CSIE5142/CSIE4302 Software Engineering

Homework # 3

Due on 12/14/2022

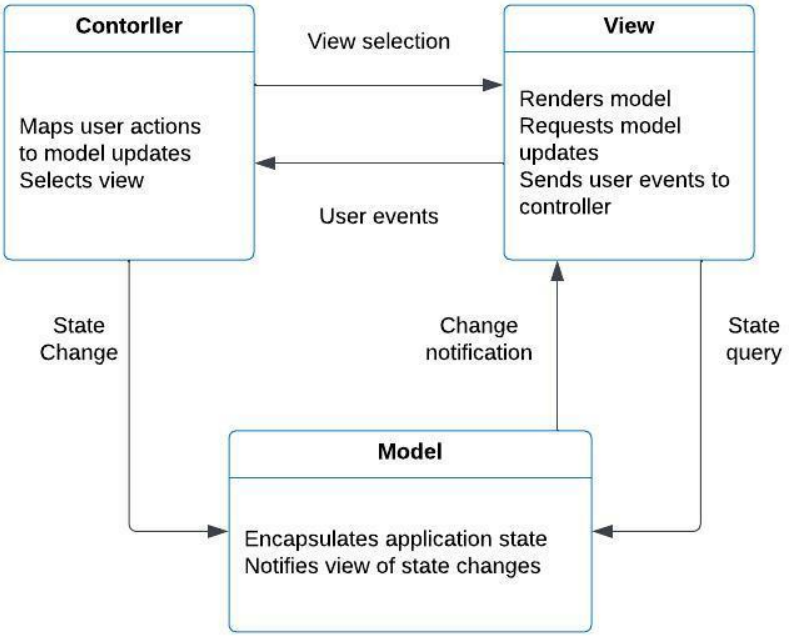
學號: 111598031 姓名: 楊佑鴻

1. (10%) What kinds of Architectural design decisions needed to be made in the software architecture design process?

   在軟體架構設計過程中，要做出架構設計的決策時，應先考慮以下問題:
   1. 是否有一種通用的應用程式架構(application architecture)，可以做為正在設計的系統的模板?
   2. 該系統會如何被分佈?(client-server, 3-tier, or N-tier)
   3. 什麼樣的架構模式或者方式較為適合?
   4. 會使用到什麼方法去建構系統? (component-based, distributed, client-server, layered, repository, SOA)
   5. 系統會如何分解為模組(modules)? (OOAD, DDD, microservices)
   6. 應使用什麼樣的控制策略(control strategy)及溝通機制(communication mechanism)?
   7. 如何評估該架構設計? (performance, security, availability, scalability, maintainability ..etc.)
   8. 應該如何將架構文件化?(UML, ADL)

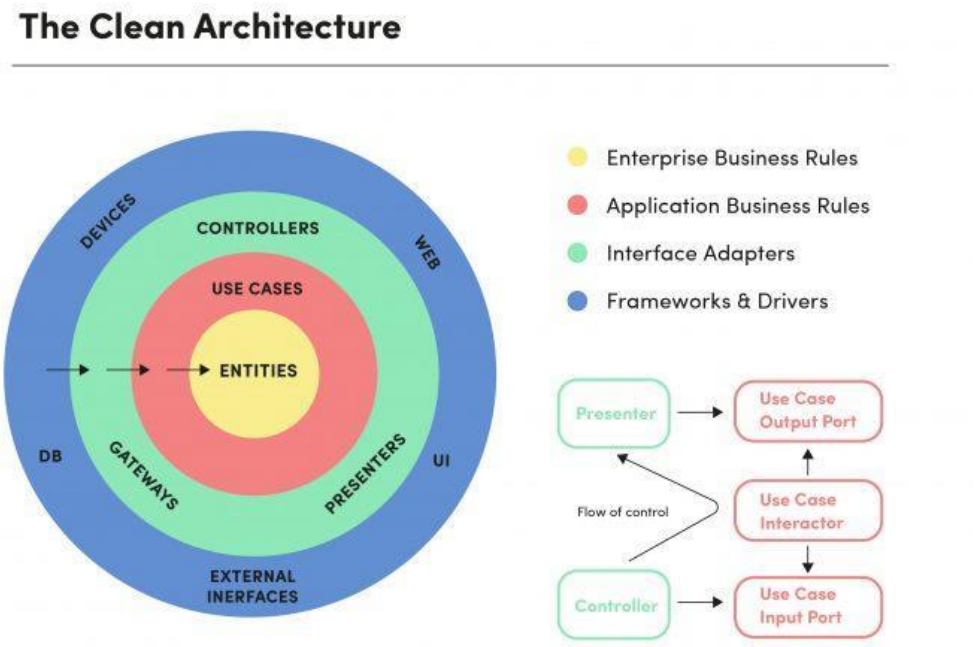
2. (35%) Software architecture
   (a) (10%) Draw and describe the **Model-View-Controller (MVC)** architecture?
   (b) (15%) Draw and describe the **clean architecture**.
   (c) (10%) What is **microservices** architecture?

