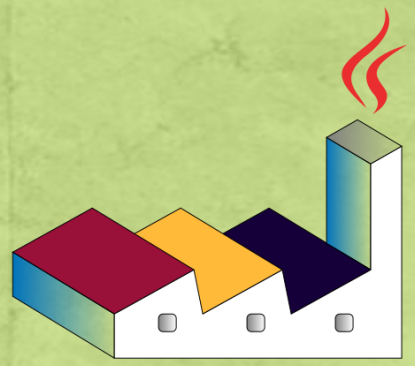
PlantUML

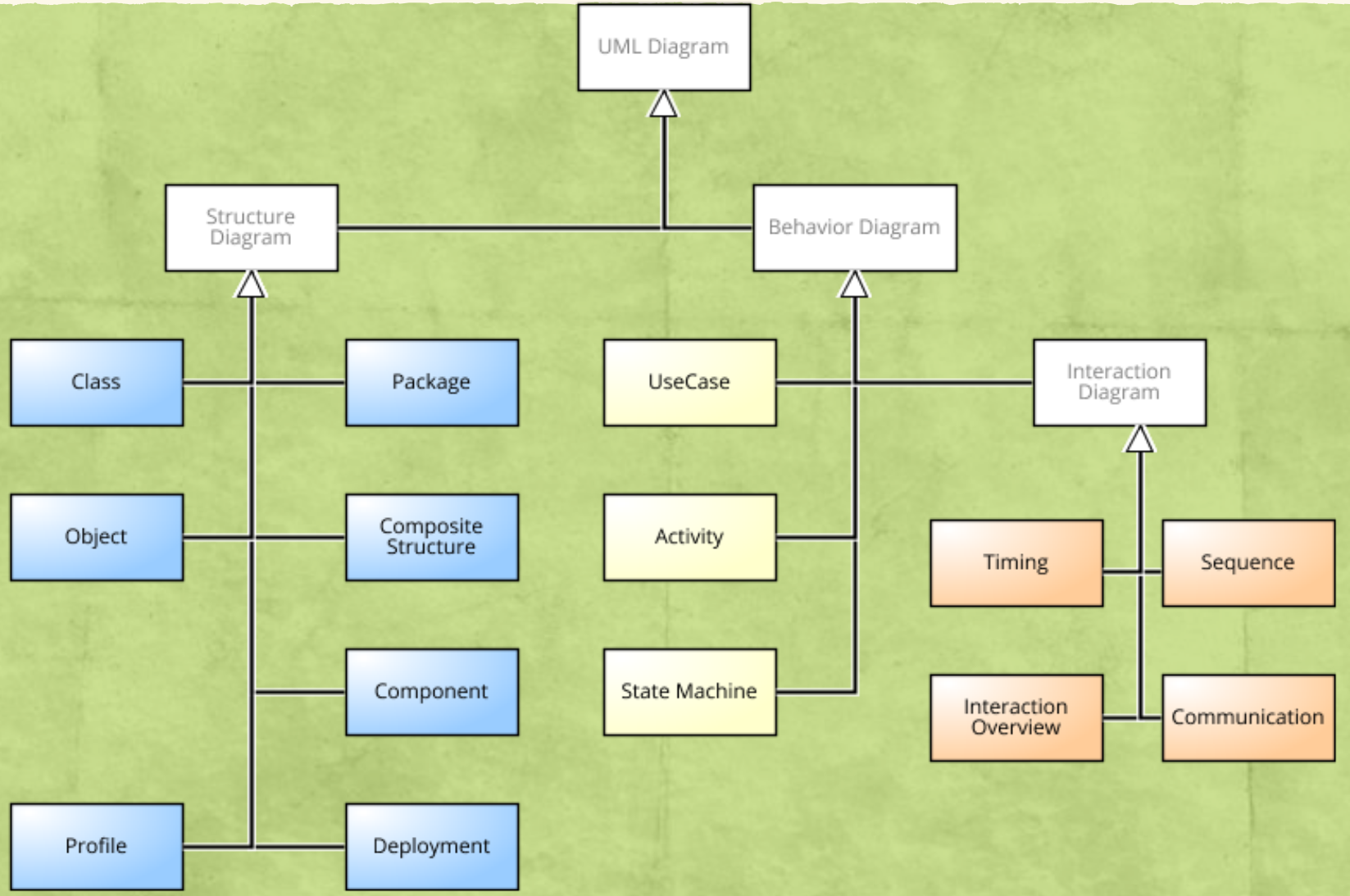


Vẽ sơ đồ bằng mã lệnh

UML là gì?

- UML (**U**nified **M**odeling **L**anguage) là ngôn ngữ mô hình hóa đa dụng, được phát triển bởi Grady Booch, Ivar Jacobson và James Rumbaugh tại Rational Software vào năm 1994–1995. Đến năm 1997, UML đã được thông qua như một tiêu chuẩn công nghiệp.
- UML cung cấp các sơ đồ để thể hiện thiết kế phần mềm dưới dạng ký hiệu đồ họa - mô hình. Và mô hình này cũng có thể được sử dụng cho mục đích lập tài liệu.

Các sơ đồ UML 2.5



Các công cụ vẽ sơ đồ

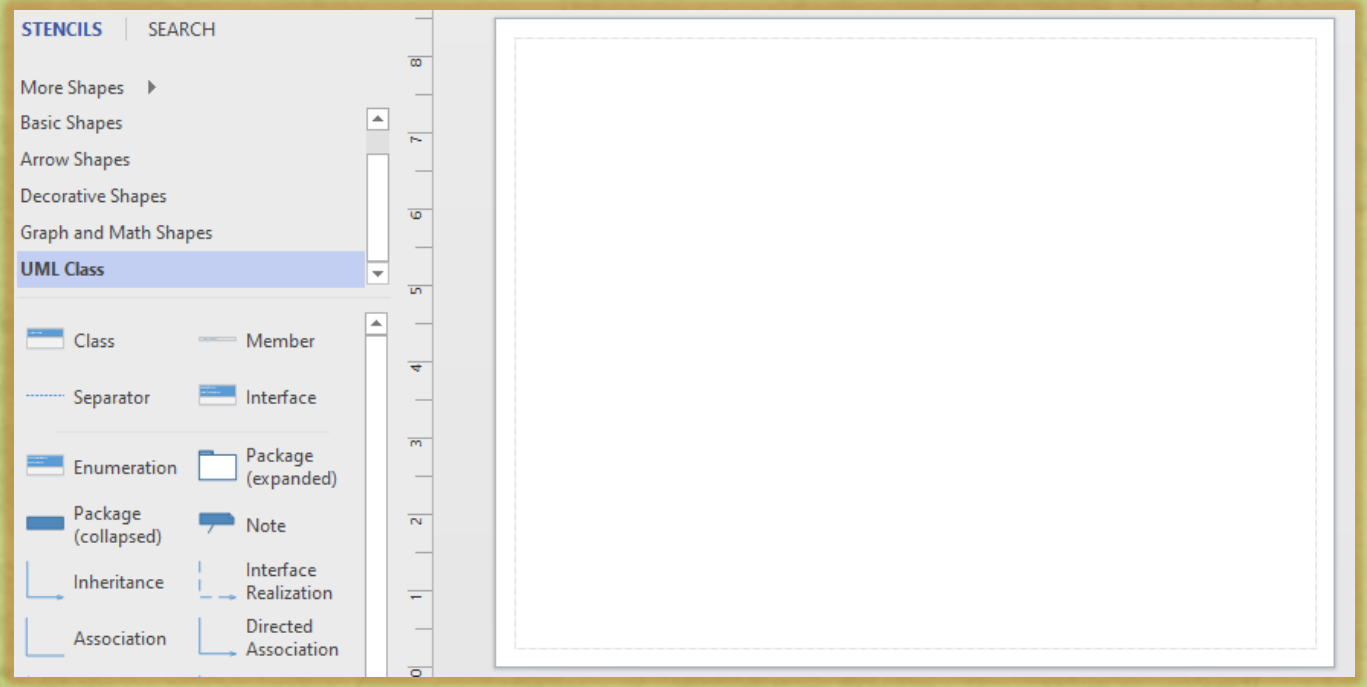
- Để vẽ sơ đồ, đơn giản chỉ cần bút và giấy. Tuy nhiên, nếu muốn lưu trữ tài liệu lâu dài và chỉnh sửa thường xuyên, chúng ta có thể phải sử dụng một công cụ chuyên nghiệp hơn.

Công cụ	Nền tảng
Microsoft Visio	Desktop
PowerDesigner	Desktop
Astah	Desktop
draw.io	Online
diagrams.net	Online
LicidChart	Online
.....	.....

