Làm cái gì để đạt đc chức năng đó system requirement

Công cụ để viết sý requirement là usercase

Functional req: là những yêu cầu đáp ứng dc những cái về nghiệp vụ của khách hàng , và những chưc năng mà là nhứng hành sử đáp ứng của hệ thống phản ứng lại so với cái khách hàng đưa vào( là để đáp lại login của người dung),, nó có thể nới những cái hệ thống nên làm và k nên làm k có k dc, chức năng khách hàng mông muốn

* ví dụ : thiết kế CPU siêu nhỏ. nhưng muốn tốc độ cao

Non-Functional là những ràng buộc trên nhũng chức năng mà hệ thống đưa ra thường là rang buộc trên toàn hệ thống để làm phần mềm tốt hơn (vd rang buộc thời gian,bảo mật …) k có vẫn hoạt động nhưng người dung không thích (cải thiện tốc độ, khả năng cũng cấp..) có thể mở rọng hoặc giới hạn ( thường là giới hạn. những cái ràng buộc này k là chức năng nhưng nó quyết dịnh phần mềm có tốt hay không

Domain k nhắn mạnh thường phân vào req

Tại sao nên có cái mẫu và gilie (trong mẫu)? cần mẫu để nhân viên ở các bộ phận cty dựa trên mẫu giúp viết tài liệu requirement đơn giản hơn tốt hơn (vd như người mới chưa biết viết)

Actor là người dùng hoặc đối tượng tương tác với một hệ thống. Mỗi Actor có thể là một người, một tổ chức hoặc một hệ thống bên ngoài tương tác với ứng dụng hoặc hệ thống của bạn, là người sẽ thực hiện 1 hành động nào đó để tác động vào hệ thống (hệ thống khác hoăc software)

Vd như guesh, student, teacher thì những ai cần logout? Kahchs thì chưa login, student và teacher login rồi, khách k sd logout còn lại sd thì nên sd kế thừa như tất cả người có tài khaonr thì có thể kế thùa

Vd như người dùng (con người: thu thu)

* hệ thống tương tác
* chatport là những hệ thống tự động thay thế cn người có thể tích hợp vào fb mess
* tại sao k nên dùng system như 1 actor vì actor là tác nhân bên ngoài (Actor phải là external entity)
* cái máy in : là actor khi tương tác với hệ thống, trong trường hợp chỉ tương tác đọc mess với máy in để in giấy từ thì ta k vẽ actor là máy in , trường hợp máy in xem mess ngược lại như khi máy in in xong sẽ báo in xong thì là actor máy in

Use Case là chức năng mà các Actor sẽ sử dụng

Mô tả usecase: admin -> manage user : mục tiêu có thể xem , thêm , sửa, xóa người dùng, đặt pass vì nó astract nó chung chung k thể hiện gì để mô tả ( Generalization- trong trừu tượng có những cái chi tiết)

Relationship hay còn gọi là conntector được sử dụng để kết nối giữa các đối tượng với nhau tạo nên bản vẽ Use Case. Có các kiểu quan hệ cơ bản sau:

–          Association

–          Generalization

–          Include

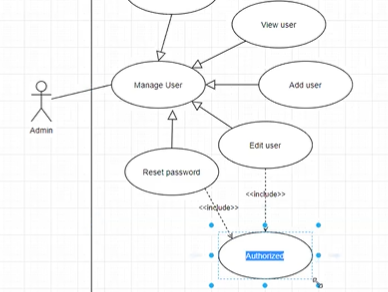
–          Extend

Association thường được dùng để mô tả mối quan hệ giữa Actor và Use Case và giữa các Use Case với nhau.

Generalization được sử dụng để thể hiện quan hệ thừa kế giữa các Actor hoặc giữa các Use Case với nhau.

Include là quan hệ giữa các Use Case với nhau, nó mô tả việc một Use Case lớn được chia ra thành các Use Case nhỏ để dễ cài đặt dễ quản lý (module hóa) hoặc thể hiện sự dùng lại, khi thực hiện a thì chắc chắn sẽ thực hiện b và b nhỏ hơn a, thì a thực hiện trước b thực hiện sau khi a include b

Mô tả reset pass hay edit: khi mà người dùng muốn resetpass thì bắt buộc mình thực hiện thao tác login…. Em nghĩ là tùy vào requirement, hay có thể là tùy vao fyc của kkhachs hàng : nếu khách hàng muons ad min đã đang nhập thì co stheer thực hiện tất cả thao tác hoặc….

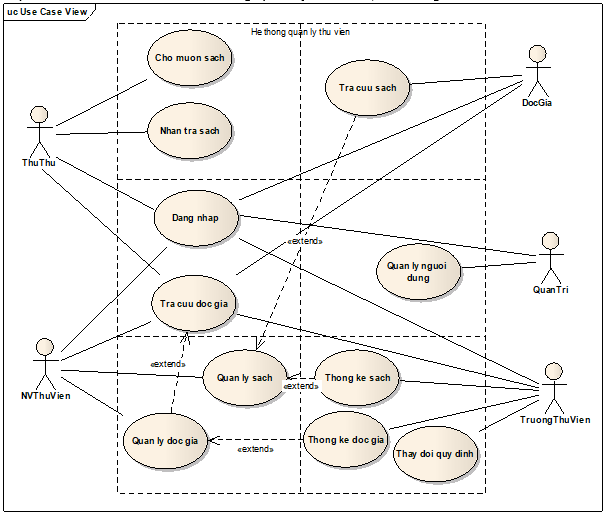


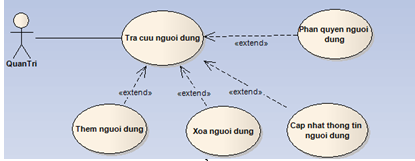
Extend dùng để mô tả quan hệ giữa 2 Use Case. Quan hệ Extend được sử dụng khi có một Use Case được tạo ra để bổ sung chức năng cho một Use Case có sẵn và được sử dụng trong một điều kiện nhất định nào đó. Nó là 1 quan hệ trực tiếp giữa a và b, trong đó b đucợ extend từ a khi thực hiện a thì thỏa 1 đk thì b mới dc thực hiện ( k phải lúc nào cung xảy ra)

Vd khi nào là include khi nào là extend ? tùy người dùng co sử dung j hay k nhưng nó vẫn xuất hiện là include

Nêú người dùng đánh giá 4 hay 5 sao thì thì nó mơi sxayr ra cho tìm tiếp thì là extend ( khi có đk)

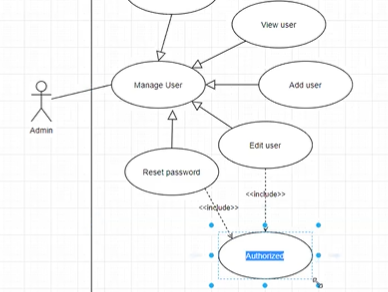
Như quản lý thư viện thì actor là thuthu, doc gia, quan tri, nv thư viên;





Software requirements

Đặc tả yc phần mềm:

* Công cụ use case uml để biết đắc tả
* Softwre req document : thường mô tả phát biểu 1 cách chính thức hay những các mà ta cần làm gồm user req và sys req , nó đa dạng về người đọc như custmer, manager, deverloper, test , support it…
* + trong tài liệu thường có:
* . perface: ai viết
* . mô tả chung những cái hệ thống cần, ngữ cảnh và nó giải quyêt snhungwx gì
* .
* ….. viết tài liệu req chi người khác đọc yc để nhìu đối tượng đọc có thể đọc dc , mà nó bao gồm nhìu từ ngữ chuyên môn nên cần bảng mô tả từ ngữ chuyên môn
* …… user req những yc khác hàng mong muốn
* …. sý as: nhìn vào thì biết hệ thống cần những gì, họ thấy dc những gì có sẵn
* .Sys req : bắt buộc, nó là detail lv của req mà từ đo snhinf vào ta biết mình làm những gì , nó chiếm chỗ nhiều nhất
* .sys..: mô tả thêm yc phân fmemef
* . app: hướng phát triển tương lai, như đợt này ta k làm cái gì nhưng lần sau ta phát triển dc
* . …tài liệu kềm theo để hiểu thêm
* .. index
* Điều cần lưu ý : vì tài liệu req là viết cho người khác đọc nên ta tìm từ ngữ đơn giản k dùng nhìu từ ngữ chuyên ngành nhiều
* Sys req là mô tả cho tiết hơn về user req , bắt đầu của sys design ( nhờ có dc tài liệ chi tiết nên viết dễ). chúng ta viết phân fmemef k mô tả ntn đó là của design nhưng nó nói cần làm nhưng xvieecj gì ( nói về yc). Có thể sd mô hình usercase hay erd. Thường k phải viết user req rồi viết design mà co stheer song song
* Vẽ ra mô hình
* Mô tả chi tiết từng use case : vì người đọc vào k biết design hay sao cho đúng cả, đúng với mỗi use case có 1 bảng mô tả chi tiết
* + trong cấu trúc có nhìu mẫu ,bản mô tả gồm : tên , chi tiết , ai làm , tên chức năng , sau khi chức năng thực hiện thì đạt dc những gì, các bước thực hiện
* - system k nên là 1 actor
* Mô tả : actor là cus, samary là cho phép cus thêm sp., mục tiêu là cho thêm sp thành công, khi nào xẩy ra là khi cus click vào giỏ, đk: là co sinternet, đk 2 là sp còn hàng, đk 3 là sx chưa bị remove, posconnection là đk đo lường là xong hay k xong, thành công hay thất bái, thành công là thêm dc sp và lưu dc xuống database, thất bại,…. Mô tả các bước cus avf sys tương tác , enternated co snhung bước khác mà ta cũng hoàn thành use nhưu bắt chọn kích thước, nhưng xcais kèm theo
* Vd đặt hàng có 2 con đường là thêm vào giỏ hàng và đạt hàng ngay,
* Những ngoại lệ xảy ra: vd như khi thêm thì mới ktra thấy k còn hàng thì ntn
* Relationship là usecase co liên quan đen những cái khác k
* Những nghiệp vụ: mỗi khách hangftaij 1 thời điểm chỉ dc chọc 3 sp, hay nếu như sp co snhieeuf màu sắc kích thước hay tùy chịn kèm theo thì khách bắt buộc chon 1 thuộc tính , hay khi thêm hàng phải ktra giỏ hàng
* Mô hình usecase theo mô hình uml mục đích miêu tả diagram ( behaver diagram) tương tác giữa cấc actor bằng hành động cụ thể, trong mô hình phải có baoerry ,
* Gồm actor. Usecase, các mỗi quan hệ, subject là cái baerry
* Actor là những cái gì có hành vi hành động và có nó có vai trò thực hiện ,,,, dc play bởi 1 người dùng hay 1 system khác hoặc 1 software hay hopware , 1 role có thể dc diễn tả bổi 1 thực thể vật lý nào đó hay nhiều thực thể vật lý, mỗi cái thì đống 1 vai trò
* Vd trong hệ thống qly phòng khám co actor bs, actoe taff, act admin, thông thường khi đưa ra actor mình sẽ có bảng mô tả chi tiết, actor doctor là 1 user là bs đã login vào hệ thống thì dc tương tác với user đó, staff đã login vào hện thống với vai trò nhận viên, admin là những người đã login vào hệ thống với với vai trò là admin, actor là nguoif login vào vớivai trò ,…
* Doctor là 1 user đã login với vai trò doctor Có cần login nữa k: thì phải đnhap rồi hệ thống mới biết là doctor mới cho sd các chức năng cho nen k cần đang nhập nữa, nhưng no tùy vào cách định nghĩa actor nữa.
* Nếu phòng lkham chỉ có 1 bs vừa là admin vauwf là bs thì khi hệ thống cũng cấp cho ổng những chức năng của bs như thêm toa … nếu login thì bằng vai trò admin thì thì dc thêm sửa xóa…
* Có luu ý là phải định nghĩa actor trước là khách hay người chưa dc đang nhập vài hệ thống , vd như student thì đã đang nhập thì k cần đang nhâp nnuawx nhưng họ có chức năng logout nhưng khin logout ròi thì hệ thống sẽ hiểu như khách nên sẽ cung cấp chức năng
* tại sao k nên dùng system như 1 actor vì actor là tác nhân bên ngoài (Actor phải là external entity)
* có 2 loại actor :
* +Primary actor những actor sd chức năng
* +supporting actor : cung cấp support cho hệ thống
* Pasioner là hành khách
* Actor cũng có quan hệ
* Generalization được sử dụng để thể hiện quan hệ thừa kế giữa các Actor hoặc giữa các Use Case với nhau , các actor trù tượng mà reli thì actor cụ thể cũng sd dc, ký hiệu giống uml class diagram ( kí hiệu thừa kế - mũi tên màu trắng huowg về abtract)
* Use case mô tả trường hợp sd của actor ký hiệu hình elip có tên bên trong, hoạc nằm bên dưới…, trong uml k quy định cách đặt tên nhưng ta thống nhất actor diễn tả = 1 hành động có 2 cách diễn tả là 1 câu diễn tả = động từ hay cụm danh từ có động từ, thầy quy định bắt đầu bằng động từ
* Mối quan hệ giữa use case và use case :
* + Association thường được dùng để mô tả mối quan hệ giữa Actor và Use Case và giữa các Use Case với nhau. Diễn tả đường nối k mũi tên và k có mối quan hệ associa tion giữa usecase và usecase mà giữa actor và usecase để diên xtar actor có hể tác động vào hệ thống bằng usecase, 1 usecase co stheer sd bởi nhiều actor,
* . đôi khi nó diễn tả chó sd chức năng 1 hay nhiều lần
* Có 3 mối quan hệ giữa các usecase:
* + Generalization: nó cũng có astract,
* Generalization được sử dụng để thể hiện quan hệ thừa kế giữa các Actor hoặc giữa các Use Case với nhau.
* Include là quan hệ giữa các Use Case với nhau, nó mô tả việc một Use Case lớn được chia ra thành các Use Case nhỏ để dễ cài đặt dễ quản lý (module hóa) hoặc thể hiện sự dùng lại, khi thực hiện a thì chắc chắn sẽ thực hiện b và b nhỏ hơn a, thì a thực hiện trước b thực hiện sau khi a include b
* -trong code tách ra để sáng sủa dễ hiểu hơn, sd trong trường hợp nhỏ hơn
* Mô tả reset pass hay edit: khi mà người dùng muốn resetpass thì bắt buộc mình thực hiện thao tác login…. Em nghĩ là tùy vào requirement, hay có thể là tùy vao fyc của kkhachs hàng : nếu khách hàng muons ad min đã đang nhập thì co stheer thực hiện tất cả thao tác hoặc….
* 
* Extend dùng để mô tả quan hệ giữa 2 Use Case. Quan hệ Extend được sử dụng khi có một Use Case được tạo ra để bổ sung chức năng cho một Use Case có sẵn và được sử dụng trong một điều kiện nhất định nào đó. Nó là 1 quan hệ trực tiếp giữa a và b, trong đó b đucợ extend từ a khi thực hiện a thì thỏa 1 đk thì b mới dc thực hiện ( k phải lúc nào cung xảy ra), extend là mối quan hệ mở rộng , mũi tên hướng về cái bị động, extand k nằm trên astracb mà nằm trên cái cụ thể
* Vd đk tài khoản mới có thể dc extend ra 1 cái mới là gesh khi nào nó xảy ra,
* Vd khi nào là include khi nào là extend ? tùy người dùng co sử dung j hay k nhưng nó vẫn xuất hiện là include
* Nêú người dùng đánh giá 4 hay 5 sao thì thì nó mơi sxayr ra cho tìm tiếp thì là extend ( khi có đk)
* Như quản lý thư viện thì actor là thuthu, doc gia, quan tri, nv thư viên;

//////////////////////////////////////////////////////

Những actor có những mối quan hệ: có quan hệ Generalization được sử dụng để thể hiện quan hệ thừa kế, gọi là quan hệ khái quát hóa

Có các actor :

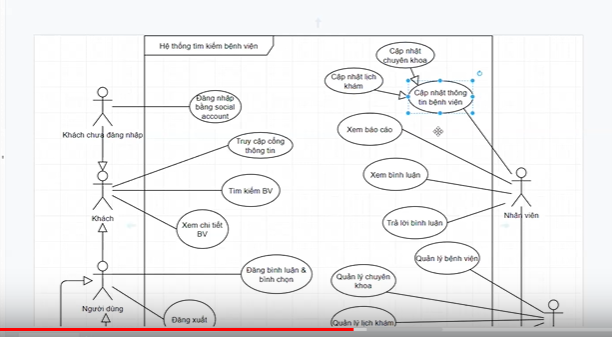
+ người quản trị : quản trị hệ thống: quản lý thactorông tin bệnh viện, quản lý chuyên khoa, quản lý lịch khám, quản lý người dung (quản lý nhân viên), quản lý bình luận, xem báo cáo ( thống kê)

+ người dung: thao tác tìm kiếm, quản lý danh mục bênh, quản lý bênh nhân: truy cap trang chủ, tìm kiếm (địa lý, chuyên khoa), xem thông tin bv, đang nhập social account, đăng bình luận, bình chọn

+ nhân viên: đăng nhập, xem bình luận, trả lời bình luận, cập nhật thông tin bệnh viện, xem báo cáo

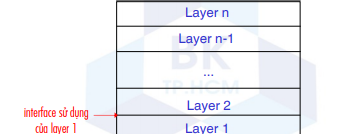
* Người quản trị có quan hệ generalization

+ người quản trị và người dùng có quan hệ với nhau, vì có thể tìm kiếm, người quantri kế thùa từ người dùng khi người quản trị có thể đăng nhập dc social account

