Cải thiện biểu đồ trình tự và lớp mức thiết kế

Nhóm 2023.1-143801-06
- Trần Phúc Mạnh Linh 20200352 (Đặc tả UC001)
- Nguyễn Thanh Lâm 20200336 (Đặc tả UC002)
- Bùi Trọng Đức 20200157 (Đặc tả UC003)
- Lê Đức Minh 2020395 (Đặc tả UC004)

1 Cách chỉnh sửa và ý nghĩa mang lại

1.1 Áp dụng mẫu thiết kế Singleton

Mẫu thiết kế Singleton là một trong những mẫu thiết kế phổ biến trong lập trình, được sử dụng trong những lớp **UserService, UserFactory, AttendanceFactory, HRSubsystemFactory** để đảm bảo rằng:

- Một lớp chỉ có duy nhất một thể hiện.
- Có thể dễ dàng truy cập toàn cục đối tượng.
- Không cần phải quan tâm cách khởi tạo đối tượng.

1.2 Áp dụng mẫu thiết kế Factory

Mẫu thiết kế Singleton là một trong những mẫu thiết kế phổ biến trong lập trình, được sử dụng trong những lớp **IEmployeeRepository, IDepartmentRepository, IOfficerAttendanceRepository, IUserRepository** với các lớp Factory gồm **UserFactory, AttendanceFactory, HRSubsystemFactory** được nhằm mục đích:

- **Tạo đối tượng mà không cần biết chi tiết cụ thể**: Mẫu Factory giúp ẩn đi logic cụ thể của việc tạo đối tượng. Khi sử dụng một phương thức tạo đối tượng từ Factory, bạn không cần biết chi tiết cách đối tượng đó được khởi tạo hoặc được xử lý bên trong.
- **Đảm bảo tuân thủ nguyên tắc "Open/Closed"**: Mẫu Factory giúp đảm bảo rằng khi thêm một triển khai mới của interface mà không cần phải sửa đổi mã nguồn hiện tại. Thay vào đó, chỉ cần thêm một lớp mới vào Factory.
- **Tăng sự linh hoạt của hệ thống**: Factory có thể được cấu hình để tạo ra các đối tượng thuộc các lớp con khác nhau dựa trên điều kiện nào đó. Điều này tăng sự linh hoạt và tái sử dụng mã nguồn.
- **Giảm sự phụ thuộc với đối tượng cụ thể**: Khi sử dụng một Factory để tạo đối tượng thì không cần phải biết chi tiết cụ thể cách khởi tạo của đối tượng đó. Điều này giảm sự phụ thuộc và giúp giữ cho mã nguồn linh hoạt hơn.

1.3 Áp dụng nguyên tắc Cohesion and Coupling

Các lớp được phân lại thành các gói **pages, user, report, attendance, hrsubsystem** với mục đính tăng tính kết dính cho các lớp trong một gói, và giảm sự phụ thuộc giữa các gói với nhau. Những lớp có cùng một nhiệm vụ, cùng một nghiệp vụ thì được gom nhóm lại một gói, và mỗi gói lại có một nhiệm vụ khác nhau.

Cohesion giúp tăng tính đóng gói (encapsulation) và giảm sự phụ thuộc giữa các module. Nó làm cho mã nguồn dễ đọc, dễ bảo trì, và dễ kiểm thử. Cohesion cao cũng giúp tái sử dụng mã nguồn dễ dàng hơn.

Coupling giúp tăng tính linh hoạt và tái sử dụng của hệ thống. Khi các thành phần độc lập và không phụ thuộc quá mức lớn vào nhau, sự thay đổi trong một thành phần không ảnh hưởng quá lớn đến các thành phần khác. Điều này làm cho hệ thống linh hoạt và dễ mở rộng, bảo trì.