   (a)



Model-View-Controller（MVC）將表示法與互動與系統資訊分離。將該系統的結構分為三個相互互動的邏輯元件。
模型元件（Model component）包含商業邏輯以及資料庫設計等等與系統資訊相關的行為或功能。
視圖元件（View component）定義和管理資訊如何呈現給使用者。顯示給使用者的資訊介面，使用者操作發送請求會送至 Controller。
控制器元件(Contronller component）介於 Model 及 View 之間，起到不同層面間的組織作用，用於控制應用程式的流程。負責處理事件並做出回應。「事件」包括用戶的行為和資料 Model 上的改變。

(b)



Clean architecture 是指組織整個專案，進而能夠達成最小化『建置和維護系統所需的人力』之目的。即使專案隨著發展擴大其規模，專案能夠保持易於理解和變更的特性。

Clean architecture 的目標著重於關注點分離（Separation of concerns），通過對軟體分層來實現這種分離。每層至少有一層業務規則(Business rules)，另一層則為介面。使用該架構會產生以下子系統: Testable, Independent of Frameworks, UI, databases and any external agency

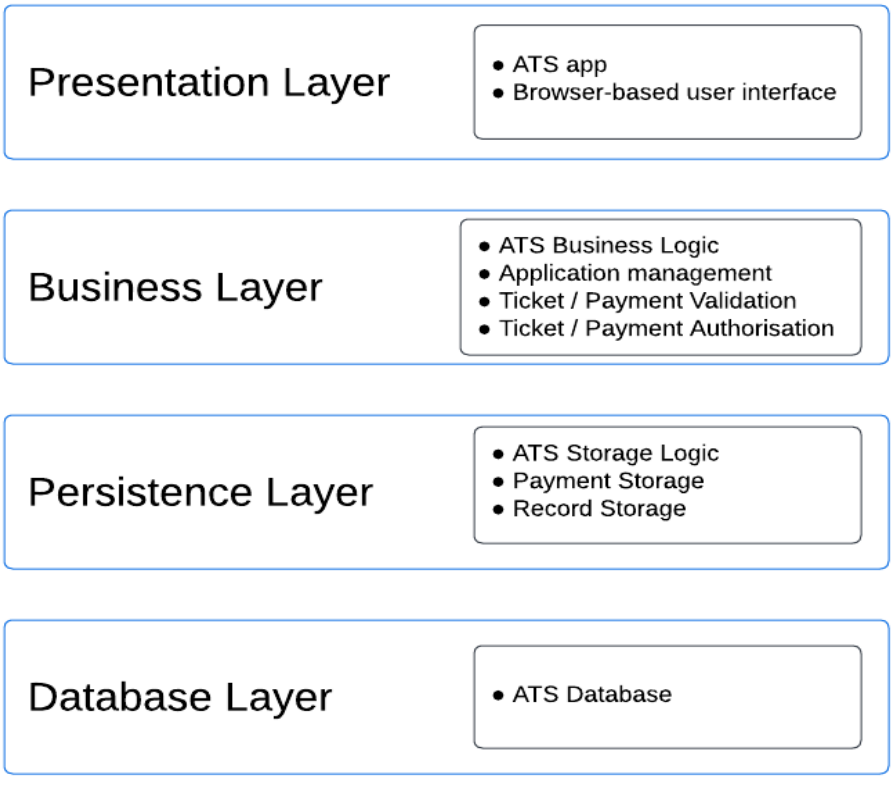
使用該架構需遵從以下規則:
跨層存取規則（The cross-layer access rule）: 跨越邊界的資訊應是簡單的資訊結構。
分層規則（The layer separation rule）: 圓圈代表軟體的不同領域，越往深處看，軟體的級別就越高。外圈為機制，內圈則為政策。
依賴規則（The dependency rule）: 原始碼的依賴只能指向內部(inwards)。