* Nếu e là nhân viên của bv hay là người quản lý cổng thông tin bệnh viện thì họ có thể làm gì nữa? ẩn bình luận, viết bào review, thay đổi 1 số quy định, xem lượt truy cập, sắp xếp thông tin lịch khám, thông tin bệnh viện, thay đổi thông tin bv, tư vấn cho khách hàng, hộ trợ gợi y sthoong tin khoa khám bệnh => chăm sóc khách
* Cập nhật thông tin bệnh viện thì nên là generalization hay extend? Khi cập nhật xem có bắt buộc hay k, bếu bắt buộc thì include, xem nó có phải trù tượng hay cụ thể k (là cập nhật cái gì) , k biết cập nhật cái gì thig lá astract là generalization, include và extend là ccó những cái cụ thể
* 
* Truy cap thong tin bv có là usecase k? nếu truy cap thong tin chỉ là vào trang chủ thì k có j mô tả mà chỉ xem là 1 bước để tìm kiếm thì k cần tách usecase nếu nó phức tạp thì nên coi là 1 use case ( nếu như cần click vào mới vào cổng thông tin)
* Bình chọn và bình luận là 1 usecase thì đề bài có 1 hành động
* Nếu muốn logout thì them 1 actor khác là người dung đã đăng nhập(rắc rối vì them 3 quan hệ thừa kế)
* Mọi actor kế thùa từ người dung bình thường
* ////////////////////////
* Những nguồn lấy dc requirement mà k cần hỏi khách hang thì khi họ đưa report thì mình phân tích, hay từ hệ thống tương tự, dùng prototype , dung đăc điểm công ty khách hang, Tìm kiếm các chuyên gia, người sử dụng có hiểu biết về hệ thống cần xây dựng để thu thập được nhiều ý kiến, đóng góp khác nhau, khách hàng của khách hang, khảo sát môi trường, những tài liệu quy định chung
* Trong requirement tại sao lại có design : các dagram ở mức basic, design để khách xem trước, erd( làm trong req đểkhách xem trước), prototype, trong req đã làm kiến trúc hệ thống r( kiến trúc hệ thống là gì
* 4 phần: 1. Sofeware roset, 2. Specification làm reqirement: có 2 lv và detail level, sau khi có intput của design sẽ là output của specification, 3. Design 4.
* + 2 lv là gì
* + detail lv là gì
* - học rofactional software là làm phần mềm 1 cách chuyên nghiệp
* Yc thực tế cần làm gì nhưng mình k biết làm gì, vd như có 1 số việc thuộc quay trình như làm phần mềm cần đánh giá mức dộ khả thi, thì bước tiếp theo req recover ( khai thác yc phần mềm về phân ticgs ghi chép đánh giá rồi lại đi lấy yc…t aphair ghi chép lại và mô hình hóa lại thanh f các mô hìn thường các cty có các mẫu
* - yc cuối cùng của lấy yc phần mềm là đặt tả
* Thiết kế pahanf mềm chuyên nghiệp thường dc làm bởi 1 team trong đó có nhiều vai trò
* - phần mềm chuyên nghiệp đảm bảo phần mềm tốt và thương dc duy trì khi đi vào hoạt động
* System model là gì : số là 1 cái trừu trượng của sytem, nó k phải là thật nó là astraction
* Design hệ thông thường bắt đầu tự đâu: (databasefirst, databasemodel, lớn là databasefirst nhỏ là codefisrt, database-> ở detial) => ở mức 2 lv có context model có kiến trúc hệ thống (quyết định nhiều thứ về sau vè func req về thiết kế những thứ sau này)
* Đầu tiên xác định system boudaries ( là các chức năng chính và ta tập trung gì) từ sys boud.. thì ta .. từu mô hình tổng quát ta phan rã nó để developer biết làm những gì ( những người thợ chỉ viết code k cần suy nghĩ nhiều còn gọi là programer -> detail)
* => mô hình kiến trúc hệ thống vẫn cần
* Từ boudaries ta mới vẽ kiến trúc hệ thống
* Vd mô hình kiến trúc tỏng quát:
* + vd diễn tả system thôi, nó diễn tả về ,,,, của hệ thống như diễn tả mới quan hệ giữa các syetem khác
* - nếu làm căn nhà thì có nên thiết kế chi tiết bên trong trước k: cấu trúc thô của ngôn nha, ọc móc căn nhà => thường có bản vẽ trước do kiến trúc sư vẽ , vẽ cấu trúc ngôi nhà trước
* Làm phần mềm suy nghĩ kiến trúc trước
* Model phần mềm là gì : là 1 cái thể hiện trừu tượng của hệ thống mà chúng ta phải làm
* Như thế nào là trừu tượng: trừu tượng là mang ý nghĩ chung chung : dạ vi dụ như chức năng quản lý --> còn cụ thể là mình quản lý nhân sự hay quản lý nhân viên
* Như thế nào là cụ thể: hình là trừu tượng, hình tam giác là cự thể
* Mô hình hóa để biết nó co gì rồi , mô hình cả cái đang có và sắp ra
* Tùy nhìn vào khái canh gì đó(dự liệu, tương tác hành vi…) thì có mô hình hóa phù hợp
* Trong hệ thống atm có nhìu tp chúng liên kết với nhau ntn, chứ những cái gì: máy quát thẻ, bàn phím, màn hình giao diện, card swipper( nơi để thẻ), camera, vi xử lý bên hệ thống. cái nào kết nối với nhau qua kiến trúc hệ thống
* Đó là quy trình gì: là mô tả quá trình giam giữ k tự nguyện , quá tringf giam giữ bệnh nhân tâm thần: đầu tiên phải xác nhận quyết định bắt giữ, conform thông qua quyết định bắt giữ, vàohệ thống thì cso 2 nhánh công bố quyền bệnh nhân, ghi cheps quyết định bắt, khi dc chuyển khai có 2 hướng xem có quy hiểm hay k nguy hiểm . nêu sk nguy hiểm chuyển qua hệ thống amit rồi chuyể vào bv tâm thần nếu co snguy hiểm thì chuyển qua hệ thống chuyển qua nơi có bảo mật chuyển vào bệnh viện có hệ thống an ninh, nếu mà k tìm dc nơi nào thfi chuyển vào đòn cảnh xác sau bước này thì quay lại hệ thống chăm sóc sức khỏe bệnh nhân rồi công bố xã họi, gia đìng cập nhật thông tin => sơ đồ cho thấy tương tác giữa các hệ thống hiện có với nhau, giúp dc mô hình và quy trình nghiệp vụ giúp mô hình rộng hơn
* Hệ thống làm viêc theo mũi tên, thanh ngang là 2 cv có cùng thời điểm bắt đầu và kết thúc để chuẩn bị bước qua giao đoạn tiếp theo hẹ thống tiếp, dấu chấm đen là bắt đầu , còn kết thuc có vòng tròn bên ngoài, mũi tên cho biết tringf tự thực hiện(a->b a trước b), thanh ngang co smuix tên đi ra thfi các cv thực hiện ss, nếu cũng đi vào thì cũng kết thúc rồi mới đến cv tiếp theo, ký hiệu rẽ nhánh có thể nhiều nhánh có label để diễn tả khi nào thực hiện nhánh đó
* => quy trình tiếp nhận bệnh nhân tâm thần: đầu tiên xác nhận co bắt giữ k -> xác nhận quyềm bệnh nhân và ghi nhận quyết đinh bắt -> xét xem có nguy hiểm k -> đưa vào bệnh viện và ghi nhận ( k nguy hiểm) nếu mà nhiễm thfi đưa cách ly nếu có nơi chữa thfi đưa đến đó nếu chưa thì đưa đén nen hcuawx dc như trạm y tế, trạm cách ly, sau đó sẽ báo về địa phương gia đình-> cập nhật thông tin -> kết thúc
* Đầu tiên là nhân viên tiếp nhận y tế sẽ gọi hàm viewinfor FID cho ptatieninfor( hàm viewinfor của patien..) để xem thông tin rồi patient muốn có thông tin thì nó gọi hàm repord gồm tham số infor PID uid rồi tiếp theo muốn lấy dữ liệu thì cẩn gọi hàm authorize trong authorization để xem có quyền xem thông tin k và sau đó trả về thông tin ( message hay data) ( paahn biệt hàm có đống ngoặc) alt là đk rẽ nhánh: nếu user có quyền xem thì trả thông tin , neus bị lỗi hay k có quyền thì error, thương ftrar về cho user => activation ở trước sẽ lơn shonw hay chứa cái phía sau vì nó đồng bộ
* + mũi tên là lệnh gọi hàm , hàm report là của MHc
* Thông điệp bất đồng bộ, không cần đợi thông điệp đó được return lại kết quả, mà có thể bắt đầu gửi 1 thông điệp khác ở 1 luồng khác
* Sequence thì k cần sắp thứ tự thì gọi 1 cái nào đó , nếu gọi thì phải trả về ( hay chờ đợi trả về) recreate object là k có sẵn chỉ có khi gọi nó( code ở mức gọi hàm)
* - chúng ta cần làm gì với hệ thống cũ : có thể sd lại được, theo mô hình hệ thống xem kiến trúc còn hợp thời k, xem database , mô hình kiến trúc , mô hình class còn sd dc k…
* Activi diagram thường mô tả mô hình phụ
* Medical reception có case là viewinfo ( về mặt nghiệp vụ) gửi 1 request (1 click 1 link 1 nút) và nso gọi hàm viewinfo của patientinfo , tiếp tục tương tác với mhc = hàm report truyền thông tin infor và bên asthou chỉ tuongr tác khi xác thực thì gọi gọi authorize để check
* Mũi tên sd cần cẩn thận vì nó gây hiểu nhầm: mũi tên là cái pos nào truyền dữ liệu cho cái nào-
* Sự tương tác cho thấy hệ thông snayf gọi cái ào
* Architecture model thì ta có thể thảo luận cái nào mạnh cái nào yếu
* + quy trình : cái mô hình tùy kinh nghiệm quan điểm nên tùy cùng bài toán mô hình sẽ khác nhau
* + đưa ra quyết dụng làm ntn: xme coi có quy trình tương tự nào chưa
* Cách hệ thống chia nhỏ hệ thống để quản lý
* - có nhiều mô hình kiến trúc: mô hình kiến trúc 4+1, nếu vẽ theo locol view, nếu project view thì áp dụng activ hay sequent, development view nói vê fkyx thuật , physical view như hardware software
* - biết ngữ cảnh để vẽ mô hình kiến trúc
* Vẽ mô hình kiến trúc cho đề giữ kì và sequence diagram cho gửi lời hạn hò
* Kiến trúc MVC : MVC là từ viết tắt bởi 3 từ Model – View – Controller. Đây là mô hình thiết kế sử dụng trong kỹ thuật phần mềm. Mô hình source code thành 3 phần, tương ứng mỗi từ. Mỗi từ tương ứng với một hoạt động tách biệt trong một mô hình.
* Model (M):
* Là bộ phận có chức năng lưu trữ toàn bộ dữ liệu của ứng dụng. Bộ phận này là một cầu nối giữa 2 thành phần bên dưới là View và Controller. Model thể hiện dưới hình thức là một cơ sở dữ liệu hoặc có khi chỉ đơn giản là một file XML bình thường. Model thể hiện rõ các thao tác với cơ sở dữ liệu như cho phép xem, truy xuất, xử lý dữ liệu,…
* View (V): Đây là phần giao diện (theme) dành cho người sử dụng. Nơi mà người dùng có thể lấy được thông tin dữ liệu của MVC thông qua các thao tác truy vấn như tìm kiếm hoặc sử dụng thông qua các website.

Thông thường, các ứng dụng web sử dụng MVC View như một phần của hệ thống, nơi các thành phần HTML được tạo ra. Bên cạnh đó, View cũng có chức năng ghi nhận hoạt động của người dùng để tương tác với Controller. Tuy nhiên, View không có mối quan hệ trực tiếp với Controller, cũng không được lấy dữ liệu từ Controller mà chỉ hiển thị yêu cầu chuyển cho Controller mà thôi.

Ví dụ: Nút “delete” được tạo bởi View khi người dùng nhấn vào nút đó sẽ có một hành động trong Controller