Các công cụ vẽ sơ đồ

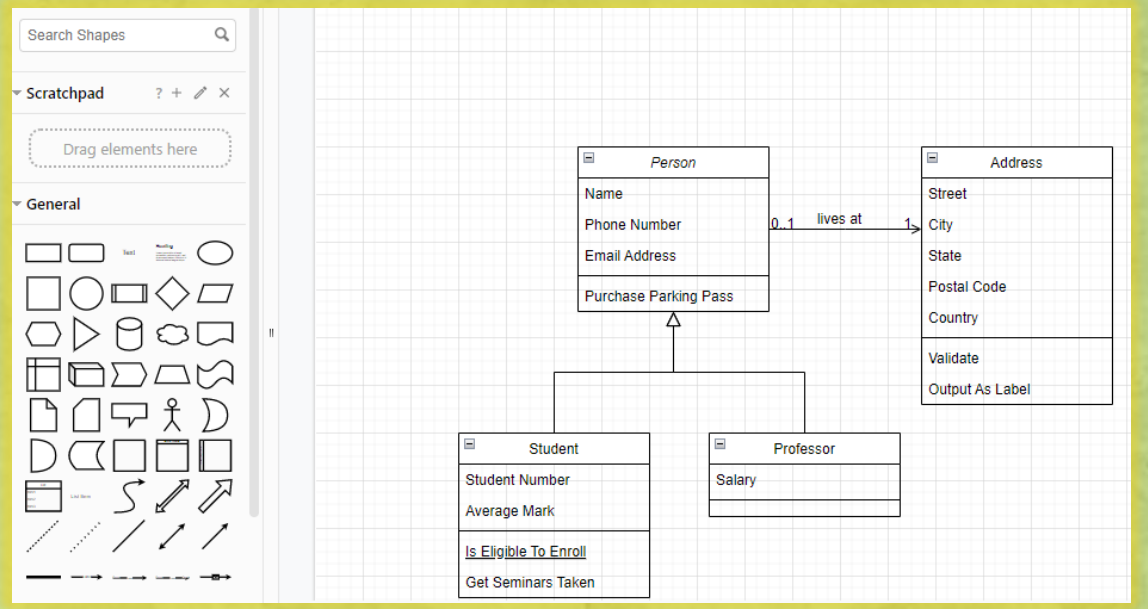
- Hầu hết các công cụ đều cung cấp giao diện đồ họa trực quan (WYSIWYG), cho phép người dùng tương tác bằng cách kéo-thả các đối tượng từ hộp công cụ vào vùng làm việc và liên kết chúng với nhau.
- Điều này rất thuận tiện cho các sơ đồ đơn giản, ít phải chỉnh sửa và người dùng quen sử dụng chuột.

Các công cụ vẽ sơ đồ



Microsoft Visio

diagrams.net



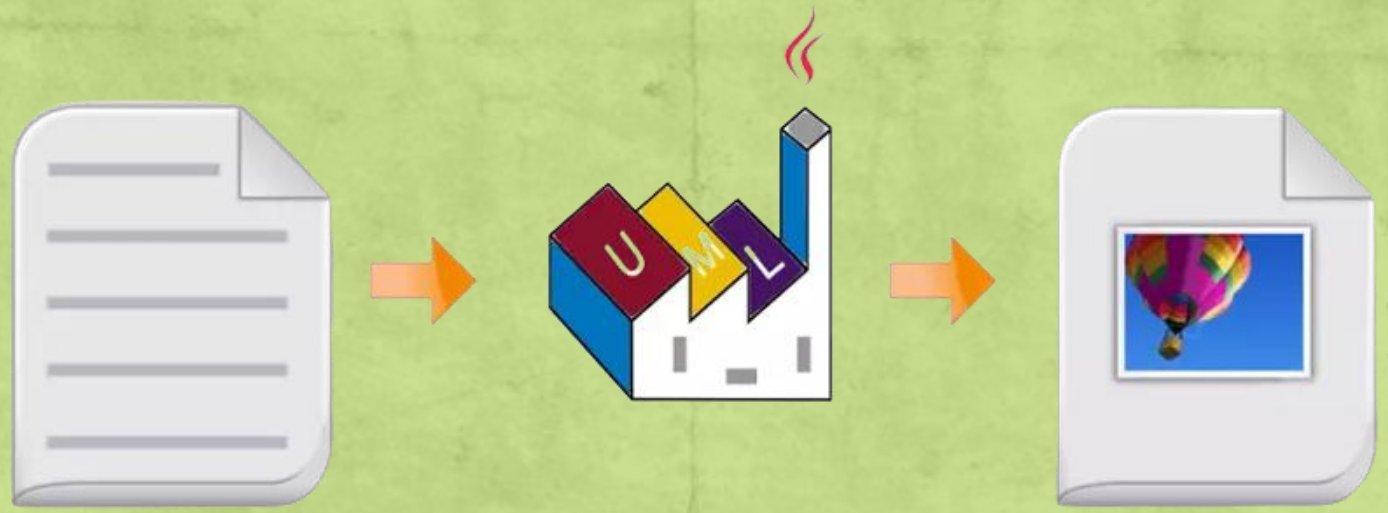
Các công cụ vẽ sơ đồ

- Tuy nhiên, các công cụ này sẽ không phải là lựa chọn tối ưu khi:
  - Các sơ đồ phức tạp
  - Nhu cầu chỉnh sửa thường xuyên
  - Cần đối chiếu sự khác biệt giữa 2 phiên bản của một sơ đồ (quản lý phiên bản)
  - Sơ đồ là kết quả hợp tác của các bên có liên quan

Các công cụ vẽ sơ đồ

- Và trong những trường hợp như thế, chúng ta nên sử dụng…

PlanUML



Plant UML là gì?

- PlantUML (plantuml.com) là một công cụ mã nguồn mở cho phép tạo sơ đồ từ ngôn ngữ văn bản thuần (plaint text).
- Bên cạnh các sơ đồ UML, PlantUML còn hỗ trợ nhiều định dạng liên quan đến phát triển phần mềm khác.

PlantUML is a component that allows to quickly write:

- Sequence diagram
- Usecase diagram
- Class diagram
- Object diagram
- Activity diagram (here is the legacy syntax)
- Component diagram
- Deployment diagram
- State diagram
- Timing diagram

The following non-UML diagrams are also supported:

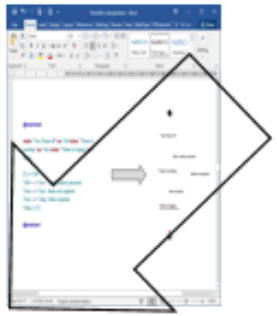
- JSON data
- YAML data
- Network diagram (nwdiag)
- Wireframe graphical interface or UI mockups (salt)
- Archimate diagram
- Specification and Description Language (SDL)
- Ditaa diagram
- Gantt diagram
- MindMap diagram
- Work Breakdown Structure diagram (WBS)
- Mathematic with AsciiMath or JLaTeXMath notation
- Entity Relationship diagram (IE/ER)

Kiến trúc của PlantUML

Online Editors (PlantText)

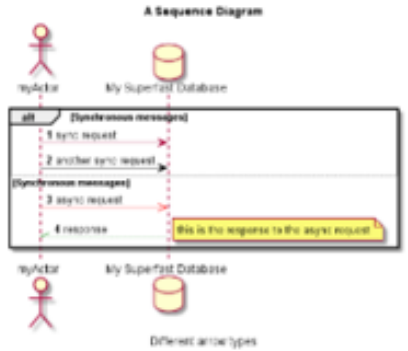
IDEs (e.g. Visual Code)

Word Processing (e.g. MS Word)



plantuml.jar

Graphviz dot.exe



Sử dụng Plant UML trực tuyến (thông qua Online Server)

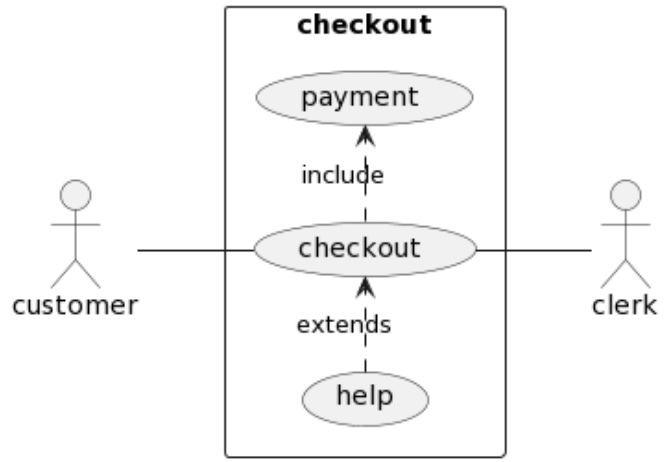
Sử dụng Plant UML trực tuyến (hoặc plainttext.com)

PlantText - The expert's design tool

Select a sample...

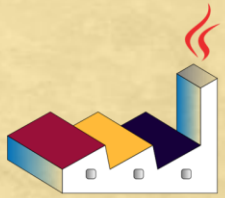
File Manager | Refresh (Alt+Enter) | File: Default Diagram

```
1  @startuml
2
3  left to right direction
4  skinparam packageStyle rectangle
5  actor customer
6  actor clerk
7  rectangle checkout {
8    customer --- (checkout)
9    (checkout) .> (payment) : include
10   (help) .> (checkout) : extends
11   (checkout) --- clerk
12 }
13
14 @enduml
```