1.4 Áp dụng nguyên lý SOLID

- *S - Một class chỉ nên giữ một trách nhiệm duy nhất*

Tất cả các lớp đều chỉ thực hiện một trách nhiệm duy nhất, như các lớp Model **OfficerAtten-dance, OfficerAndAttendance, User, Employee, Department** chỉ chứa dữ liệu về đối tượng, trong khi các lớp **EmployeeAdapter, DepartmentAdapter, SqliteOfficerAtte-ndanceRepository, SqliteUserRepository** phụ trách việc thao tác lưu trữ đối tượng trong database hoặc tương tác với hệ thống bên ngoài để lấy đối tượng, …

- *O - Có thể thoải mái mở rộng 1 module, nhưng hạn chế sửa đổi bên trong module đó*

Phần lớn các hàm đều được thiết kế theo dạng subcribe, unsubscribe để tránh việc thay đổi logic hàm, như các hàm **createRepository** và **registerRepository** trong lớp **UserFactory, AttendanceFactory**

- *L - Trong một chương trình, các object của class con có thể thay thế class cha mà không làm thay đổi tính đúng đắn của chương trình*

Không có lớp nào sử dụng kế thừa nên xem như hợp nguyên tắc.

- *I - Thay vì dùng 1 interface lớn, ta nên tách thành nhiều interface nhỏ, với nhiều mục đích cụ thể*

Subsystem được tách thành **IEmployeeRepository, IDepartmentRepository** nhằm để thực hiện truy vấn với từng kiểu dữ liệu lần lượt là **Employee, IDepartment** thay vì dồn vào 1 lớp duy nhất.

- *D - Các module cấp cao không nên phụ thuộc vào các module cấp thấp. Cả 2 nên phụ thuộc vào abstraction. Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại. (Các class giao tiếp với nhau thông qua interface, không phải thông qua implementation.)*

Thay vì phải để mã phụ thuộc vào **EmployeeAdapter, DepartmentAdapter, SqliteOfficerAtte-ndanceRepository, SqliteUserRepository** thì cho những đoạn mã phụ thuộc thay thế lần lượt vào **IEmployeeRe-pository, IDepartmentRepository, IOfficerAttendanceRepository, IUserRepository** sẽ mang lại những điều sau:

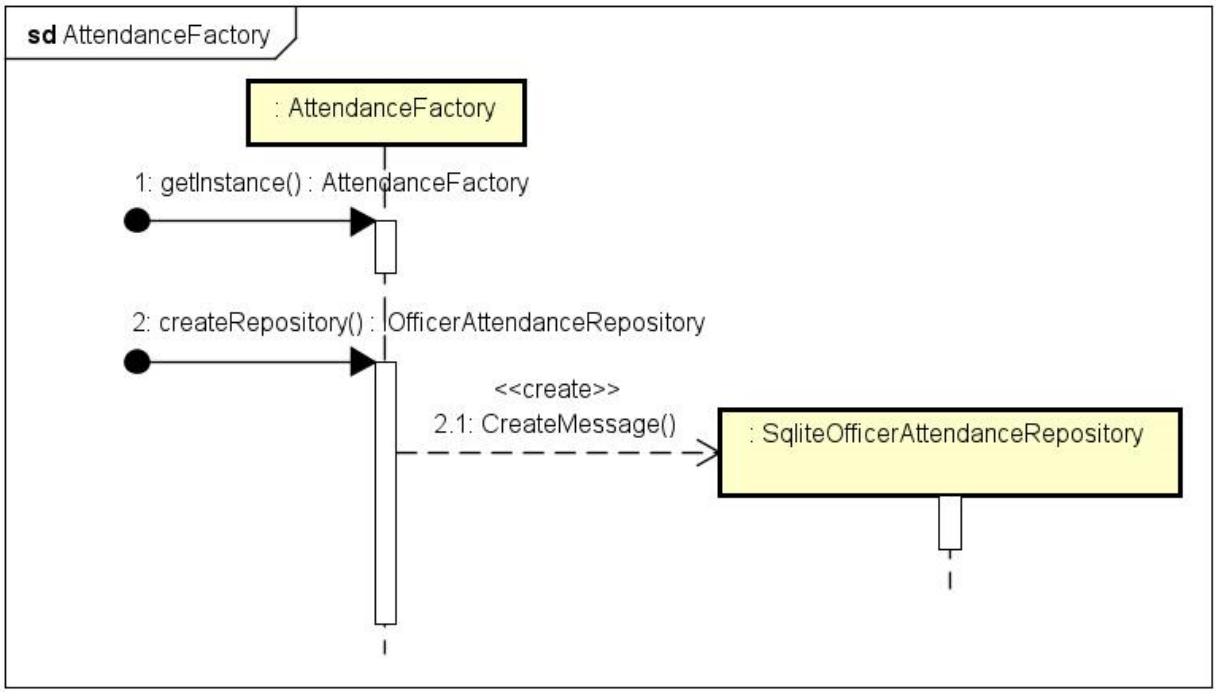
- **Định hình Cấu trúc Hệ thống**: Nguyên tắc Dependency Convention giúp định hình cấu trúc của hệ thống bằng cách quy ước về cách các thành phần phụ thuộc lẫn nhau. Điều này tạo ra một cấu trúc tự nhiên và dễ hiểu, giúp người đọc mã nguồn dễ dàng theo dõi và hiểu cấu trúc tổng thể của hệ thống.
- **Giảm Độ phức tạp**: Nguyên tắc Dependency Convention giúp giảm độ phức tạp của hệ thống bằng cách hạn chế sự phụ thuộc giữa các thành phần (thay vào đó là phụ thuộc vào interface). Điều này giúp giảm nguy cơ xung đột và khó khăn trong quá trình phát triển và bảo trì.
- **Dễ Bảo trì và Nâng cấp**: Nguyên tắc Dependency Convention giúp việc bảo trì và nâng cấp trở nên dễ dàng hơn. Các thay đổi có thể được thực hiện mà không làm ảnh hưởng đến toàn bộ hệ thống, chỉ bằng cách thay đổi implement này bằng implement khác của interface, và người phát triển có thể tập trung vào các thành phần cụ thể mà họ đang làm việc.
- **Tăng Tính tái sử dụng**: Dependency Convention thúc đẩy việc sử dụng lại mã nguồn bằng cách tạo ra các thành phần độc lập, có thể được tái sử dụng trong các phần khác nhau của hệ thống hoặc trong các dự án khác.