* Controller (C):Bộ phận có nhiệm vụ xử lý các yêu cầu người dùng đưa đến thông qua view. Từ đó, C đưa ra dữ liệu phù hợp với người dùng. Bên cạnh đó, Controller còn có chức năng kết nối với model.
* Mỗi bộ phận thực hiện chức năng nhất định, nhưng chúng có sự thống nhất, liên kết với nhau tạo nên mô hình MVC. Mô hình này tương đối nhẹ. Nó có thể tích hợp được nhiều tính năng có trong ASP.NET hiện giờ. Ví dụ như authentication (quá trình xác thực).
* + Điểm mạnh: Nhẹ, tiết kiệm băng thông: MVC không sử dụng viewstate nên khá tiết kiệm diện tích băng thông. Khi sử dụng, người dùng có thể sử dụng ứng dụng trên web cần tương tác gửi và nhận dữ liệu một cách liên tục. Do đó, việc giảm băng thông giúp cho website hoạt động tốt và ổn định hơn.
* Kiểm tra dễ dàng: Với MVC, bạn có thể dễ dàng kiểm tra, rà soát lỗi phần mềm trước khi tới tay người tiêu dùng, đảm bảo chất lượng và độ uy tín cao hơn.
* Chức năng control: Trên các nền website thì ngôn ngữ lập trình như CSS, HTML, Javascript có một vai trò vô cùng quan trọng. Việc sử dụng mô hình MVC sẽ giúp bạn có một bộ control ưu việt trên nền tảng các ngôn ngữ hiện đại với nhiều hình thức khác nhau.
* View và size: View sẽ là nơi lưu trữ các dữ liệu. Càng nhiều yêu cầu được thực hiện thì kích thước càng tệp càng lớn. Khi đó, đường truyền mạng cũng giảm tốc độ load. Việc sử dụng mô hình MVC sẽ giúp bạn tiết kiệm được diện tích băng thông một cách tối ưu.
* Chức năng Soc (Separation of Concern): Chức năng này cho phép bạn phân tách rõ ràng các phần như Model, giao diện, data, nghiệp vụ.
* Tính kết hợp: Việc tích hợp ở mô hình MVC cho phép bạn thoải mái viết code trên nền tảng website. Khi đó, server của bạn sẽ được giảm tải khá nhiều.
* Đơn giản: Đây là một mô hình với kết cấu tương đối đơn giản. Dù bạn không có quá nhiều chuyên môn cũng có thể sử dụng được.
* + Điểm yếu: MVC thường được sử dụng vào những dự án lớn. Do đó, với các dự án nhỏ, mô hình MVC có thể gây cồng kềnh, tốn thời gian trong quá trình phát triển cũng như thời gian trung chuyển dữ liệu.
* MVC đang là mô hình được ứng dụng rất nhiều trong lập trình.
* Hệ thống MVC phát triển tốt sẽ cho phép phát triển front – end, back – end cùng trên hệ thống mà không có sự can thiệp, chia sẻ, chỉnh sửa các tập tin trong khi một hoặc hai bên vẫn đang làm việc.
* Việc sử dụng mô hình tương đối đơn giản. Chỉ cần hiểu rõ quy trình vận hành, nắm được các chức năng của từng bộ phận thì việc triển khai mô hình MVC tương đối dễ dàng.
* >
* Đặc điểm : nơi chứa dữ liệu, nơi tương tác và …. Tác biệt ra. Chứa dữ liệu tưng tác với dữ liệu và thực hiện thao tác trên dữ liệu/ contronler lấy dữ liệu từ model và đưa dữ liệu qua view
* Vd nếu muốn chỉnh sửa dữ liệu ta có thể k cần chỉnh sửa về model mà chỉ cần chỉnh sửa về view là dc
* Nó tách độc lập với nhau nên dễ chỉnh sửa, khi cần sửa cái này thì có thể k cần chỉnh cái kia( như chỉ chỉnh sửa về mặt giao diện…) -> ưu điểm / khi làm việc với team co snhunwgx kỹ năng riêng thì co stheer giao việc dễ hơn
* Yếu điểm: với những phần mềm đơn giản, dễ xử lý k cần làm MVC ( vì nó cần nhiều code hơn)
* Đặc điểm, ưu điểm nhược điểm của mô hình kiến trúc đa tầng
* + đặc điểm: phân ra nhiều tầng và mỗi tầng làm 1 cv khác nhau, và tằng trên muốn giao tiếp thì hcir giao tiếp với tầng liền kề ( thường là vậy)
* \* user interface: trình bày giao diện người dùng( gần giống view) nhưng không giống do nó có thể là 1 ứng dụng ( nó ở quy mô lớn hơn)
* \* tầng riêng của ứng dụng dùng để xác thực người dụng
* \* tầng xử lý nghiệp vụ (k thể k có): những cá utility của hệ thống: xử ly slogic, nghiệp vụ…
* \* tầng lưu trữ dữ liệu database
* Tầng có thể là 1 úng dụng là 1 nền tảng riêng, 1 server riêng biệt
* Fb cũng có kiến trúc đa tầng: vì ta có thể sd hệ thống authoization của fb đang nhập trên các ứng dụng khác nên nó phải có tầng riêng
* + fb có ứng dụng trên mobile trên destop
* + lợi điểm:
* + yếu điểm:
* Theo ngụ ý của tác giả, kiến trúc đa tầng là một cách cấu trúc project theo 4 tầng chính:
* +presentation: chứa các thành phần đảm nhiệm việc gửi phản hồi và hiển thị giao diện phía người dùng
* +application: chứa logic code tương ứng với tính năng của ứng dụng
* +domain: tầng này biểu diễn các thực thể hoặc service
* Ưu điểm:
* Chúng ta chỉ cần hiểu những lớp bên dưới lớp chúng ta đang làm;
* Mỗi lớp có thể được thay thế bởi một thể hiện tương đương (equivalent implementation), không ảnh hưởng đến các lớp khác;
* Một lớp có thể được sử dụng bởi một số lớp cấp cao khác nhau.
* Những bất lợi là:
* Lớp không thể đóng gói (encapsulate) tất cả mọi thứ (một field được thêm vào UI, rất có thể cũng cần phải được thêm vào DB);
* Các lớp bổ sung (extra) có thể gây ảnh hưởng đến hiệu suất, đặc biệt nếu ở các tier khác nhau.
* +infrastructure (hay còn gọi là persistence layer): các đối tượng nằm ở tầng này liên quan tới thao tác với cơ sở dữ liệu, Data Access Object,...
* 2 quy tắc bất biến của kiến trúc phân tầng mà mọi lập trình viên phải tuân theo gồm có:
* +Tất cả các mối ràng buộc (dependecy) chỉ được đi theo 1 chiều, từ tầng cao xuống tầng thấp hơn.
* + Logic của mỗi tầng độc lập với nhau (Ví dụ: tầng presentation không chứa các câu truy vấn CSDL...)
* 
* **Đặc tả: Hệ thống gồm nhiều cấp chức năng dạng chồng lên**
* **nhau, mỗi layer có chức năng cụ thể, rõ ràng và cung cấp các**
* **dịch vụ cho layer ngay trên mình. Layer cấp thấp nhất chứa các**
* **dịch vụ cơ bản nhất và ₫ược dùng cho toàn hệ thống**
* ** Tình huống nên dùng : xây dựng thêm khả năng mới trên hệ**
* **thống có sẵn, hay khi có nhiều nhóm phát triển khác nhau, mỗi**
* **nhóm chịu trách nhiệm về 1 layer chức năng cụ thể, hay khi có**
* **yêu cầu bảo mật nhiều cấp.**
* ** Ưu ₫iểm : cho phép hiệu chỉnh bên trong layer bất kỳ sao cho**
* **interface không ₫ổi. Có thể giải quyết 1 chức năng nào ₫ó (xác**
* **nhận user) ở nhiều cấp theo cách thức tăng dần.**
* ** Khuyết ₫iểm : khó tách (chức năng) của từng cấp ( như là tách ra mấy tầng, chức năng), layer trên khó tương tác với layer phía dưới nó nhưng không liền kề. Hiệu quả giảm sút khi nhiều layer phải tương tác nhau ₫ể giải quyết 1 chức năng nào đó**
* Mô hình repository architecture:

Kiến trúc kho (Repository Architecture)

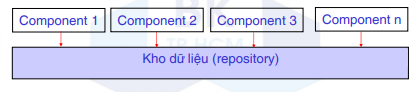
 Đặc tả : Tất cả dữ liệu của hệ thống ₫ược quản lý trong 1 kho

chứa tập trung, mọi thành phần chức năng của hệ thống ₫ều có

thể truy xuất kho chứa này. Các thành phần không tương tác trực

tiếp với nhau, chỉ thông qua kho chứa tập trung. Kiến trúc co stheer chia sẻ dữ liệu cho nhiều ứng dụng khác nhau

vd như là githud là nơi quản lý dữ liệu code mà mình post lên, và quản lý source => có 1 nơi trung tâm chứa dữ liệu và có nhiều ứng dụng xung quanh, có 1 dữ liệu ddauw dữ liệu vào kho và các ứng dụng khác có thể pull về sài



 Tình huống nên dùng : khi hệ thống tạo và chứa 1 lượng rất lớn

thông tin trong thời gian dài, hay trong các hệ thống dựa vào dữ

liệu, ở ₫ó việc chứa thông tin vào kho sẽ kích hoạt 1 tool hay 1

chức năng hoạt ₫ộng.

 Ưu ₫iểm : các thành phần ₫ộc lập nhau, không ai biết gì về ai

Khác( các ứng dụng con k cần biết dữ liệu đó ở đâu có).,có thể share lượng dữ liệu lơn scho nhiều uwgns dụng, vì dữ liệu tập trung ở 1 kho nên khi ta bảo mật hay backup thì nó nằm ở 1 chỗ thôi

 Khuyết ₫iểm : kho là ₫iểm yếu nhất, nếu có lỗi sẽ ảnh hưởng

toàn bộ các thành phần chức năng. Có vấn ₫ề về truy xuất ₫ồng

thời kho, phân tán kho trên nhiều máy cũng khó khăn.

 Thí dụ : Môi trường IDE gồm nhiều thành phần dùng chung kho

thông tin, mỗi tool tạo thông tin và ₫ể trong kho ₫ể các tool khác

dùng.

Kiến trúc client-server (client-server Architecture)

 Đặc tả : Hệ thống gồm 2 loại phần tử chức năng : server cung cấp

1 số dịch vụ( nơi xử lý), client là phần tử sử dụng dịch vụ bằng cách truy xuất

₫ến server tương ứng( nhận dữ liệu…) có thể trên sql server,,, web server, ….

- là mô hình chạy thông qua mạng (internet), … mỗi server có thể có nhiều client gửi yc cùng lúc



 Tình huống nên dùng : khi database dùng chung từ nhiều vị trí

khác nhau hay khi tải hệ thống thay ₫ổi ₫ộng (nhân bản server

thành nhiều phần tử).

 Ưu ₫iểm : server có thể phân tán tự do trên mạng, tập trung các xử lý về 1 nơi, client k cần cấu hình máy cao do k cần quá nhìu xử lý, server cần cấu hình máy cao hơn do xử lý nhiều việc

 Khuyết ₫iểm : ₫ộ hiệu quả phụ thuộc vào mạng và hệ thống nên

khó lường trước. các server ₫ược quản lý dc tập trung quản lý backup server

vd : Hệ thống quản lý phim ảnh dùng mô hình client-server

////////////

Node.js là một cách để chạy JavaScript trên server; nhưng nó còn hơn thế nữa. Nếu bạn là một người có hứng thú trong việc phát triển web

E quản lý theo mô hình client server, đối với server thì thông qua các ứng dụng web hoặc ….

/////////////////

- kiến trúc hệ thống giúp xác định các thành phần

1. Mô hình MVC: là khá niệm chung cso thể áp dụng cho nhìu ngữ cảnh ứng với mỗi trường hợp nó sinh ra các sd khác nhau

- Kiến trúc MVC : MVC là từ viết tắt bởi 3 từ Model – View – Controller. Đây là mô hình thiết kế sử dụng trong kỹ thuật phần mềm. Mô hình source code thành 3 phần, tương ứng mỗi từ. Mỗi từ tương ứng với một hoạt động tách biệt trong một mô hình.

- Đặc điểm: contronler lấy dữ liệu từ model và đưa dữ liệu qua view

+ M: lưu trữ dữ liệu, xử lý dữ liệu

+ V: giao diện người dùng, chịu trách nhiệm hiển thị

+ C: xử lý các yêu cầu người dùng đưa đến thông qua View, cái giúp người dùng tương tác với hệ thống của chúng ta như người dùng controller để điều khiển,

Vd lập trình wed: nguoif dùng gửi qua cỏtroller ….

- Điểm mạnh:

+ vì tách ra 3 phần riêng biệt nên dễ dàng kiểm tra và xét lỗi

+ sửa controller thì ko ảnh hưởng đên views vì mô hình này đã tách biệt phần xử lý và hiển thị

+ tách và giao công việc dễ cho nhiều team cùng thực hiện 1 dự án

- Điểm yếu:

+ thường sử dụng trong những dự án lớn

+ do đó trong các dự án khá đơn giản như gồm vài trang wed thôi (giao diện người dùng và datadase) thì MVC trở nên cồng kềnh, tốn tgian cho việc tách biệt 3 phần.