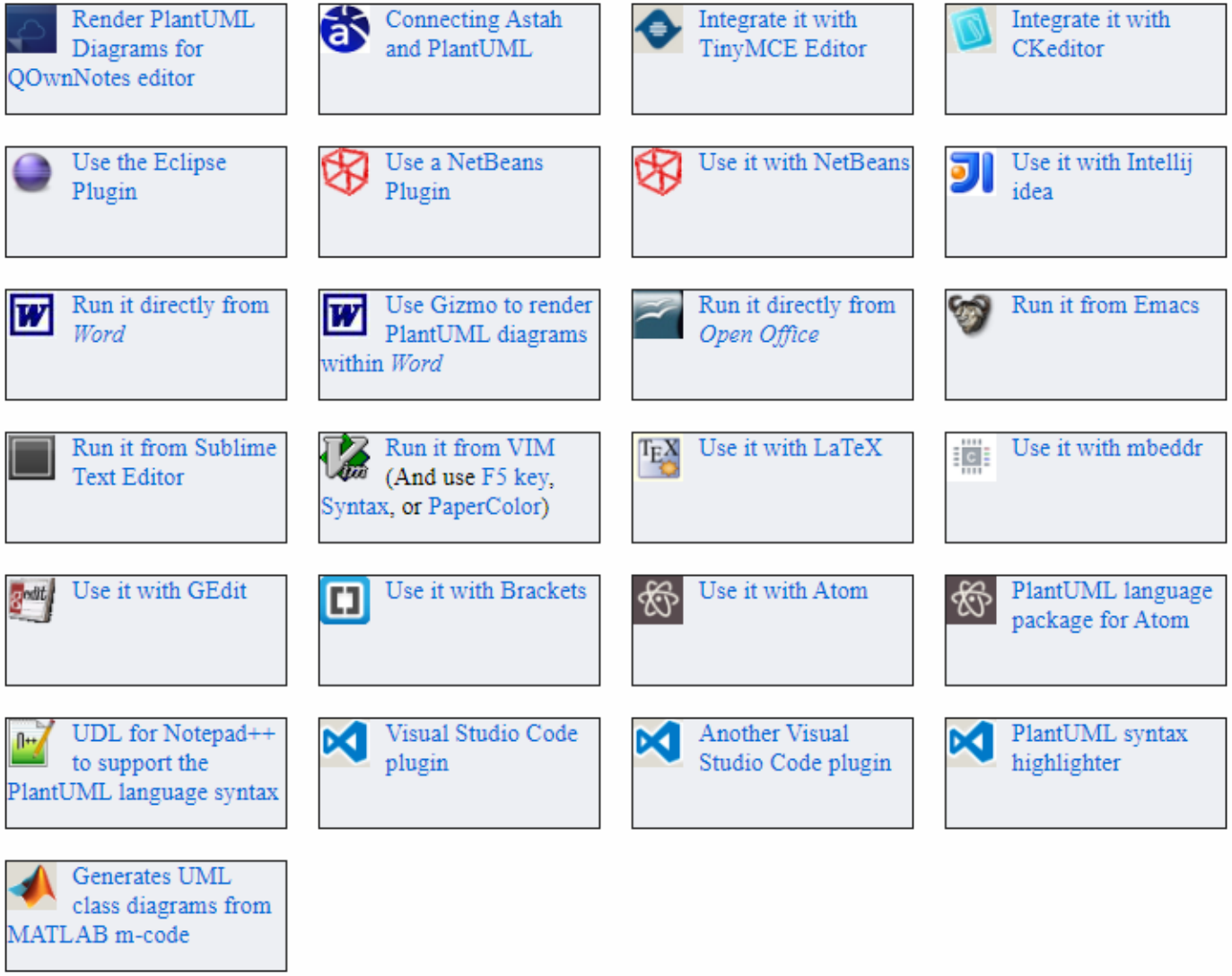


PNG | SVG | TXT | Edit

Sử dụng Plant UML (trong các IDE và phần mềm phổ biến)

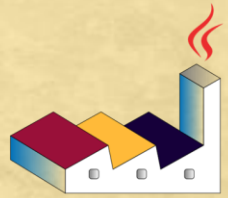


- Google Docs
- Microsoft Word
- IntelliJ IDEA
- Visual Studio Code
- NetBeans



- GitHub
- Eclipse
- LaTeX
- Atom
- MATLAB

Sử dụng Plant UML
(gọi các APIs từ ngôn ngữ lập trình)

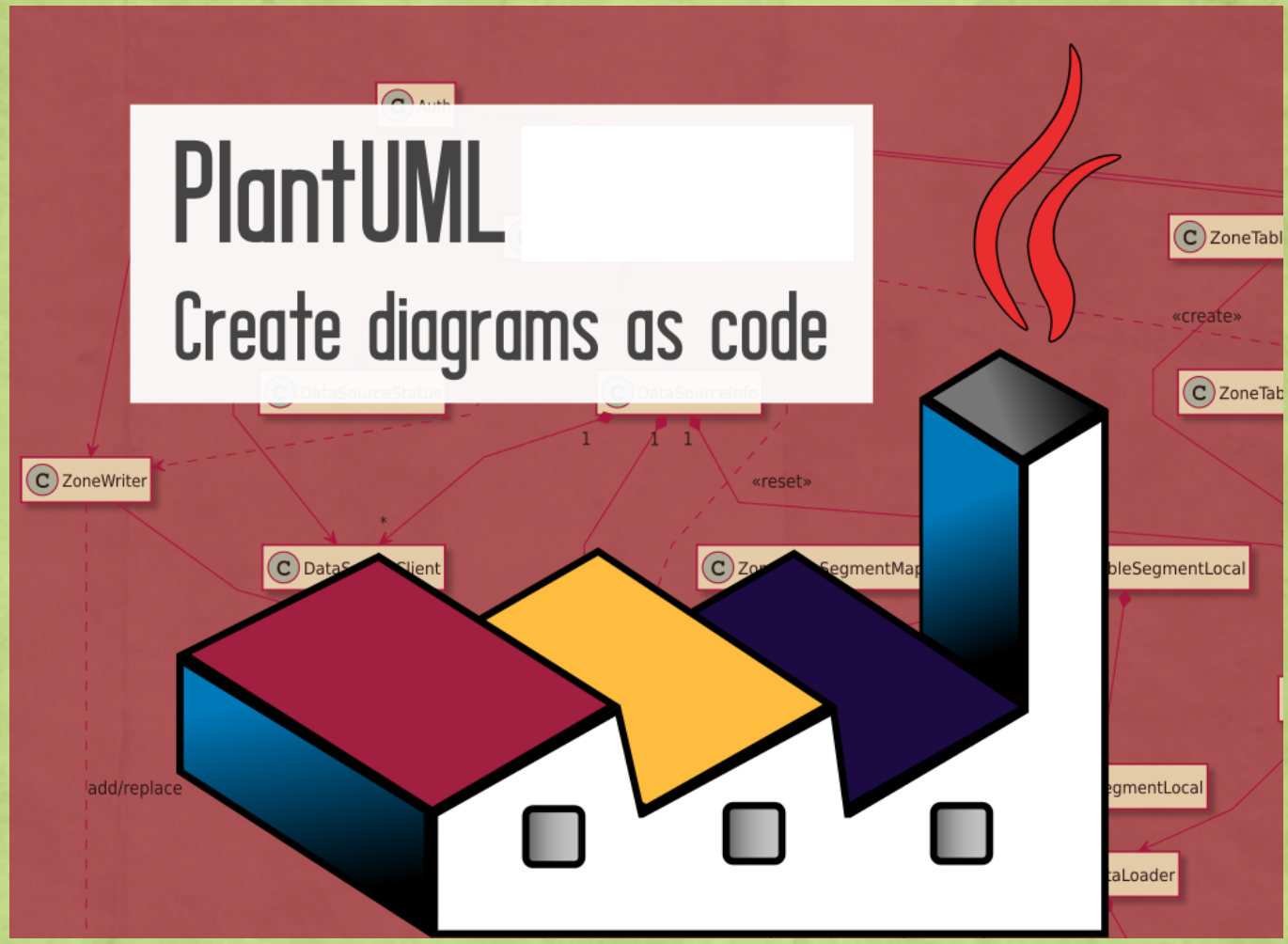


Use it with Markdown	Use it from HTML code with JQuery	JOII-based classes diagram generator	Call it from PHP
Call it from Java	Call it from Python	Another python remote client interface	Integration with IPython
Python tools for PlantUML	Call it from Groovy	Use builder pattern with Groovy PlantUML builder	Use command line
Write an ANT task	Use the Maven2 plugin	Use it with Gradle	Use it on Salesforce.com with Apex
A Leiningen plugin for generating UML diagrams using PlantUML	Emacs Lisp DSL for PlantUML	Generate PHP classes from your PlantUML diagram	

PlantUML

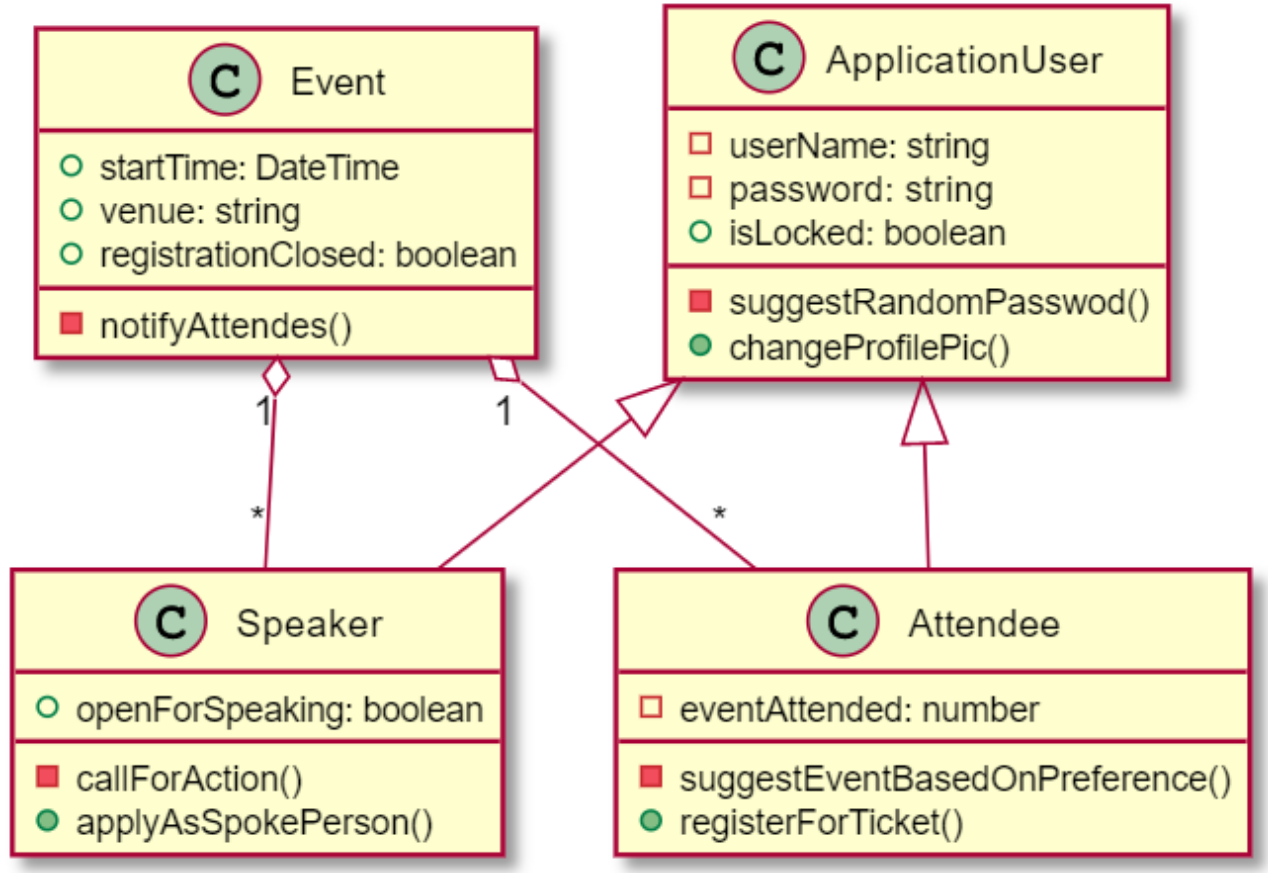
"vẽ"

