clear_all(self):  
    pass
```

Notice that both MainContract.View and MainContract.Presenter were extended from their corresponding base class.

Next, we will move to the presenter. Open main_presenter.py, you will see the MainPresenter which is the implementation of MainContract.Presenter.

The MainPresenter should implement all of the required methods stated in its interface. You will need to write these methods to interact with the business logic.

Hint: The view also passed in the constructor with type MainContract.View. That's the way the presenter updates the view. Also, don't forget to update the view when the data changed.

Q9: What is the role of the presenter?

ทำหน้าที่คล้าย ๆ กับ controller คือจะทำงานในการตอบสนองต่อการกระทำของ view อีกทั้งยังเป็นตัวส่ง update ไปยัง view เพื่อทำให้เกิดการเปลี่ยนแปลงของหน้าต่างหลังจากจบการทำงานที่ในส่วนนั้นเสร็จ

Q10: What is the main difference between the method in the MainController of the previous section and the method which you just implemented in the MainPresenter?

หลังจากทำคำสั่งที่ได้รับมาจากหน้า view เสร็จแล้วก็จะทำการ update ข้อมูลไปที่ส่วนของ view

The next part is to implement the view so that it can interact with the presenter. If you look at the base_view.py, you will notice the set_presenter method which is called by the BasePresenter constructor. This way both view and presenter are now seeing each other.

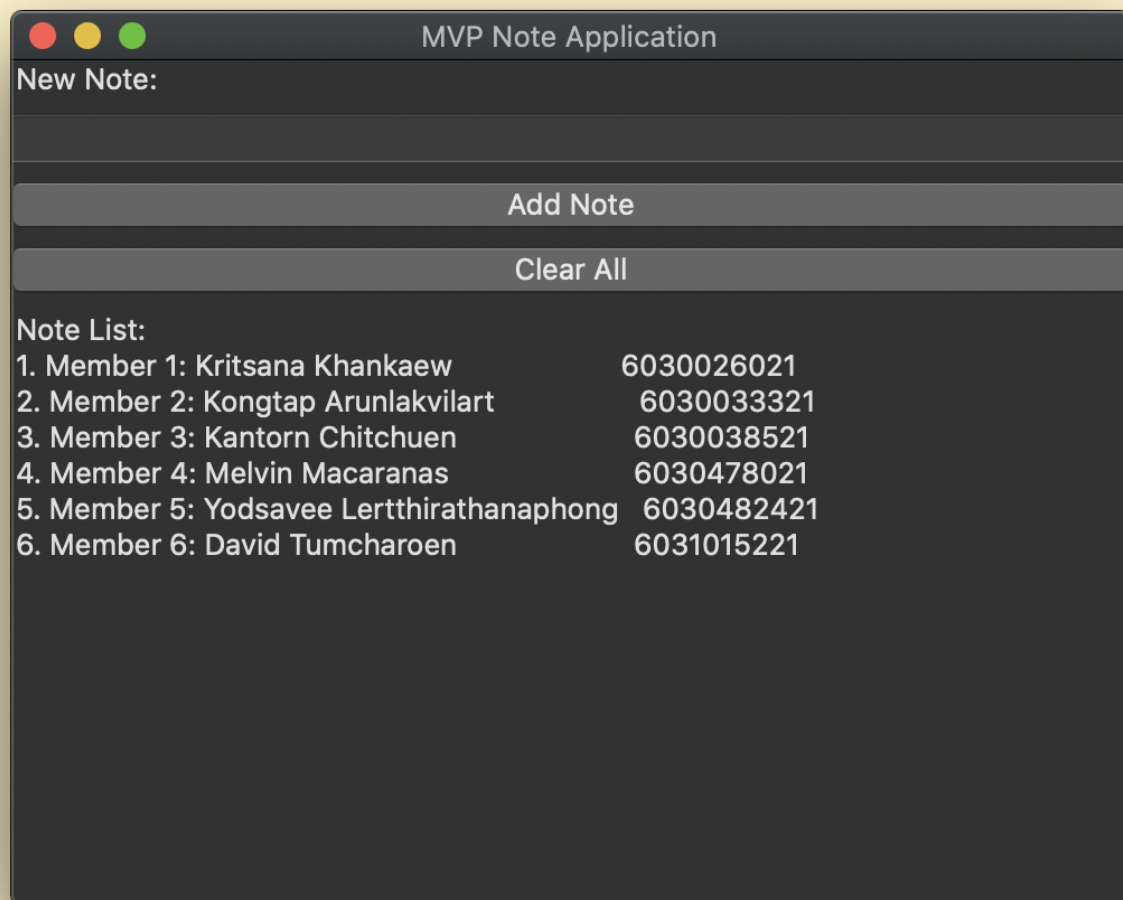
Open the file main_view.py and implement all missing method implementation.