1. Layered - phân tầng:

- tối thiểu- tối đa là 2 - n

- thường từ 2-5 tầng

- Đặc điểm:

+ phân ra nhiều tầng, mỗi tầng chịu 1 trách nhiệm, cv khác nhau; mỗi tầng đc chia cho mỗi team dev khác nhau, có bao nhiêu tầng tùy vào đặc điểm thiết kế

+ theo 1 thứ tự nào đó: tầng trên chỉ có thể giao tiếp đc với tầng trên hoặc dưới kề nó, nếu 1 tầng có vấn đề thì nó chỉ ảnh hưởng đến tầng liên kề

+ phân tầng có thể là 1 ứng dụng riêng bao gồm các kiến trúc nhỏ bên trong, có thể chứa 1 MVC

+ mỗi tầng có thể sd công nghệ, kỹ thuật khác nhau, co stheer deloy trên cấc server khác nhau

- Ưu điểm:

+ thường áp dụng trong các ứng dụng lớn( vẫn áp dụng dc hco dự án lớn nhưng ít tầng lại)

+ tăng tính bảo mật

+ tăng tính độc lập

- Nhược điểm:

+ gây khó khăn cho người mới lập trình, ko có cái nhìn tổng quát

+ do giao tiếp của các tầng có thể sd nhiều kỹ thuật khác nhau nên tốn nhiều tgian

3. Repository

- Đặc điểm:

+ là kiến trúc tập trung dữ liệu vào 1 nơi và chia sẻ dữ liệu cho nhiều ứng dụng khác nhau (git)

- Ưu điểm:

+ share lượng dữ liệu lớn cho ứng dụng

- Nhược điểm:

+ sub-system phải tuân theo 1 chuẩn nào đó, phải kết nối với nhau

+ mỗi lần thay đổi kiến trúc của repository

thì client liên quan có thể bị hỏng

- sd khi share data. Hay sd hệ thống con đưa dữ liệu về 1 chỗ mà k làm việc cnugf với nhau

- có thể bị xung đột dữ liệu, do làm việc theo đồng bộ dữ liệu

- mô hình pipe anh filter : nó như 1 project

 Đặc tả : Nới rộng kiến trúc lô tuần tự lên tầm cao mới :

 Các filter không nhất thiết là phần mềm ₫ộc lập lẫn nhau,

chúng có thể là các thread chạy trong 1 chương trình.

 Có thể có nhiều ống con trong từng ₫oạn xử lý.

Kiến trúc ₫ường ống và lọc (Pipe and filter Architecture)

 Tình huống nên dùng : trong các ứng dụng xử lý dữ liệu mà dữ

liệu nhập cần ₫ược xử lý bởi nhiều công ₫oạn khác nhau và có

tính ₫ộc lập cao trước khi tạo ra kết quả cuối cùng.

 Ưu ₫iểm : dễ dàng thay ₫ổi/bảo trì/dùng lại từng filter của hệ

thống, phù hợp với nhiều hoạt ₫ộng nghiệp vụ, dễ dàng nâng cấp

bằng cách thêm filter mới, hiệu quả cao hơn kiến trúc lô tuần tự.

 Khuyết ₫iểm : 2 filter kề nhau cần tuân thủ ₫ịnh dạng dữ liệu

chung.

Bt về nhà: lên trag uml dagram

Lick trang chủ vào coponent dagram

Vẽ component diagram về bt giữa kỳ, vẽ sequence theo mvc gửi yc hẹn hò( người dùng đang trên trang kq tìm kiếm click vào nút hẹn hò và system làm sao ghi nhận, k có việc đối tác chấp nhận hay từ chối)

////////////////////

- Design hệ thông thường bắt đầu tự đâu: (databasefirst, databasemodel, lớn là databasefirst nhỏ là codefisrt, database-> ở detial) => ở mức 2 lv có context model có kiến trúc hệ thống (quyết định nhiều thứ về sau vè func req về thiết kế những thứ sau này)

- Design hệ thông thường bắt đầu tự đâu -> Đầu tiên xác định system boudaries ( là các chức năng chính và ta tập trung gì) từ sys boud thiết kế mô hình kiến trúc( như phát họa tổng quang những object tan cần vẽ-> xđ thành phần hay system làm nên hệ thống -> hay chia nhỏ hệ thống lớn thành con và mối quan hệ ).- có nhiều loại mô hình kiến trúc. thì ta .. từu mô hình tổng quát ta phan rã nó để developer biết làm những gì ( những người thợ chỉ viết code k cần suy nghĩ nhiều còn gọi là programer -> detail)

- diễn tả về hardware soft coi cái nào tham gia vào hệ thống physical view

- nhìn về development thì chia nhỏ ra cco những system nào sd công nghệ gì….

Kiến trúc hệ thống thường theo mô hình bosandline

Khi sd mũi tên phải xem kỹ coi chiều liên hệ…

Với 1 hệ thống lớn mà k có kiến trúc thì sẽ sai sót thiếu 1 số thứ hay đánh giá thấp 1 số thứ hay k co soverview nào để đánh giá-> do đó cần biết quy trình để làm giảm thiểu thời gian giảm sai sót ai cũng hiểu

- design bắt đầu từ sẽ ktra hệ thống thiết kế ntn rồi thiết kế kiến trúc rồi vẽ mô hình

- sau khi có kiến trúc rồi sẽ thiết kế thành phần của kiến trúc -> sau khi có thành phần rồi thì thiết kế các class đối tượng trong đó

- các loại uml digram là usecase, seq, component, activity, class diagram

- các loại dỉgram cần thiết kế trong design: usecase, seq, component, class diagram, pakage digram

- quy trình thiết kế phần mềm activity design diễn ra song song với requirement ( để biết trước rủi ra hay cần làm gì) -> từ kiến kiến chia nhỏ ra (design hay chia nhỏ hệ thống lớn thành con và mối quan hệ chia đến nhỏ nhất là detail design) sau đó chia nhỏ ra thành các lớp -> sau đó có bảng mô tả chi tiết từng lớp( vẽ diagram không đủ) ( thường mô tả dưới dạng bảng) -> kèm với tùng diagram thường kèm theo …

- yếu điểm là thiết kế giao tiếp do nó đồi hỏi kỹ năng nghệ thuật,

Từ mô hình kiến trúc phần rã thành các component -> xđ các lớp theo mô hình mvc… ->

Trong gr có bn controller -> mỗi giao dinej 1 controller-> phân tích contr co snhung thuộc tính phương thức-> mối liên quan giữa các class ( classdigram) -> mô hình xử lý sequen diagram (mô hình tương tác giữa các component)

- mẫu kiến trúc: kiến trúc mvc, clientserver, đa tầng layer, kho, Pipe and filter( là kiến trúc trải qua nhiều hệ thống xử lý liên kết thành 1 chuỗi mà đầu vào của hệ thống này là đầu ra của hệ thống khác) -> 1 hệ thống gồm nhiều mẫu k nhất nhiết là 1 mẫu nhưng mà mẫu nào là mẫu chính của kiến trúc đó

- mô hình kiến trúc phân rã nhỏ ra thành component rồi xđ các lớp theo nghiệp vụ

////////////////

- Design bắt đầu từ ktra hệ thống cần phải thiết kế ntn -> thiết kế kiến trúc hệ thống -> vẽ mô hình uml

- các loại uml digram là usecase, seq, component, activity, class diagram

- trong các loại mô hình uml phân ra làm 4 loại ( hay các mô hình design nói chung cũng phân ra làm 4 loại) :

1. actornal profectied (có những hệ thống nào liên kết với nhau trong 1 hệ thống lớn – có Biểu đồ hoạt động (Activity Diagram) mô tả những xử lý)

2. Structural model ( mô tả cấu trúc dữ liệu) có Biểu đồ lớp (Class Diagram)

3. behavior model ( mô tả hành xử của hệ thông skhi có event xảy ra ) có state machine

//- usecase model ( tương tác giữa bên ngoài tác đồng bên trong)

4. Interaction diagram: Biểu đồ trình tự (Sequence Diagram): tương tác giữa các object bên trong bên ngoài về kỹ thuật

Ngữ cảnh Activity: xũ lỹ phương trình ,dữ liệu

User case: giúp hiểu sự tương tác giữa các actor và hệ thông bên trong

Sequen : diễn tả sự tương tác bên trang hệ thống với actor cũng như giuawc các các cpmponent

Cấu trúc Class : diễn tar cấu trúc dữ liệu giữa các ọect

HÀnh xử: State: Thuộc về diễn tả hệ thông hành xữ lại khi có event xảy ra

- Design hệ thông thường bắt đầu tự đâu -> Đầu tiên xác định system boudaries ( là các chức năng chính và ta tập trung gì hay còn gọi là hệ thống con) từ sys boud thiết kế mô hình kiến trúc( như phát họa tổng quang những object tan cần vẽ-> xđ thành phần hay system làm nên hệ thống -> hay chia nhỏ hệ thống lớn thành con và mối quan hệ ).- có nhiều loại mô hình kiến trúc. thì ta phân rã nó ra thành các thành phần nhỏ hơn( thành component diagram).. từu mô hình tổng quát ta phan rã nó để developer biết làm những gì ( những người thợ chỉ viết code k cần suy nghĩ nhiều còn gọi là programer -> detail)

− Biểu đồ lớp (Class Diagram) : cho biết cấu trúc của 1 object và mối quan hệ đến các ojbect khác ( chĩ thấy mặt tĩnh) -> để diễn tả tương tác cần seqence diagram -> khi object tương tác với nhau ntn thì vẽ bihavior model