(c)

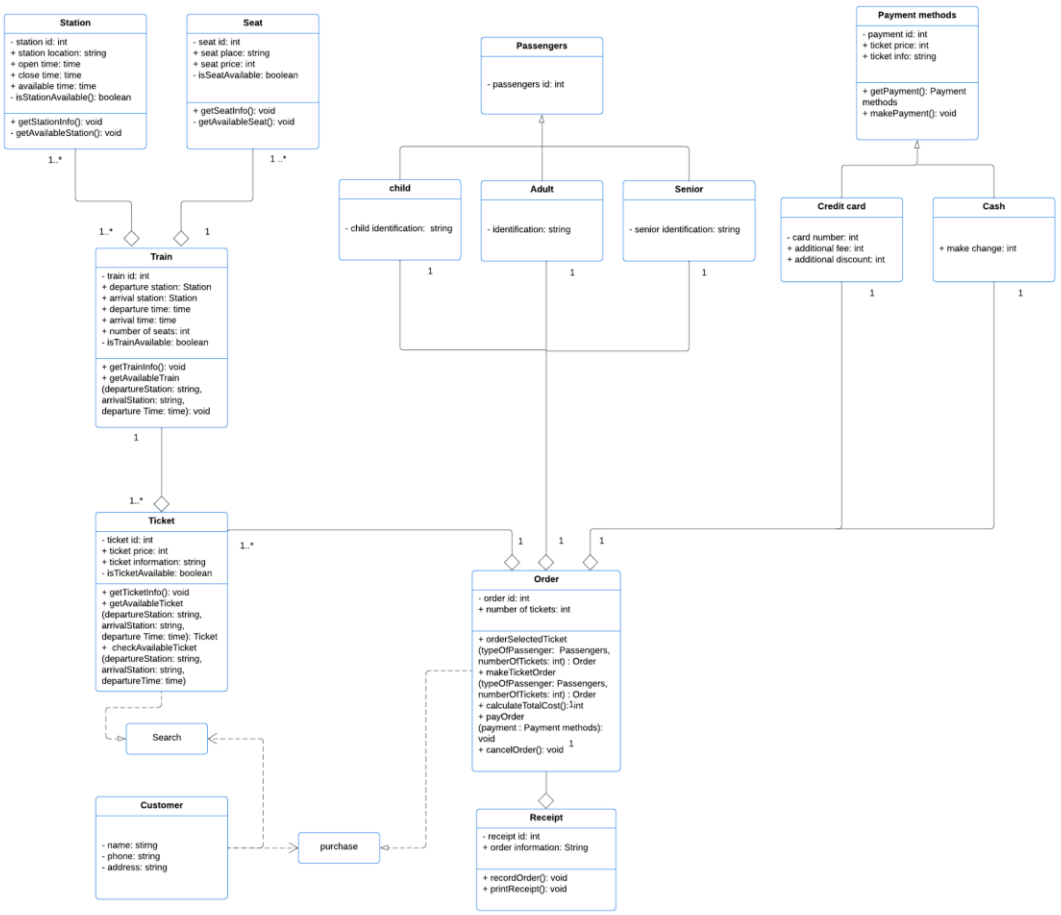
Microservices architecture 微服務架構（通常簡稱為「微服務」）是一種應用程式開發架構類型。微服務可讓大型應用程式分解為較小的獨立元件，每個元件都有各自負責的範圍。為了向單一使用者提供服務，微服務型應用程式可呼叫許多內部微服務來建構回應。