được gì?

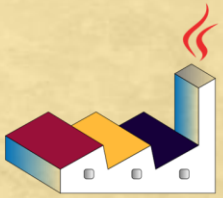


Class Diagram

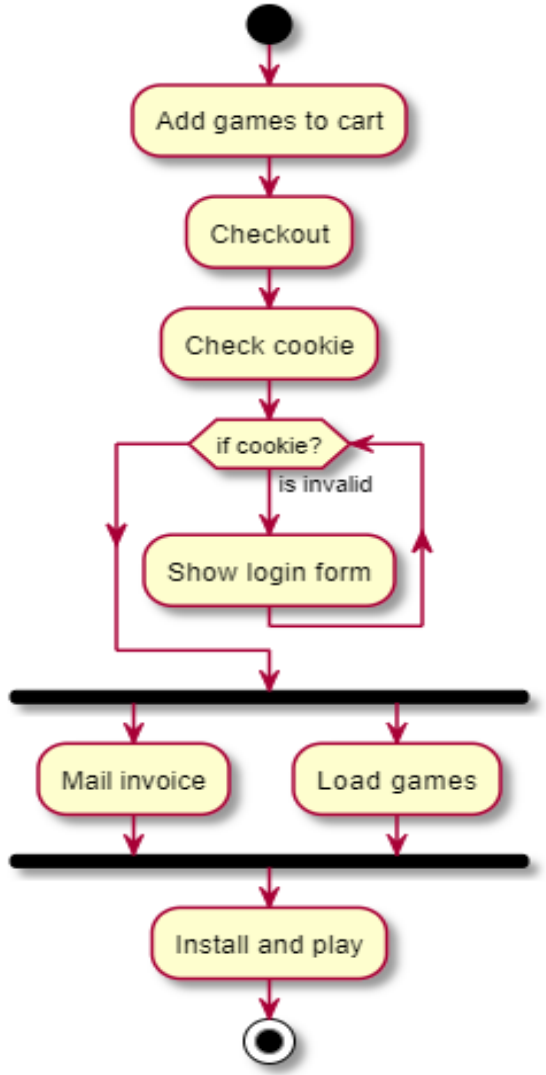

```
@startuml
scale 2
''Hide circle
class Event {
    +startTime: DateTime
    +venue: string
    +registrationClosed: boolean
    -notifyAttendes()
}

class ApplicationUser {
    -userName: string
    -password: string
    +isLocked: boolean
    -suggestRandomPasswod()
    +changeProfilePic()
}

class Speaker {
    +openForSpeaking: boolean
    -callForAction()
    +applyAsSpokePerson()
}

class Attendee {
    -eventAttended: number
    -suggestEventBasedOnPreference()
    +registerForTicket()
}

ApplicationUser <|-- Speaker
ApplicationUser <|-- Attendee
Event "1" o-- "*" Speaker
Event "1" o-- "*" Attendee
@enduml
```



Activity Diagram

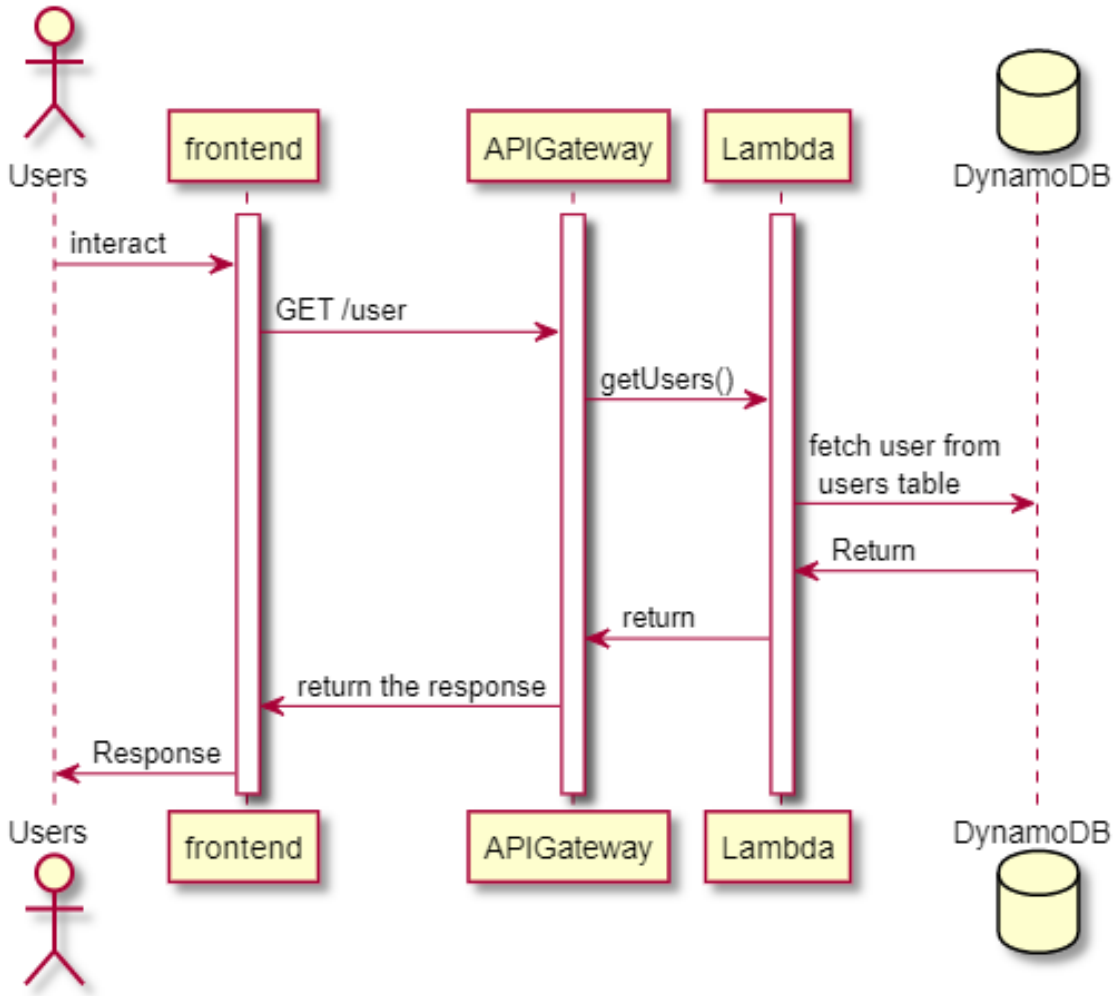



```
@startuml
start
:Add games to cart;
:Checkout;
:Check cookie;
while (if cookie?) is (is invalid)
    :Show login form;
endwhile
fork
    :Mail invoice;
fork again
    :Load games;
end fork
:Install and play;
stop
@enduml
```