- sau design cần thực thi cần chú ý

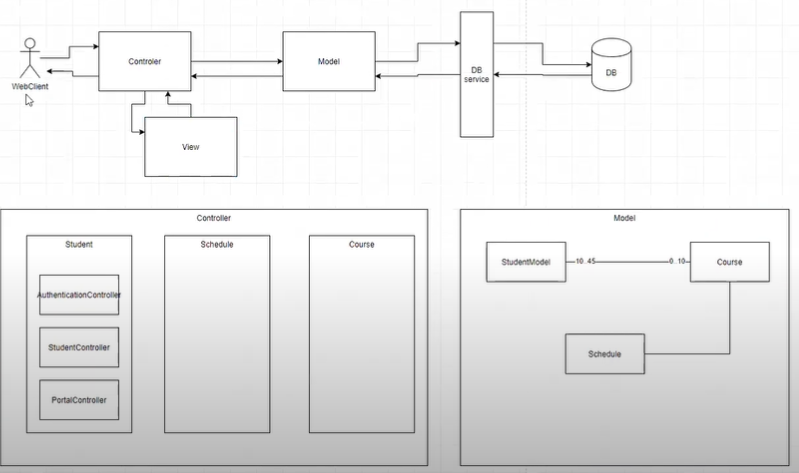
- Class diagram

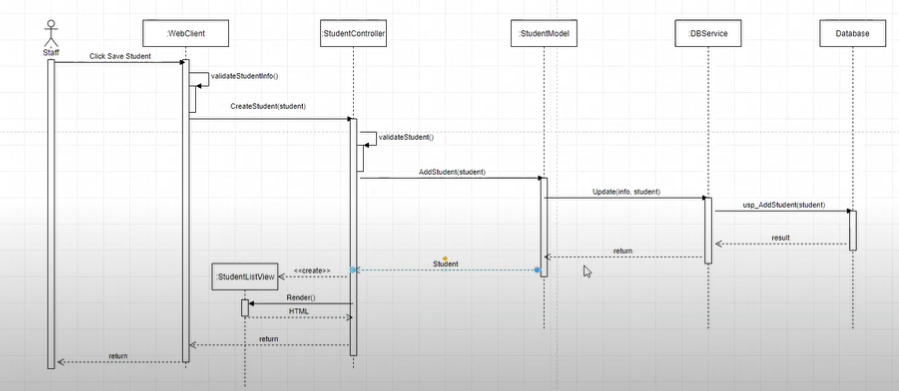
- trạng thái và chuyển đổi trạng thái là state machine

- Class diagram có nhìu level (chi tiết hay ít chi tiết hơn) ở đây có 2 mức độ:

+ Domian model diagram: mô tả tên và thuộc tính object

+ Implementation: có nhiều ký hơn và có phương thức thuộc tính





- Biểu đồ use case (Use Case Diagram)

• Biểu đồ hoạt động (Activity Diagram)

• Biểu đồ tương tác (Interaction Diagrams)

− Biểu đồ trình tự (Sequence Diagram)

− Biểu đồ giao tiếp/cộng tác (Communication/Collaboration Diagram)

• Biểu đồ trạng thái (Statechart Diagram)

• Biểu đồ cấu trúc tĩnh (Static Structure Diagrams)

− Biểu đồ lớp (Class Diagram)

− Biểu đồ đối tượng (Object Diagram)

• Biểu đồ thực thi (Implementation Diagrams)

− Biểu đồ thành phần (Component Diagram)

− Biểu đồ triển khai (Deployment Diagram

1. Class diagram:

Vẽ class diagram bang draw.io cho yêu cầu quản lý phòng trọ.

- Mô tả ví dụ quản lý thư viện (như hình): Trong hình này có library => library có nhiều cuốn sách (book) => sách thì chia thành catalog (thư mục sách). Có nhiều người mượn sách (patron), mỗi người có 1 Account trong hệ thống => Account thì có nhiều trạng thái (account state), Sách (book) thì có tác giá (Author), librarian là người thủ thư.

- Class diagram có nhìu level (chi tiết hay ít chi tiết hơn) ở đây có 2 mức độ:

+ Domian model diagram: mô tả tên và thuộc tính object

+ Implementation: có nhiều ký hơn và có phương thức thuộc tính

Book là lớp trừu tượng => BookItem là lớp cụ thể.

Dấu “/” thuộc tính

Class tách 3 phần : tên lớp tên actibile, tên phương thức , tên function

1. xác định thàng nào chứa thàng nào , thàng nào chứa thằng nào

- Aggregation (thu nạp): trong object của classA có chứa object của classB => object của classA bị hủy thì object của classB vẫn có thể tồn tại. A bị hủy chưa chắc B bị hủy

- Composition (hợp thành): chặt chẽ hơn, trong object của classA có chứa object của classB => object của classA bị hủy thì object của classB ko thể tồn tại (bị hủy). A mất thì tất cả B đều mất; B mất ko ảnh hưởng đến A

- classdiagram cho quản lý thư viện

2. State Machine:

behavior

- Diễn tả được các trạng thái của các object trong hệ thống khi có 1 số event hiện ra

- Mục đích: giúp chúng ta hiểu được sự chuyển đổi trạng thái của object hoặc component, system tùy vào stimulus(tác động, kích thích, event)

- imlementation :

+ lưu ý :

+ designe patterns: khi design đến đâu coding đến đó

Giúp code dễ hiểu rõ ràng mọi người làm nhue nhau

Cái component nào có sẵn, cần chọn cái nào cần mua bn tiền

Môi trường test ntn của từng người, tester có đủ môi trường test k

Môi trường hoạt đồng thực tế có những cái khác so với môi ttruong test

Việc sd lại: có thể k làm mới tất cả mọi thứ( có thể sd lại: frame work free,..); sd lại các object( vd như sd lại class…); sd lại nguyên 1 component; sd lại dạng system

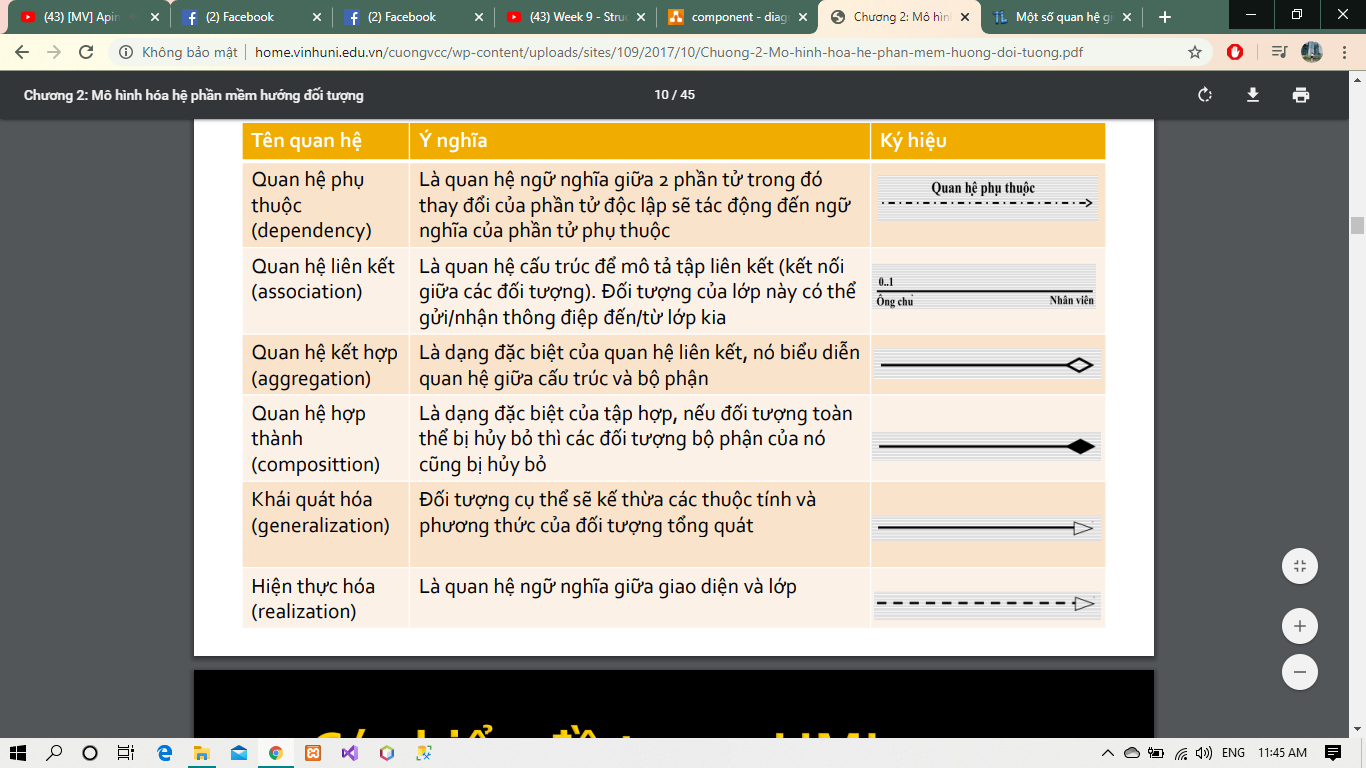
Sd lại dưới dạng có sẵn reuse chưa chắc tốn ít : như tốn thời gian nghiêm cứu

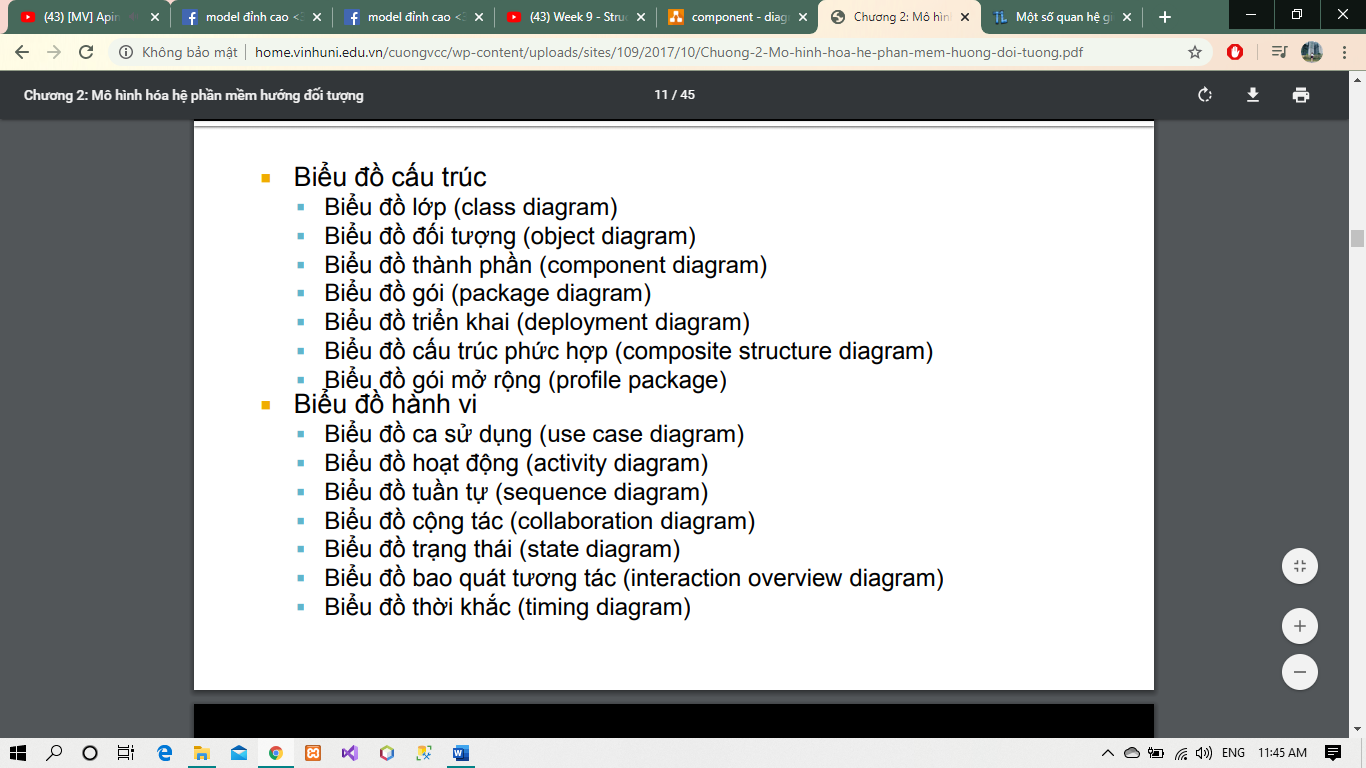
Host-target dev: database nằm chung 1 chỗ nhưng họ chạy trên hệ thống khác nhau