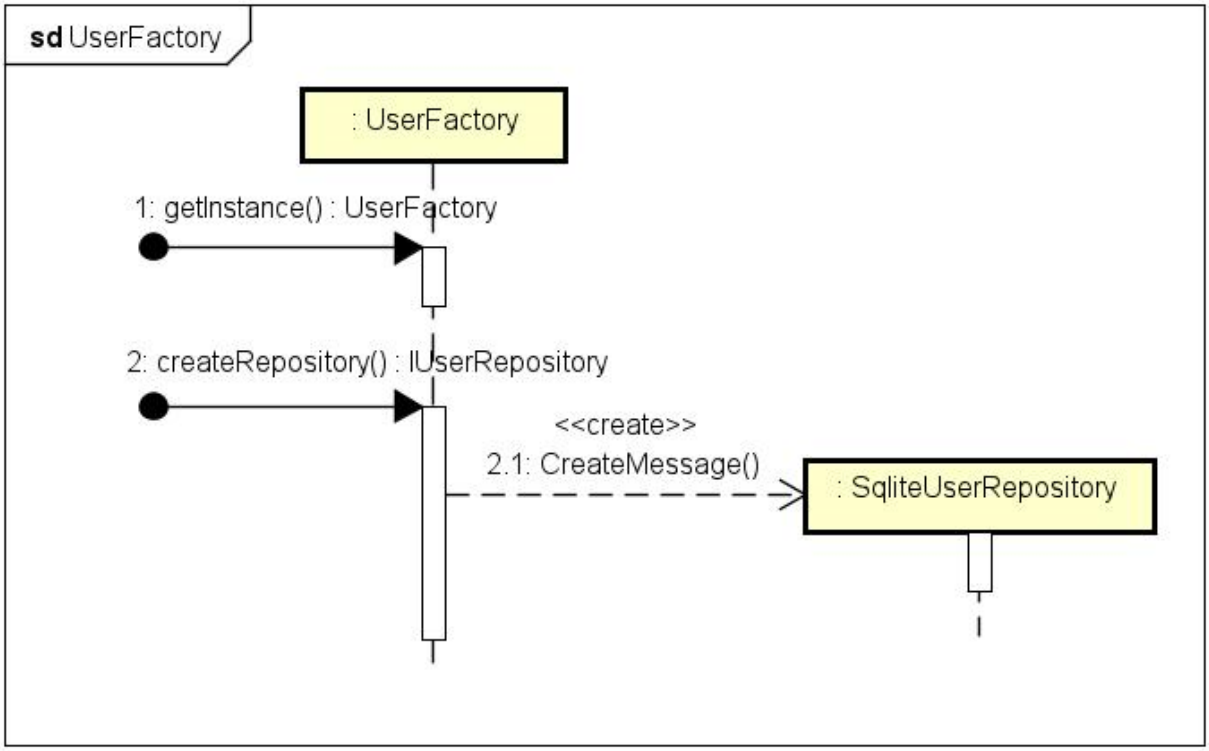
- **Tăng Tính Mô đun hóa**: Nguyên tắc này hỗ trợ tính mô đun hóa của hệ thống, trong đó mỗi mô đun (hoặc thành phần) có trách nhiệm và chức năng cụ thể. Điều này giúp tạo ra các mô đun độc lập, dễ kiểm thử và duy trì.