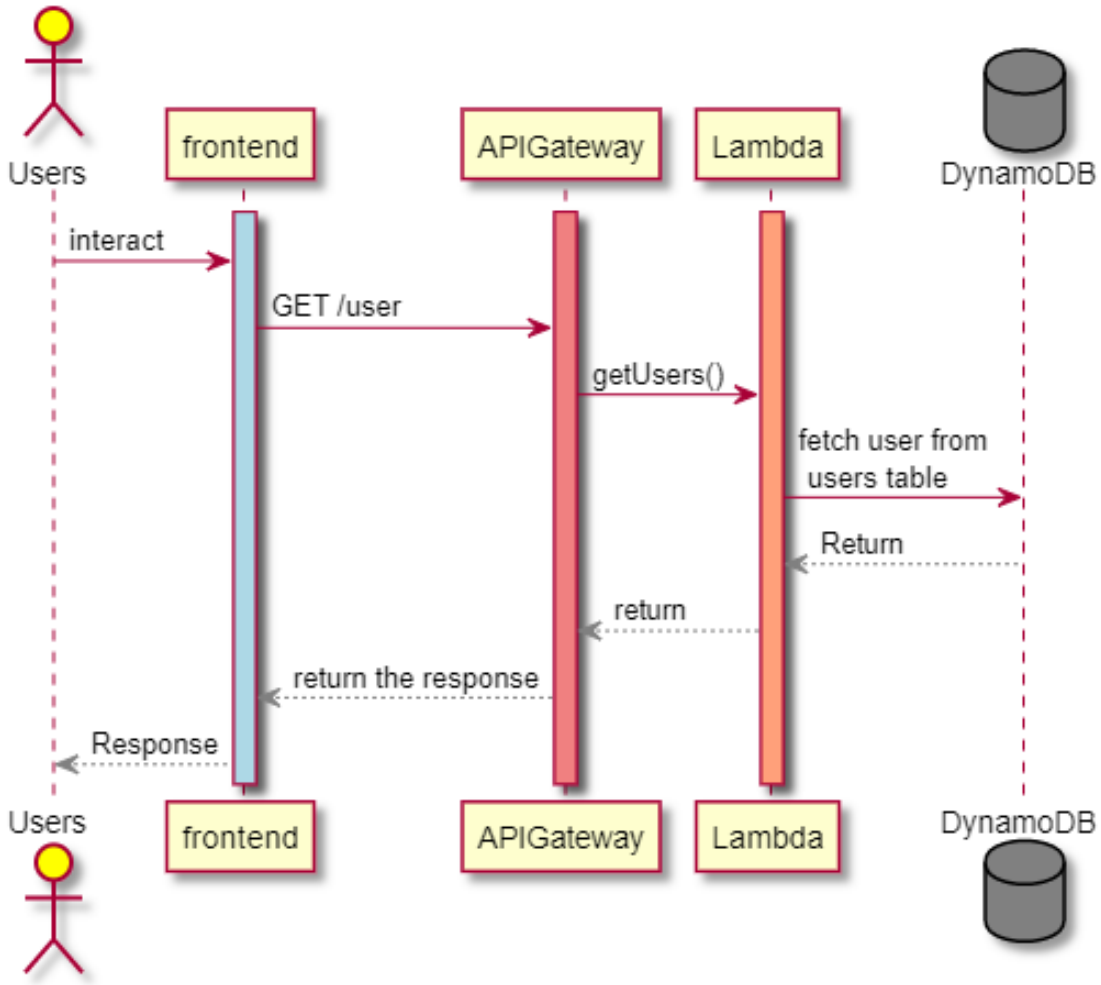
Sequence Diagram


```
@startuml
actor Users
activate frontend
activate  APIGateway
activate  Lambda
database DynamoDB

Users -> frontend : interact
frontend -> APIGateway : GET /user
APIGateway-> Lambda : getUsers()
Lambda -> DynamoDB : fetch user from\n users table
DynamoDB -> Lambda: Return

APIGateway <- Lambda  : return

APIGateway -> frontend : return the response
frontend -> Users : Response
@enduml
```



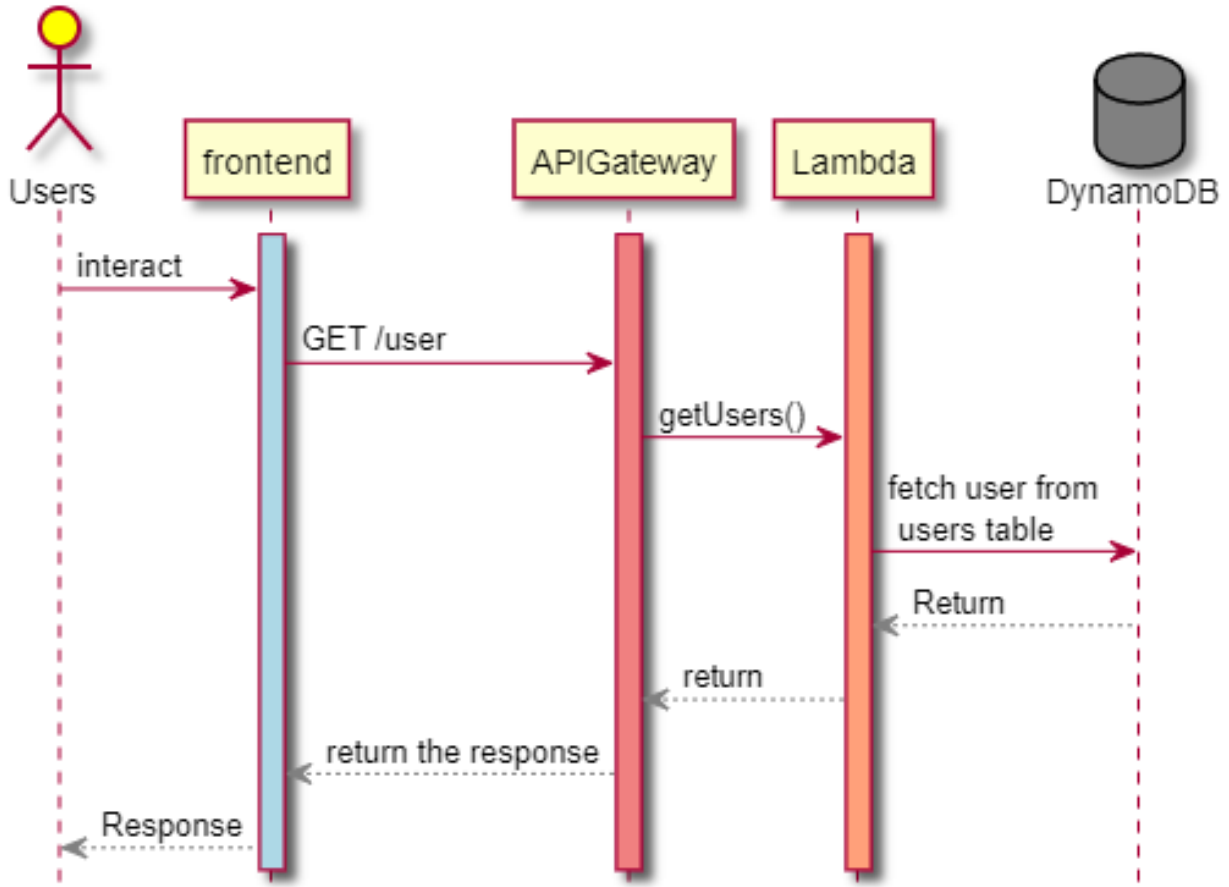
Sequence Diagram


```
@startuml
actor Users #Yellow
activate frontend #LightBlue
activate  APIGateway #LightCoral
activate  Lambda #LightSalmon
database DynamoDB #grey

Users -> frontend : interact
frontend -> APIGateway : GET /user
APIGateway-> Lambda : getUsers()
Lambda -> DynamoDB : fetch user from\n users table
DynamoDB -[#Gray]-> Lambda: Return

APIGateway <-[#Gray]- Lambda  : return

APIGateway -[#Gray]-> frontend : return the response
frontend -[#Gray]-> Users : Response
@enduml
```

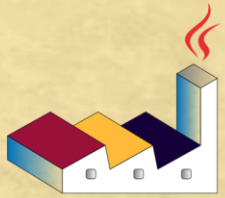


Sequence Diagram

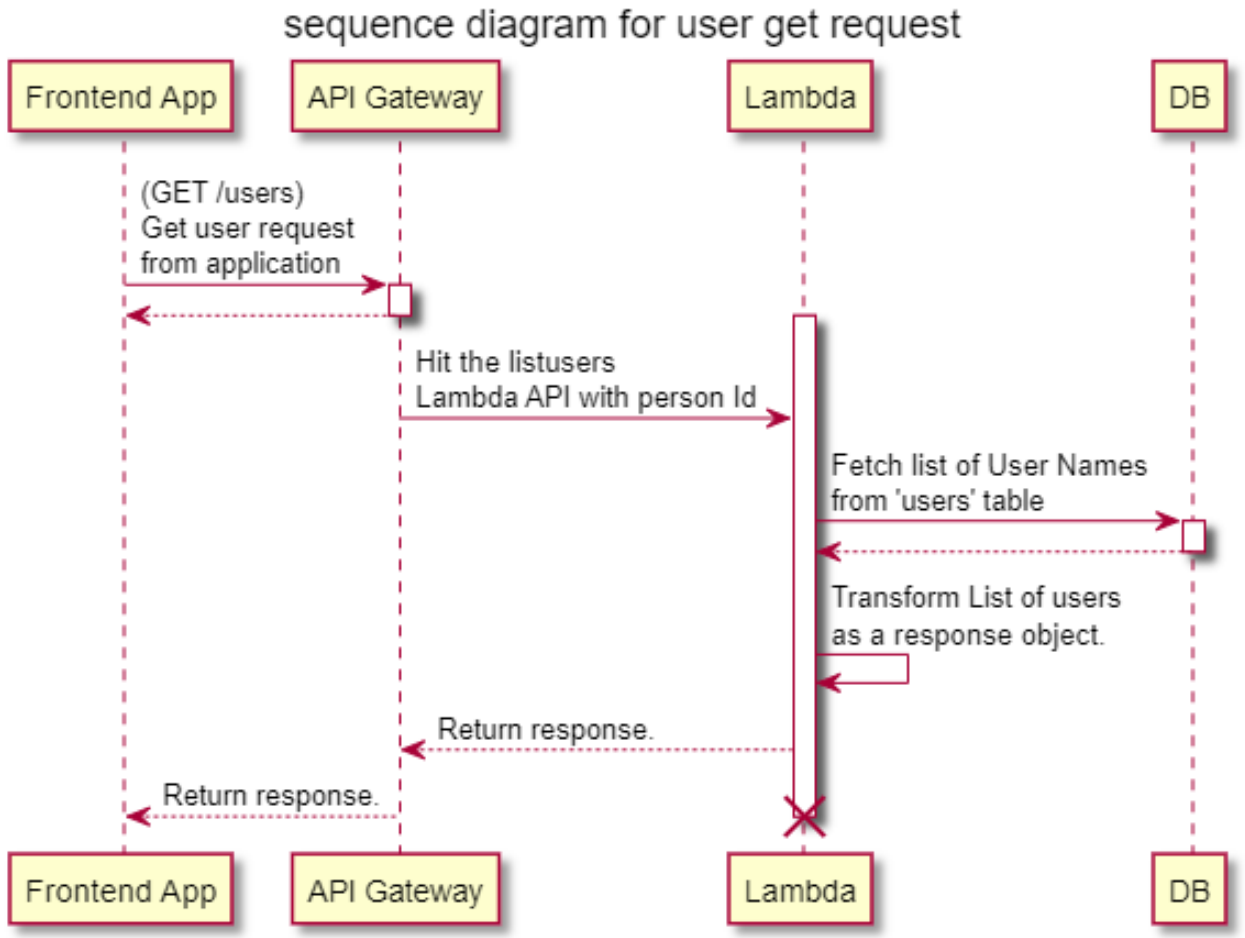

```
@startuml
hide footbox
actor Users #Yellow
activate frontend #LightBlue
activate  APIGateway #LightCoral
activate  Lambda #LightSalmon
database DynamoDB #grey

Users -> frontend : interact
frontend -> APIGateway : GET /user
APIGateway-> Lambda : getUsers()
Lambda -> DynamoDB : fetch user from\n users table
DynamoDB -[#Gray]-> Lambda: Return

APIGateway <-[#Gray]- Lambda  : return

APIGateway -[#Gray]-> frontend : return the response
frontend -[#Gray]-> Users : Response
@enduml
```