3. (40%) Consider the *automatic ticket seller (ATS)* system of a railroad company that allows customers to **purchase tickets**. When a customer would like purchase tickets, the customer needs to interact with the touch screen of the ATS system to select the **"departure station"**, **"arrival station"**, and the **departure time** of the train (or train number). If the desired train has no available seats, the ATS system will ask the customer to select another train or cancel this transaction. Otherwise, the ATS system will ask the customer to choose **the type of passengers**, such as child, adult, or senior, and **the number of tickets**. The ATS system then will show the total fees of the tickets and ask the customer to select **the types of payment methods**, such as credit card or cash. After payment of the customer is completed, the ATS system will print and dispense the required tickets. (*Note that the above requirement description is not complete. You need to explore and analyze the detailed steps of the interactions between the customer and system*. The score of this problem will depend on the completeness of your answer.)
   (a) (10%) Draw the **architecture diagram** for the above system. You may apply architecture styles (e.g., layered architecture) appropriated to the system.
   (b) (15%) Identify the objects and draw the **domain model** (or the **class diagram**) for the system. Show the essential attributes and relationships of the model. *State your own assumptions clearly if any.*
   (c) (15%) Draw the **sequence diagram** for the **"purchase tickets"** scenario. *State your own assumptions clearly if any.*

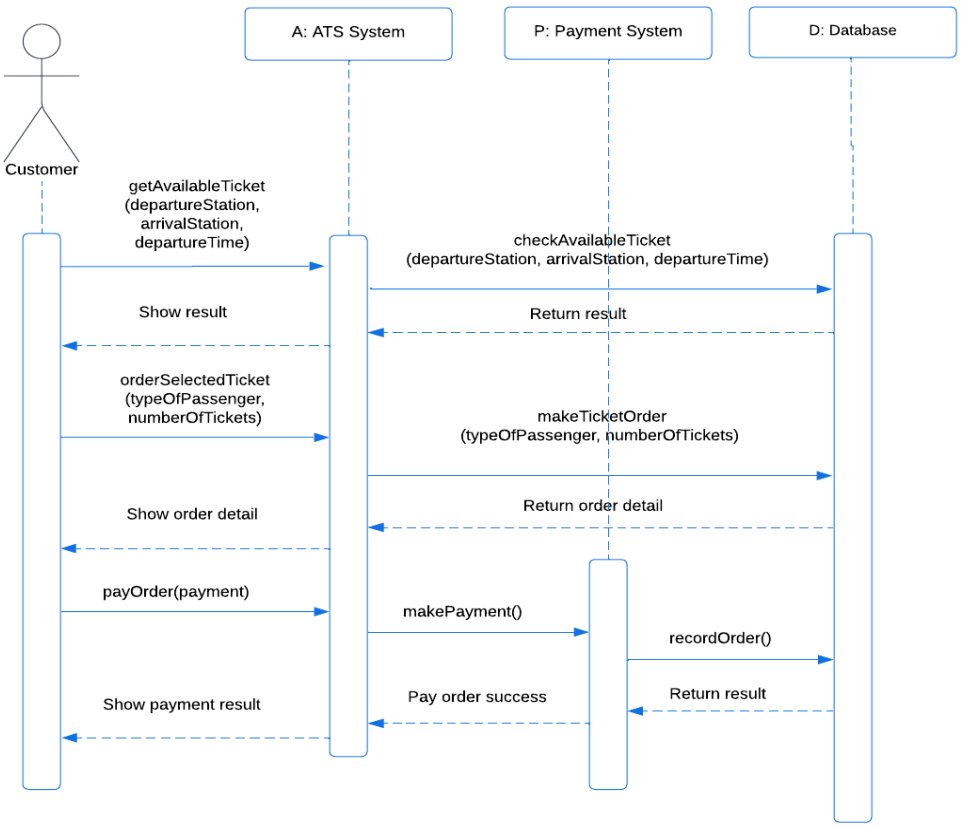
(a)