Hint: Your code only needs to interact with the presenter. Also don't update the view inside the view and let the presenter do that.

Q11: How did you interact with the presenter? Do you think it makes the implementation of view harder or easier? Why?

view จะติดต่อกับ presenter โดยผ่านทาง contract ซึ่งทำหน้าที่เป็น interface จะทำให้การ implementation ของ view นั้นง่ายขึ้น เพราะไม่จำเป็นต้องกังวลการเขียน update หน้าต่างเนื่องจากส่วน presenter ทำไว้แล้ว และเมื่อเกิดการเปลี่ยนแปลงที่ view ก็ไม่จำเป็นต้องตามไปแก้ที่ presenter ด้วย

Q12: Put the screenshot of the MVP Note Application displaying your members' name in each note.



Q13: What are pros and cons of MVP pattern compared to MVC pattern?

ข้อดี:

- ลด coupling ระหว่าง view กับ presenter
- ทำการ test ได้ง่ายกว่า
- view มีแค่ส่วนที่แสดง ui

ข้อเสีย

- มีการทำงานที่ซับซ้อนกว่า
- presenter จะมีการทำงานที่มากขึ้น เมื่อเทียบกับ controller

Q14: With MVP pattern, do you think that your application is more testable? Why?

เมื่อใช้ pattern แบบ MVP จะทำให้การ test ของ application นั้นง่ายขึ้น เนื่องจากเมื่อเราทำการสร้าง test unit ขึ้นมาก็ไม่จำเป็นต้อง implement ในส่วนของ logic แต่ทำการเรียกใช้ คำสั่งจาก contract ก็เพียงพอแล้ว

MVVM: Model-View-ViewModel

Next, we will look into the MVVM pattern. In this pattern, we incorporate the reactive programming paradigm in which we make the view update itself automatically when the data change. This can be achieved by letting the views to act as observers while the view model serves the observable stream needed for the UI.

In this assignment we use RxPY, the ReactiveX library for Python, to do reactive programming. You can investigate the RxPY documentation at <https://rxpy.readthedocs.io/en/latest/> and ReactiveX at <http://reactivex.io/>.

Q15: What is reactive programming?

เป็นรูปแบบของโปรแกรมซึ่ง ถ้าหาก input มีการเปลี่ยนแปลง จะทำให้ output เปลี่ยนทันที

Q16: What is the observer pattern?

observer pattern จะประกอบด้วยผู้ส่งและผู้รับ ซึ่งหากผู้รับต้องการข้อมูล ให้ทำการ subscribe และจะมีการ update เรื่อย ๆ

In this pattern, the controller is replaced by the view model. The view model serves the observable stream. The view owns the view model as a field and then subscribes to it so that the view knows when to update itself.

One advantage is that the view model is totally decoupled from the view which also means that you can use it with other views without changing the content in the view model.

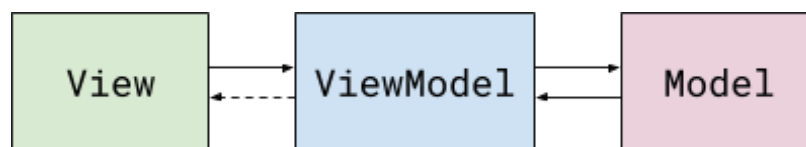


Figure: The MVVM pattern and the interaction between each layer

According to the figure, notice that the arrow from ViewModel to View is a dashed line. This is because the view doesn't interact with the view but the view just observes the change in view model.

Q17: Do you think that the view model should know which view object is owning it? Why?

ไม่จำเป็น เนื่องจาก view model เพียงแค่ส่งข้อมูลที่รับมาไปยัง model และส่งข้อมูลจะถูกกลับไปยัง view ที่ subscribe อยู่

Change directory to the “mvvm” folder

```
# From assignment root
$ cd mvvm
```

You will see these structure

```
mvvm
├── main.py
├── mvvm
│   ├── __init__.py
│   ├── models
│   │   ├── entities
│   │   │   ├── __init__.py
│   │   │   └── note.py
│   │   ├── __init__.py
│   │   └── repositories
│   │       ├── __init__.py
│   │       └── note_repository.py
│   └── view_models
```

```

|
|  ├── __init__.py
|  ├── main_view_model.py
|  └── views
|      ├── base_view.py
|      ├── __init__.py
|      └── main_view.py

```

Now, open main_view_model.py and implement all the missing things.