Sequence Diagram

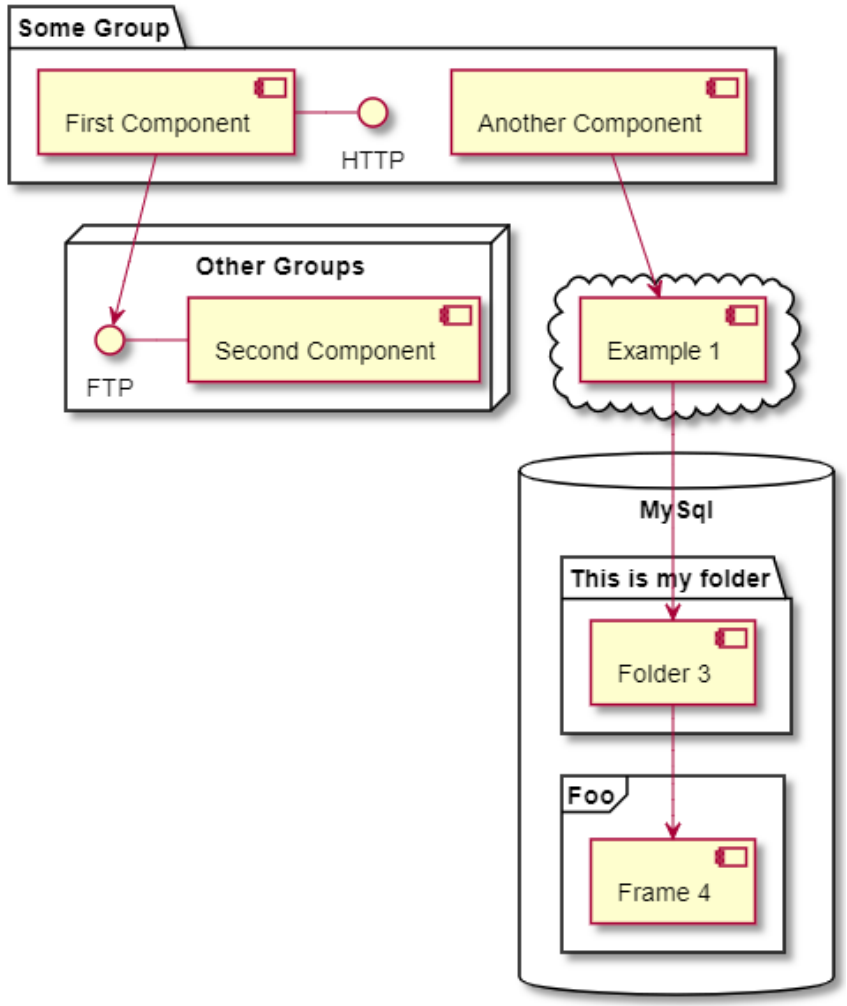



```
@startuml get-user-list

participant "Frontend App" as app
participant "API Gateway" as gateway
participant "Lambda" as lambda
participant "DB" as db

title sequence diagram for user get request

app -> gateway: (GET /users)\nGet user request \nfrom
activate gateway
return

activate lambda
gateway -> lambda: Hit the listusers \nLambda API with

lambda -> db: Fetch list of User Names \nfrom 'users'
activate db
return

lambda -> lambda: Transform List of users \nas a respo
gateway <-- lambda: Return response.
app <-- gateway: Return response.
destroy lambda

@enduml
```



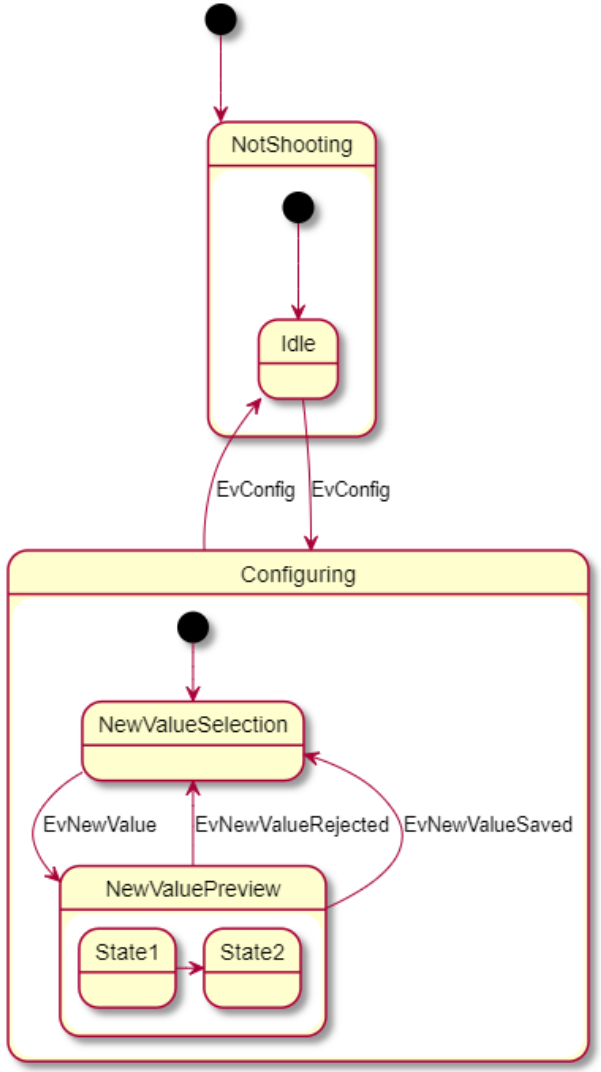
Component Diagram


```
@startuml
package "Some Group" {
    HTTP - [First Component]
    [Another Component]
}

node "Other Groups" {
    FTP - [Second Component]
    [First Component] --> FTP
}

cloud {
    [Example 1]
}

database "MySql" {
    folder "This is my folder" {
        [Folder 3]
    }
    frame "Foo" {
        [Frame 4]
    }
}

[Another Component] --> [Example 1]
[Example 1] --> [Folder 3]
[Folder 3] --> [Frame 4]
@enduml
```



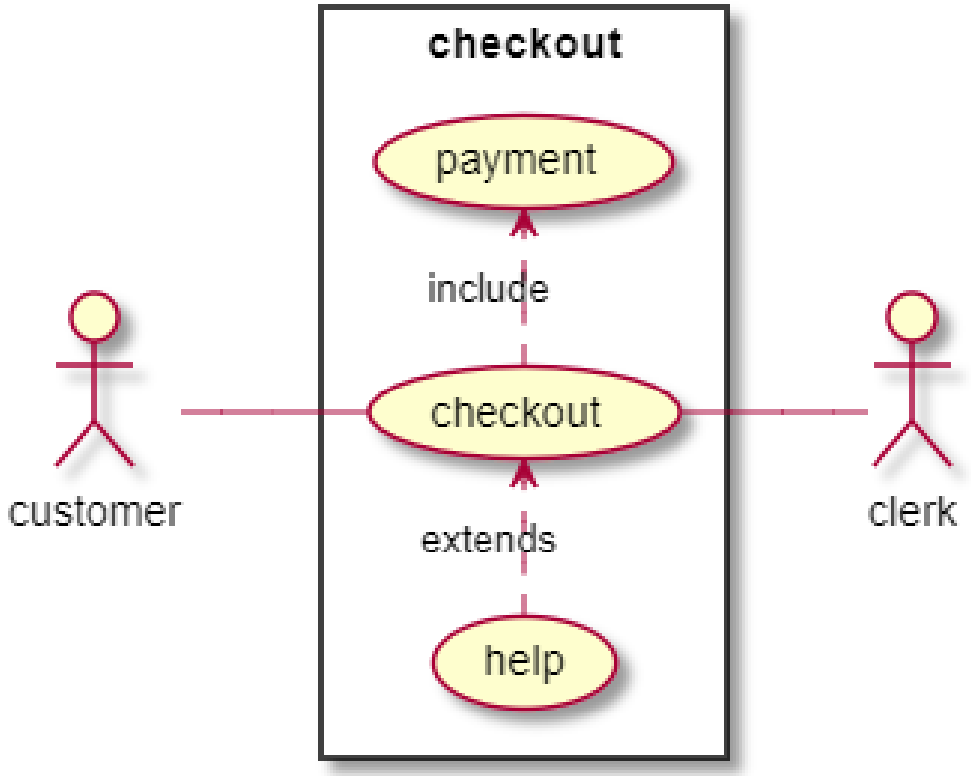
State Diagram


```
@startuml
[*] --> NotShooting

state NotShooting {
    [*] --> Idle
    Idle --> Configuring : EvConfig
    Configuring --> Idle : EvConfig
}

state Configuring {
    [*] --> NewValueSelection
    NewValueSelection --> NewValuePreview : EvNewValue
    NewValuePreview --> NewValueSelection : EvNewValueRejected
    NewValuePreview --> NewValueSelection : EvNewValueSaved

    state NewValuePreview {
        State1 -> State2
    }
}
@enduml
```