2 Thiết kế biểu đồ trình tự mức thiết kế

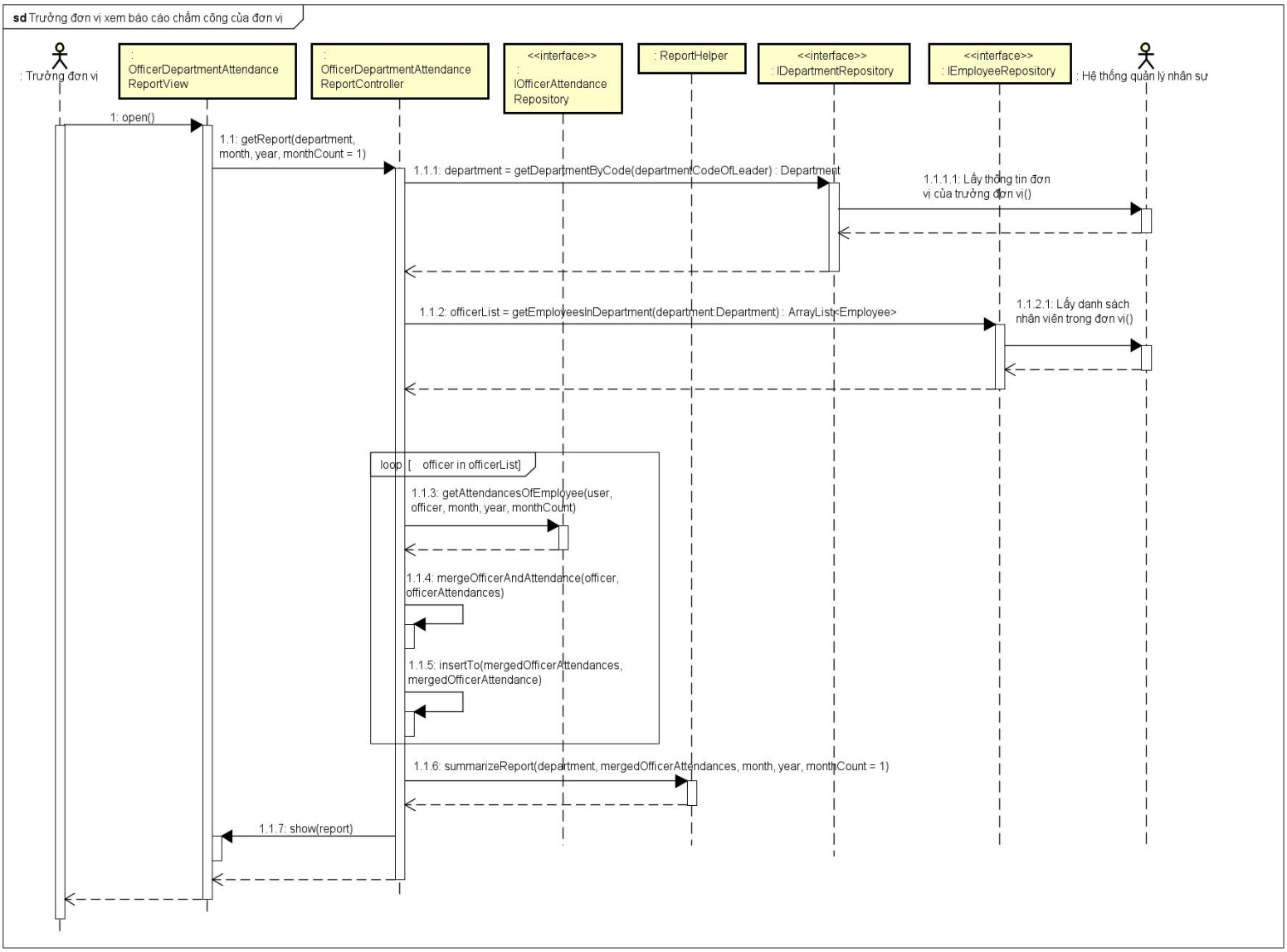
2.1 Use case "Xem báo cáo chấm công của đơn vị nhân viên văn phòng"