Hint: Look at https://rxpy.readthedocs.io/en/latest/reference_subject.html.

Q18: How do you create the observable stream (the behavior subject in this assignment)?

สร้างใน funtion __init__ ของ MainViewModel

Q19: How do you emit the new data (notes in this assignment) to the behavior subject?

ข้อมูลใหม่จะถูกเพิ่มผ่าน main_view_model จากนั้น behavior subject ที่ subscribeee อยู่จะได้รับข้อมูล

Q20: What is the role of the view model?

เป็นตัวที่ติดต่อกับผู้ใช้งาน จะทำการรับข้อมูลและแสดงผลให้ผู้ใช้งาน

Q21: What are the main differences between the presenter and the view model?

หากมีการเปลี่ยนแปลงข้อมูล view model จะเปลี่ยนทันที แต่ presenter จะต้องทำการเรียกข้อมูลก่อน

Q22: In terms of testability, what do you think is easier to test between Presenter and ViewModel? Why?

ViewModel ง่ายกว่า เนื่องจาก Presenter จะต้องสร้าง view ขึ้นมาเพื่อใช้ในการ test แต่ ViewModel สามารถ test ได้เลย

Next, we will move into the view implementation. Open main_view.py and implement all the missing things.

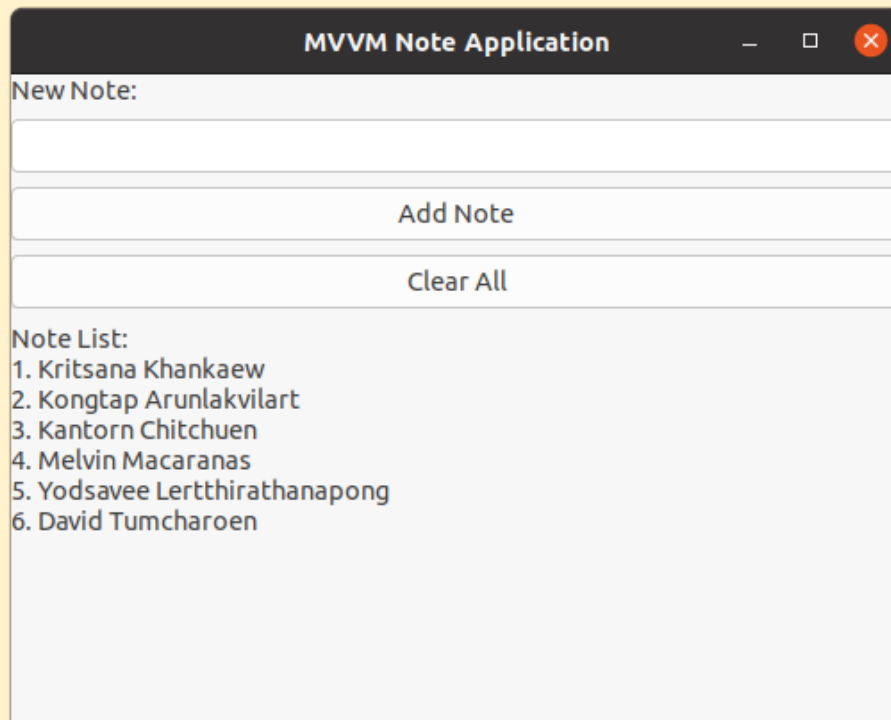
Hint 1: The view will update itself by subscribing to the observable stream in the view model.

Hint 2: You will be mostly interacting with the view model.

Q23: How did you interact with the view model?

เราจะทำการติดต่อ view model โดยส่งข้อมูลผ่านการเรียก function ของ view model และรับข้อมูลโดยการ subscribe

Q24: Put the screenshot of the MVVM Note Application displaying your members' name in each note.



Q25: What are pros and cons of MVVM pattern compared to MVC pattern?

ข้อดี

1. MVVM จะแสดงผลทันทีที่ข้อมูลใน model เปลี่ยนแปลง
2. MVVM สามารถ test ได้ง่ายกว่า

ข้อเสีย

1. MVVM ต้องมีการแปลงกันระหว่าง ViewModel และ View ในขณะที่ MVC สามารถใช้จาก Model ได้โดยตรงเลย

Q26: According to MVC, MVP and MVVM pattern, what pattern would you prefer for your application? Why?

MVVM เนื่องจาก shorten url เป็น reactive programming