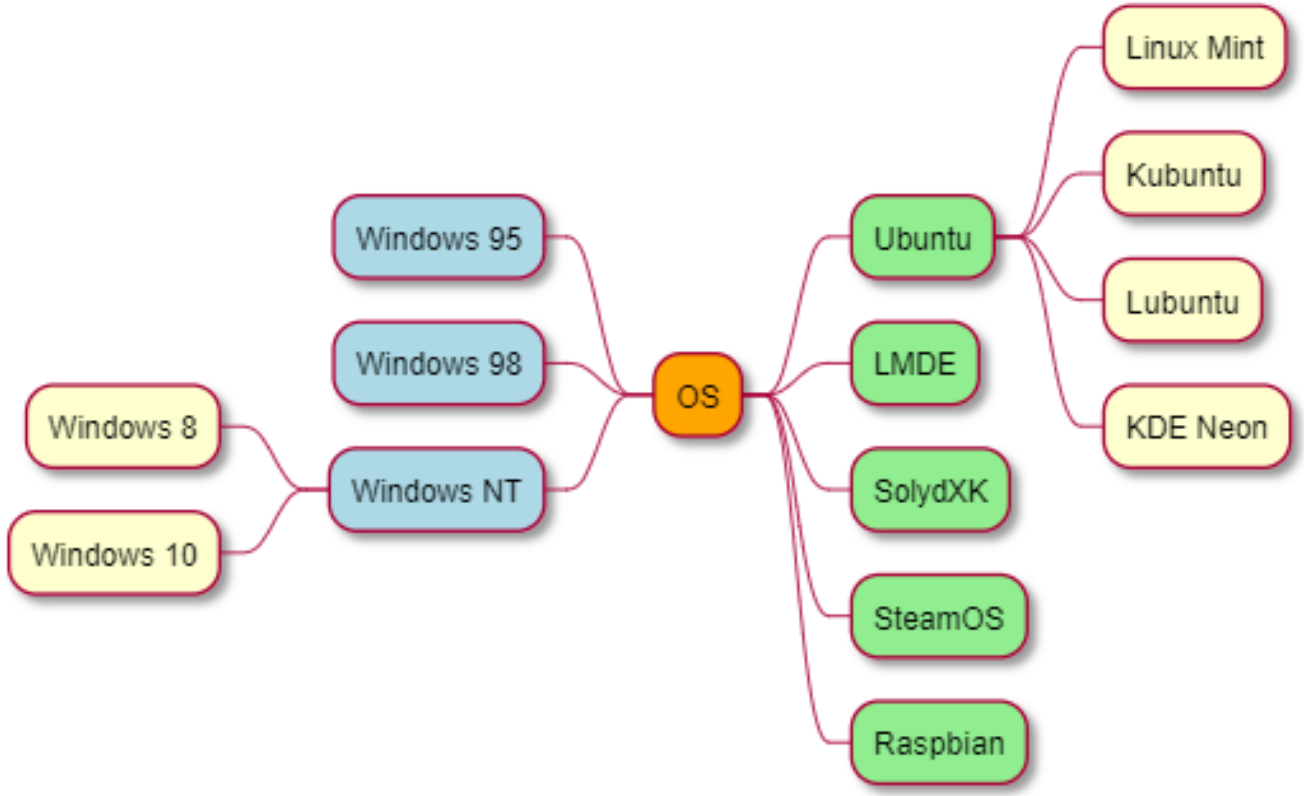
Use Case Diagram


```
@startuml
left to right direction
skinparam packageStyle rectangle

actor customer
actor clerk
rectangle checkout {
    customer -- (checkout)
    (checkout) .> (payment) : include
    (help) .> (checkout) : extends
    (checkout) -- clerk
}
@enduml
```



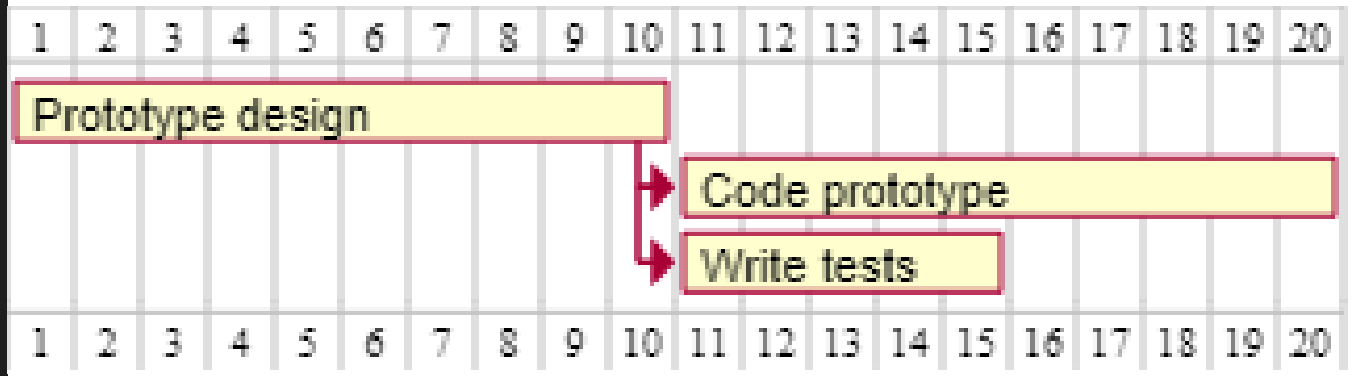
Mindmap


```
@startmindmap
+[#Orange] OS
++[#lightgreen] Ubuntu
+++ Linux Mint
+++ Kubuntu
+++ Lubuntu
+++ KDE Neon
++[#lightgreen] LMDE
++[#lightgreen] SolydXK
++[#lightgreen] SteamOS
++[#lightgreen] Raspbian
--[#lightblue] Windows 95
--[#lightblue] Windows 98
--[#lightblue] Windows NT
--- Windows 8
--- Windows 10
@endmindmap
```



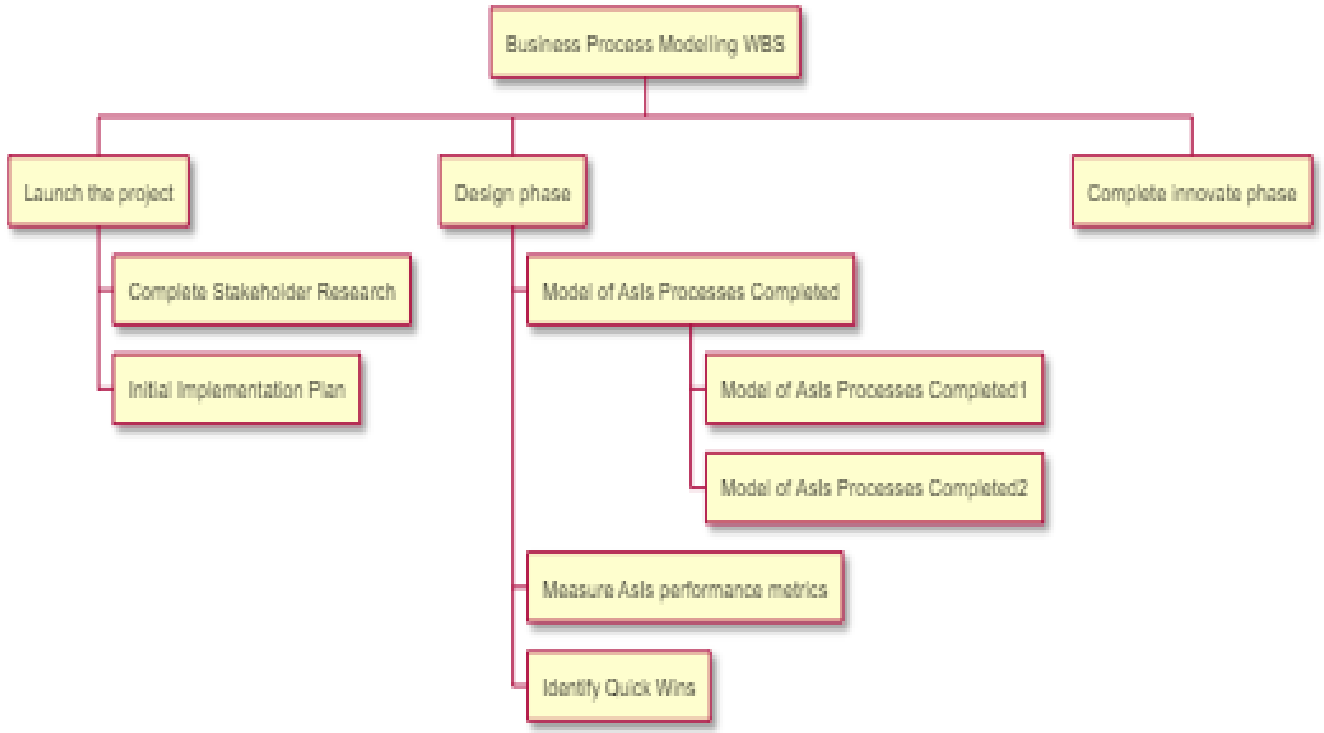
Gantt chart


```
@startgantt
[Prototype design] lasts 10 days
[Code prototype] lasts 10 days
[Write tests] lasts 5 days
[Code prototype] starts at [Prototype design]'s end
[Write tests] starts at [Code prototype]'s start
@endgantt
```