(b)


Station
- station id: int
+ station location: string
+ open time: time
+ close time: time
+ available time: time
- isStationAvailable(): boolean
+ getStationInfo(): void
- getAvailableStation(): void
Seat
- seat id: int
+ seat place: string
+ seat price: int
- isSeatAvailable: boolean
+ getSeatInfo(): void
- getAvailableSeat(): void
Passengers
- passengers id: int
Payment methods
- payment id: int
+ ticket price: int
+ ticket info: string
+ getPayment(): Payment methods
+ makePayment(): void
child
- child identification: string
Adult
- identification: string
Senior
- senior identification: string
Credit card
- card number: int
+ additional fee: int
+ additional discount: int
Cash
+ make change: int
Train
- train id: int
+ departure station: Station
+ arrival station: Station
+ departure time: time
+ arrival time: time
+ number of seats: int
- isTrainAvailable: boolean
+ getTrainInfo(): void
+ getAvailableTrain (departureStation: string, arrivalStation: string, departure Time: time): void
Ticket
- ticket id: int
+ ticket price: int
+ ticket information: string
- isTicketAvailable: boolean
+ getTicketInfo(): void
+ getAvailableTicket (departureStation: string, arrivalStation: string, departure Time: time): Ticket
+ checkAvailableTicket (departureStation: string, arrivalStation: string, departureTime: time)
Search
Customer
- name: stimg
- phone: string
- address: string
purchase
Order
- order id: int
+ number of tickets: int
+ orderSelectedTicket (typeOfPassenger: Passengers, numberOfTickets: int) : Order
+ makeTicketOrder (typeOfPassenger: Passengers, numberOfTickets: int) : Order
+ calculateTotalCost():1int
+ payOrder (payment : Payment methods): void
+ cancelOrder(): void
Receipt
- receipt id: int
+ order information: String
+ recordOrder(): void
+ printReceipt(): void

(c)


A: ATS System
P: Payment System
D: Database
Customer
getAvailableTicket (departureStation, arrivalStation, departureTime)
checkAvailableTicket (departureStation, arrivalStation, departureTime)
Show result
Return result
orderSelectedTicket (typeOfPassenger, numberOfTickets)
makeTicketOrder (typeOfPassenger, numberOfTickets)
Show order detail
Return order detail
payOrder(payment)
makePayment()
recordOrder()
Pay order success
Return result
Show payment result

4. (15%) The **SOLID** stands for five basic principles of object-oriented programming and design. Please describe the main concept of each design principle in SOLID below.

Initial	**Stands for**	**Name**	**Concept**
S	**SRP**	單一職責原則 (Single Responsibility Principle)	每個類別必須且只有一個單一的目的、職責及改變的理由。 （一個類別應只有一個職責。）
O	**OCP**	開放封閉原則 (Open Closed Principle)	一個類別應對擴增開放，但應對修改封閉。
L	**LSP**	里氏替換原則 (Liskov Substitution Principle)	延生的類別應總是擴展自它的父類別，且不會改變其行為。 (子類別必須可以替換它們的父類別，或者子類別必須滿足父類別定義的方法)
I	**ISP**	介面隔離原則 (Interface Segregation Principle)	不應強迫客戶依賴它不使用的方法。 (使用許多客戶客製化的介面比一個通用的介面好。)
D	**DIP**	依賴反轉原則 (Dependency Inversion Principle)	高層模組不該依賴於低層模組，兩者皆應依賴於抽象。 抽象不該依賴於具體，具體應依賴抽象。