* Tạo ra các đối tượng IUserRepository, IOfficerAttendanceRepository

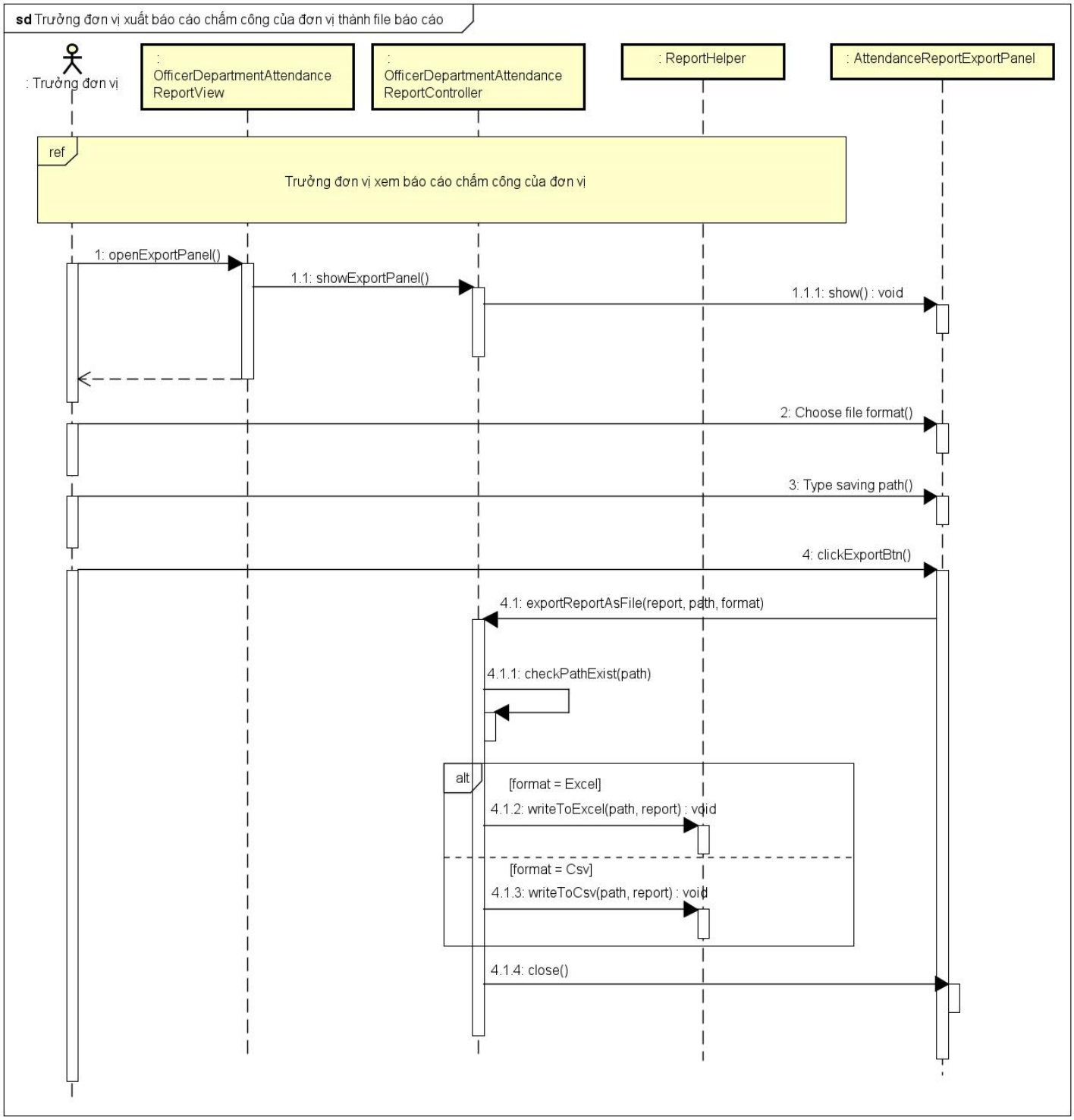