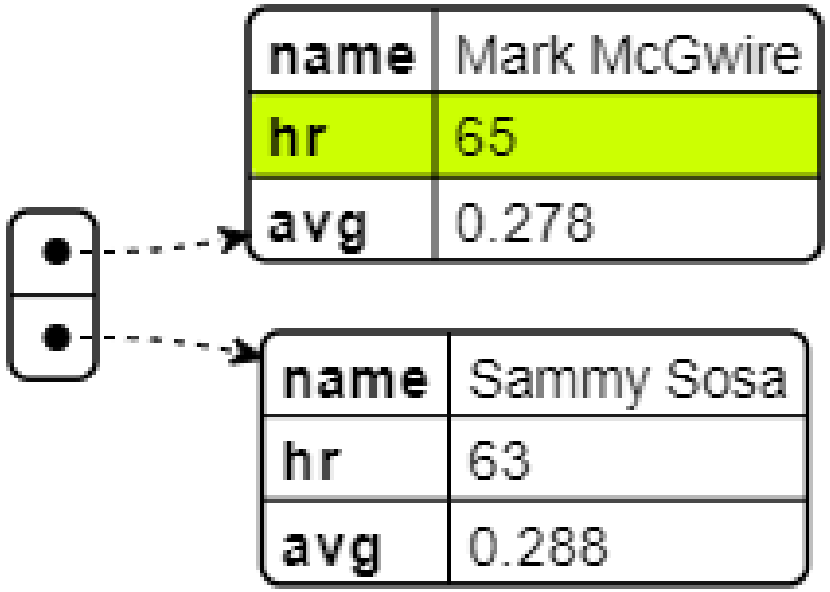
Work Breakdown Structure (WBS)


```
@startwbs
* Business Process Modelling WBS
** Launch the project
*** Complete Stakeholder Research
*** Initial Implementation Plan
** Design phase
*** Model of AsIs Processes Completed
**** Model of AsIs Processes Completed1
**** Model of AsIs Processes Completed2
*** Measure AsIs performance metrics
*** Identify Quick Wins
** Complete innovate phase
@endwbs
```

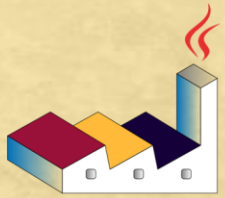


Visualizing JSON data

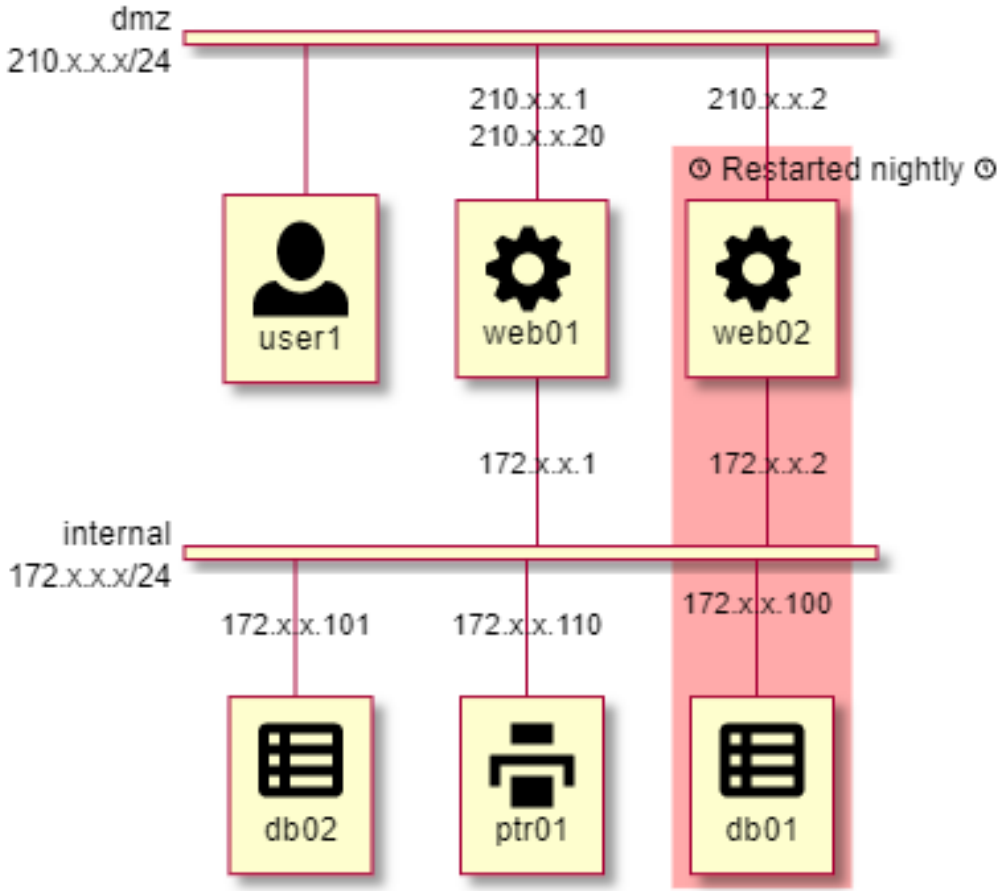

```
@startjson
#highlight "0" / "hr"
[
{
    "name": "Mark McGwire",
    "hr":   65,
    "avg":  0.278
}
,
{
    "name": "Sammy Sosa",
    "hr":   63,
    "avg":  0.288
}
]
@endjson
```



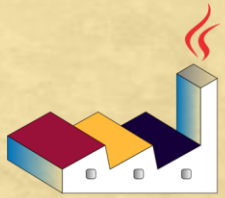
Network Diagram




```
@startuml

nwdiag {
    group nightly {
        color = "#FFAAAA";
        description = "<&clock> Restarted nightly <&clock>";
        web02;
        db01;
    }
    network dmz {
        address = "210.x.x.x/24"

        user [description = "<&person*4.5>\n user1"];
        // set multiple addresses (using comma)
        web01 [address = "210.x.x.1, 210.x.x.20",  description = "<&cog*4>\nweb01"]
        web02 [address = "210.x.x.2",  description = "<&cog*4>\nweb02"];

    }
    network internal {
        address = "172.x.x.x/24";

        web01 [address = "172.x.x.1"];
        web02 [address = "172.x.x.2"];
        db01 [address = "172.x.x.100",  description = "<&spreadsheet*4>\n db01"];
        db02 [address = "172.x.x.101",  description = "<&spreadsheet*4>\n db02"];
        ptr  [address = "172.x.x.110",  description = "<&print*4>\n ptr01"];
    }
    }
@enduml
```



Điểm mạnh của Plant UML

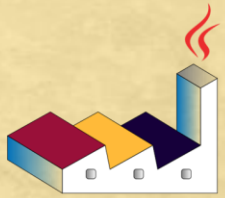


- Miễn phí, hệ sinh thái rộng lớn.
- Tài liệu hỗ trợ đầy đủ, chi tiết.
- Tạo sơ đồ từ văn bản thuần túy và con người có thể đọc được.
- Hỗ trợ làm việc nhóm trên các nền tảng quản lý mã nguồn.
- Thời gian tạo, chỉnh sửa sơ đồ nhanh hơn so với các công cụ trực quan.

Điểm mạnh của Plant UML

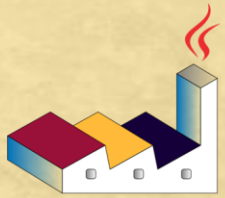
- Hỗ trợ hầu hết các sơ đồ UML và một số sơ đồ khác. Đặc biệt, với sơ đồ Sequence thì mã lệnh PlantUML có thể xem như mã giả, giúp lập trình viên không bỏ sót các trường hợp.
- Dễ dàng kết xuất ra ảnh (nhiều định dạng) từ tập tin mã lệnh hoặc ngay khi đang viết mã lệnh.
- Phù hợp với xu hướng "***everything as code***", bao gồm cả sơ đồ.

Hạn chế của Plant UML



- Thói quen của người dùng
- Tập cú pháp mã lệnh tương đối nhiều
- Bố cục của sơ đồ được quyết định bằng giải thuật tối ưu nên khó tùy chỉnh theo quan điểm cá nhân

Tài nguyên tham khảo



- PlantUML (plantuml.com)
- Tài liệu hỗ trợ, hướng dẫn sử dụng PlantUML (plantuml.com/en/guide)
- Trình soạn thảo PlantUML trực tuyến (http://www.plantuml.com/plantuml/uml/SyfFKj2rKt3CoKnELR1Io4ZDoSa70000)
- Ứng dụng soạn thảo PlantUML trực tuyến (plainttext.com)
- Visual Code