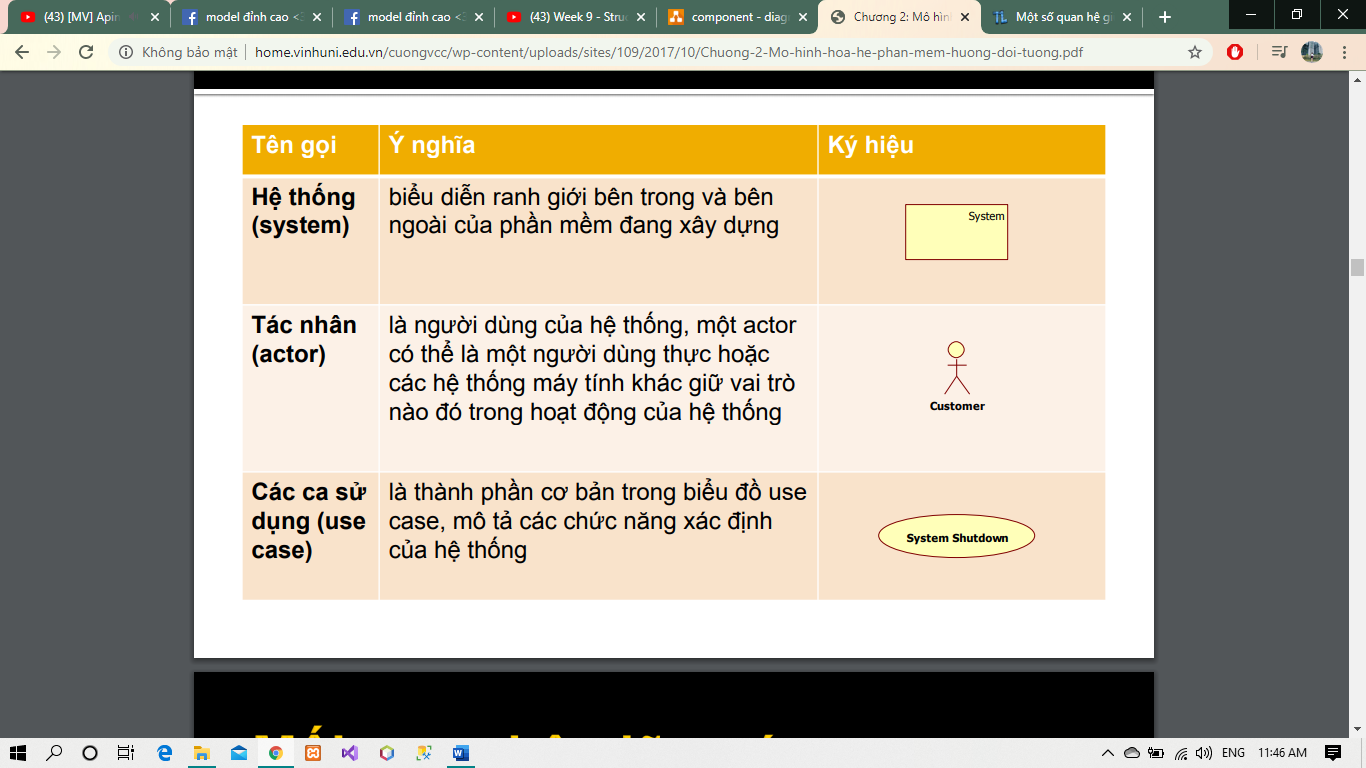
Open source dev: có đáng tin cậy k

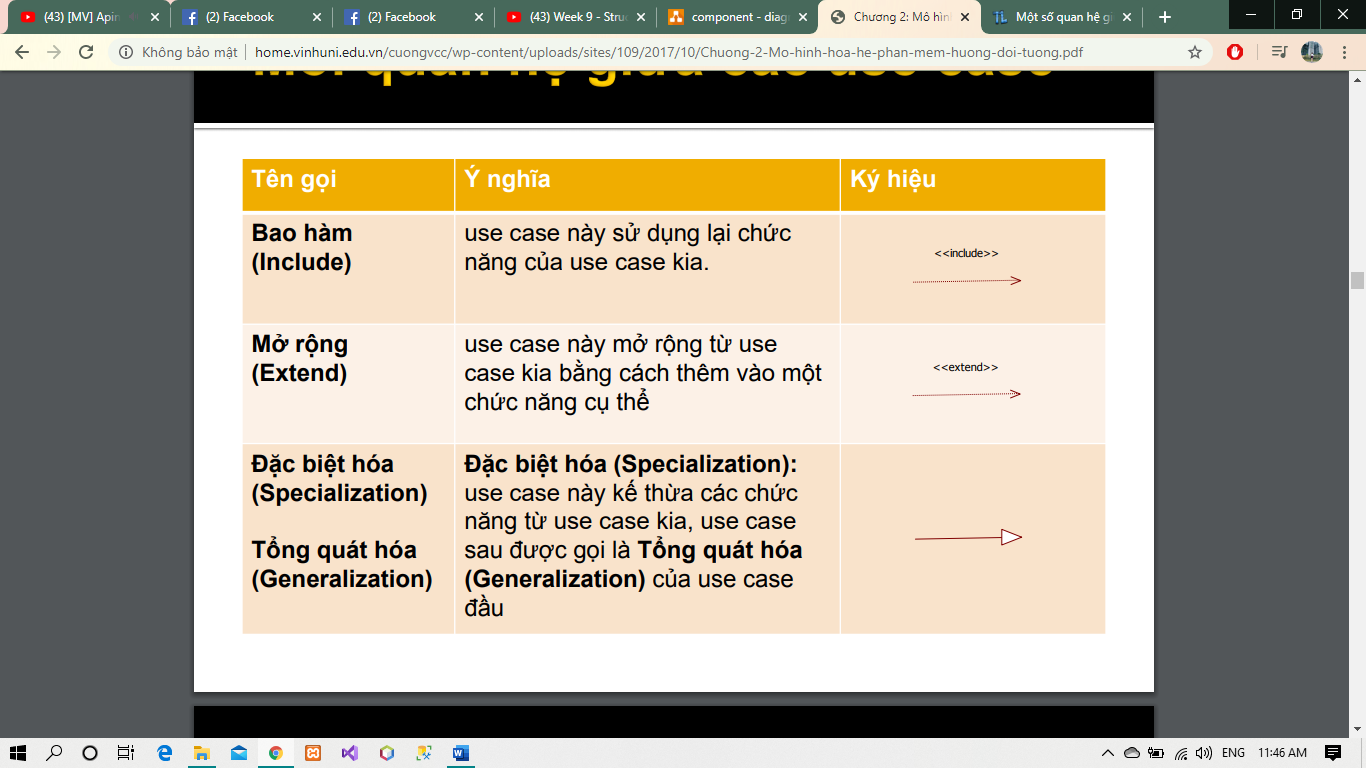
+ licenses: GPL(

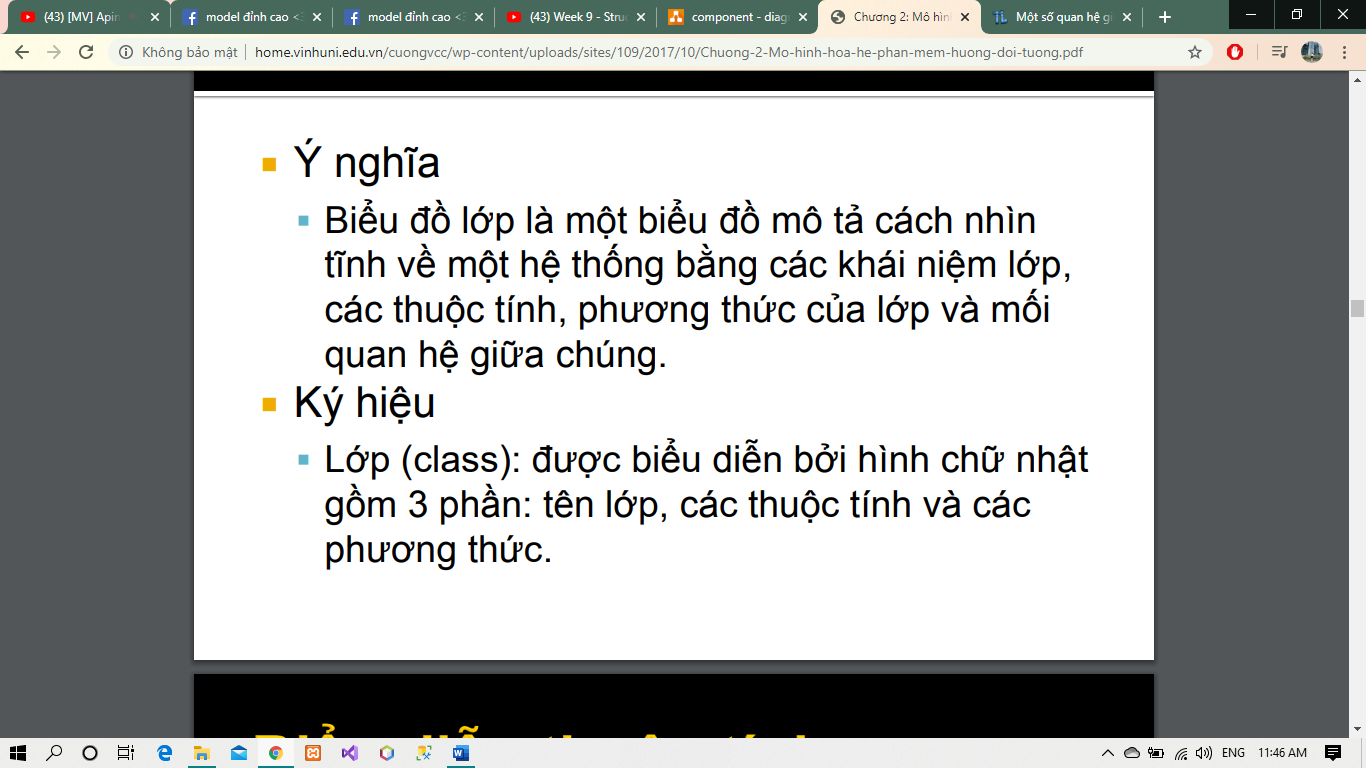
Bt về nhà : vẽ class diagram cho 1 nghiệp vụ trong bt cuối kỳ

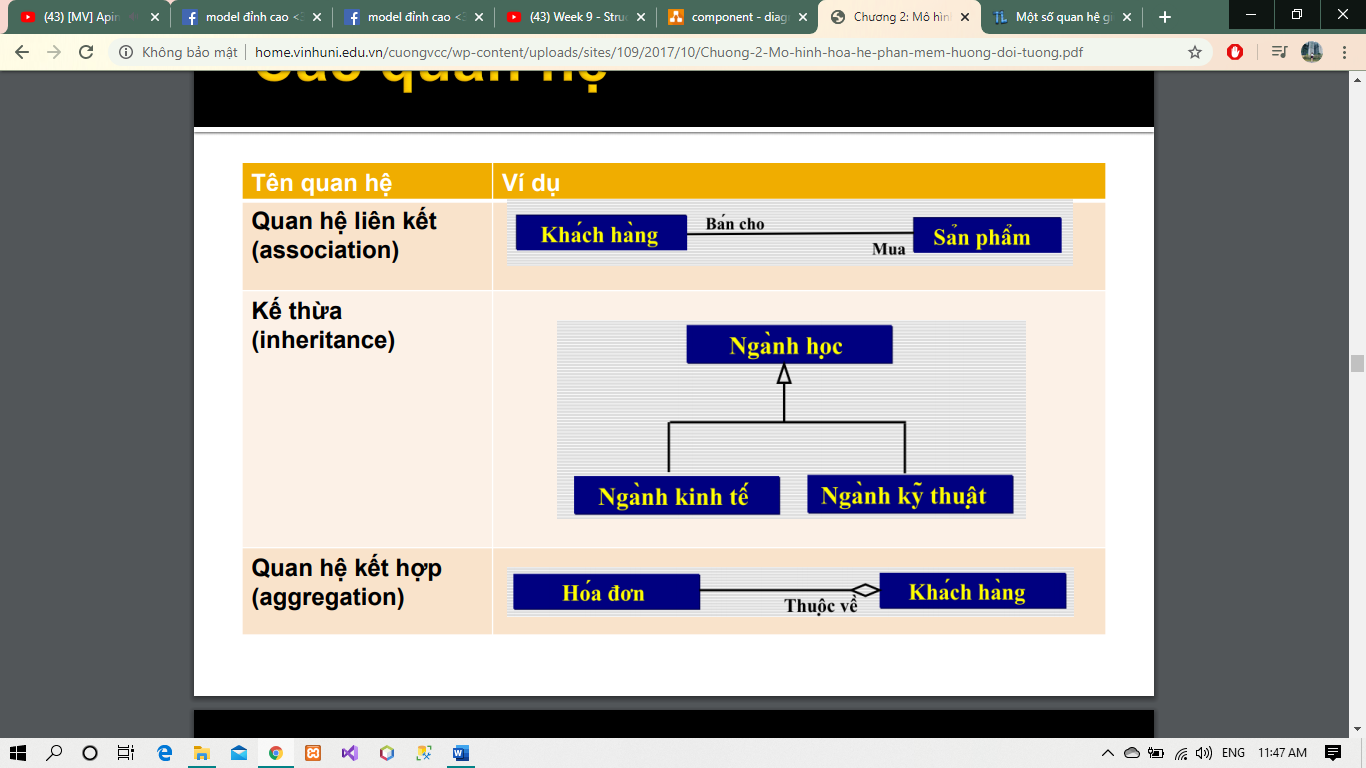


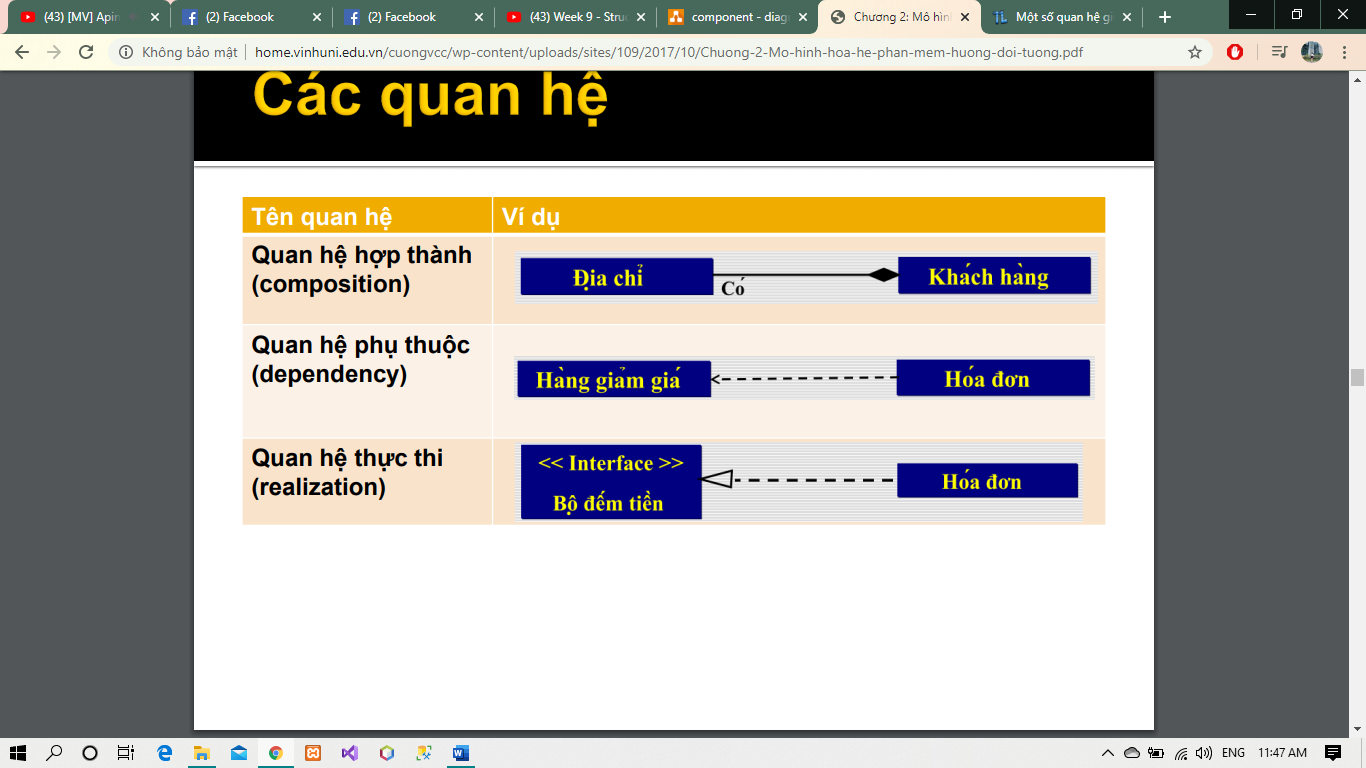


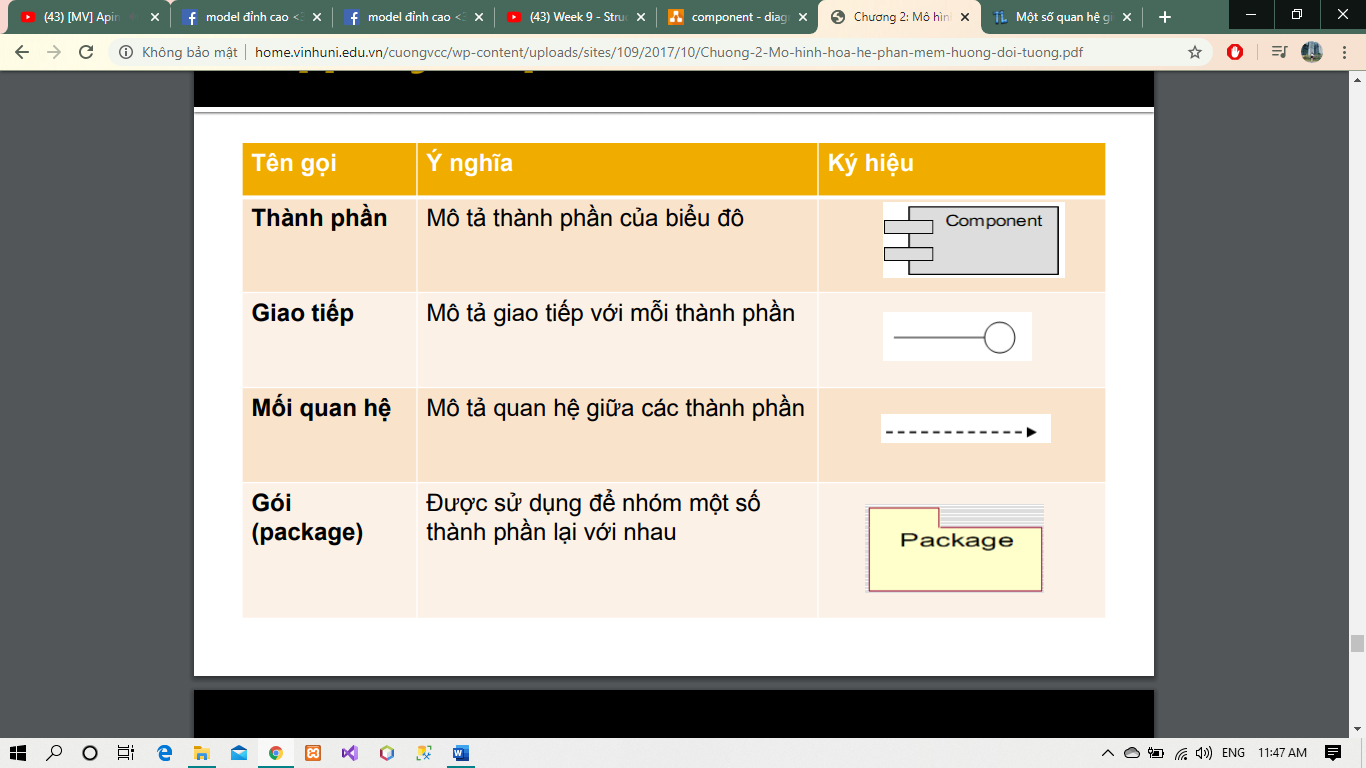












//////////////testing

Test có nhìu trường hợp để test: như login: k corect, corect, 3 lần k corect… test design

Chuển bị test data: username là abs, pass 1234 k tồn tại trong system gọi là test data trong test

Thực hiện cái việc test : test các trường hợp rồi ra test results

Vd máy tính test tác nhân lấy nhìu bộ dữ liệu