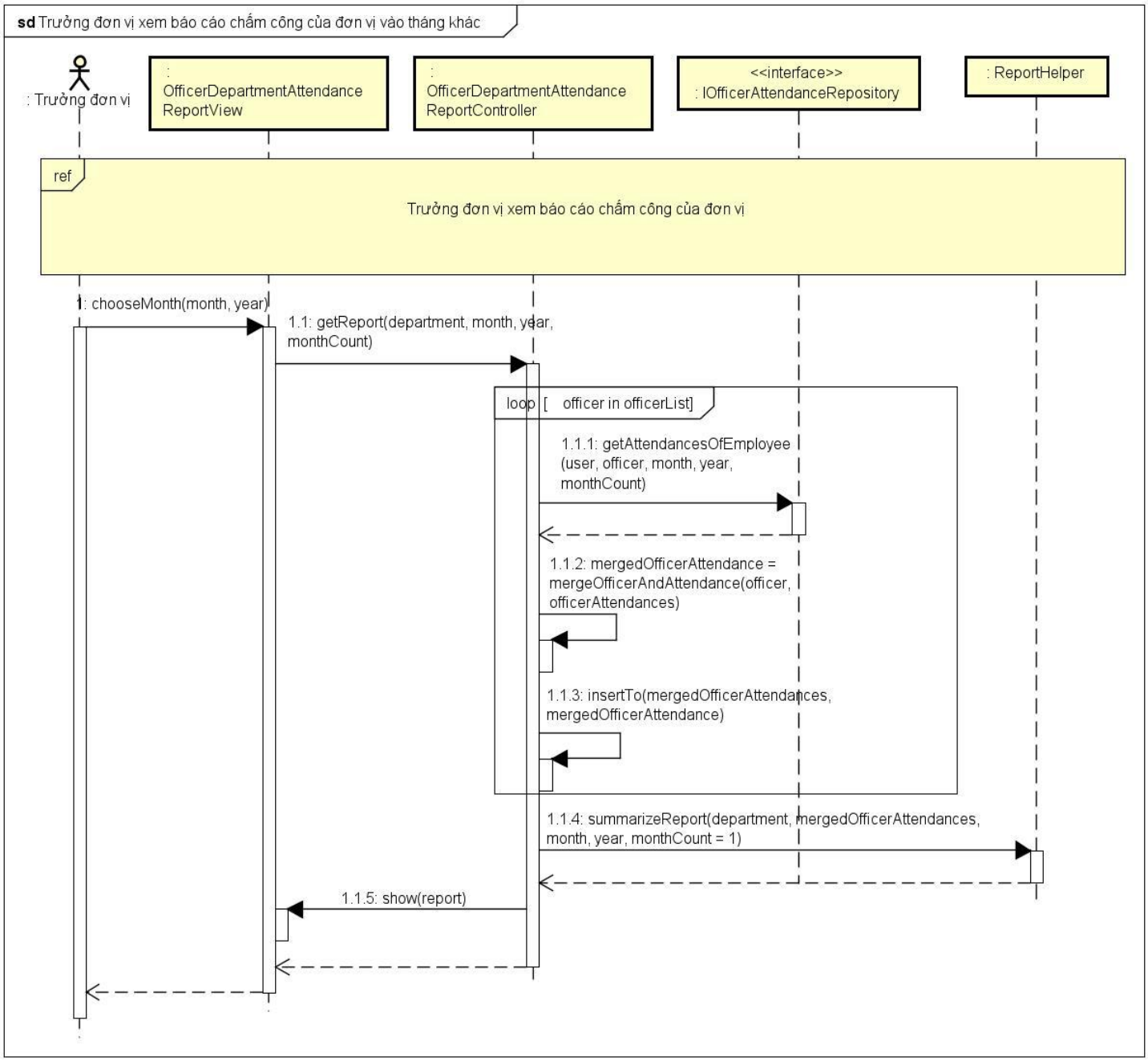
* Main Scenario: Trưởng đơn vị xem báo cáo chấm công của đơn vị



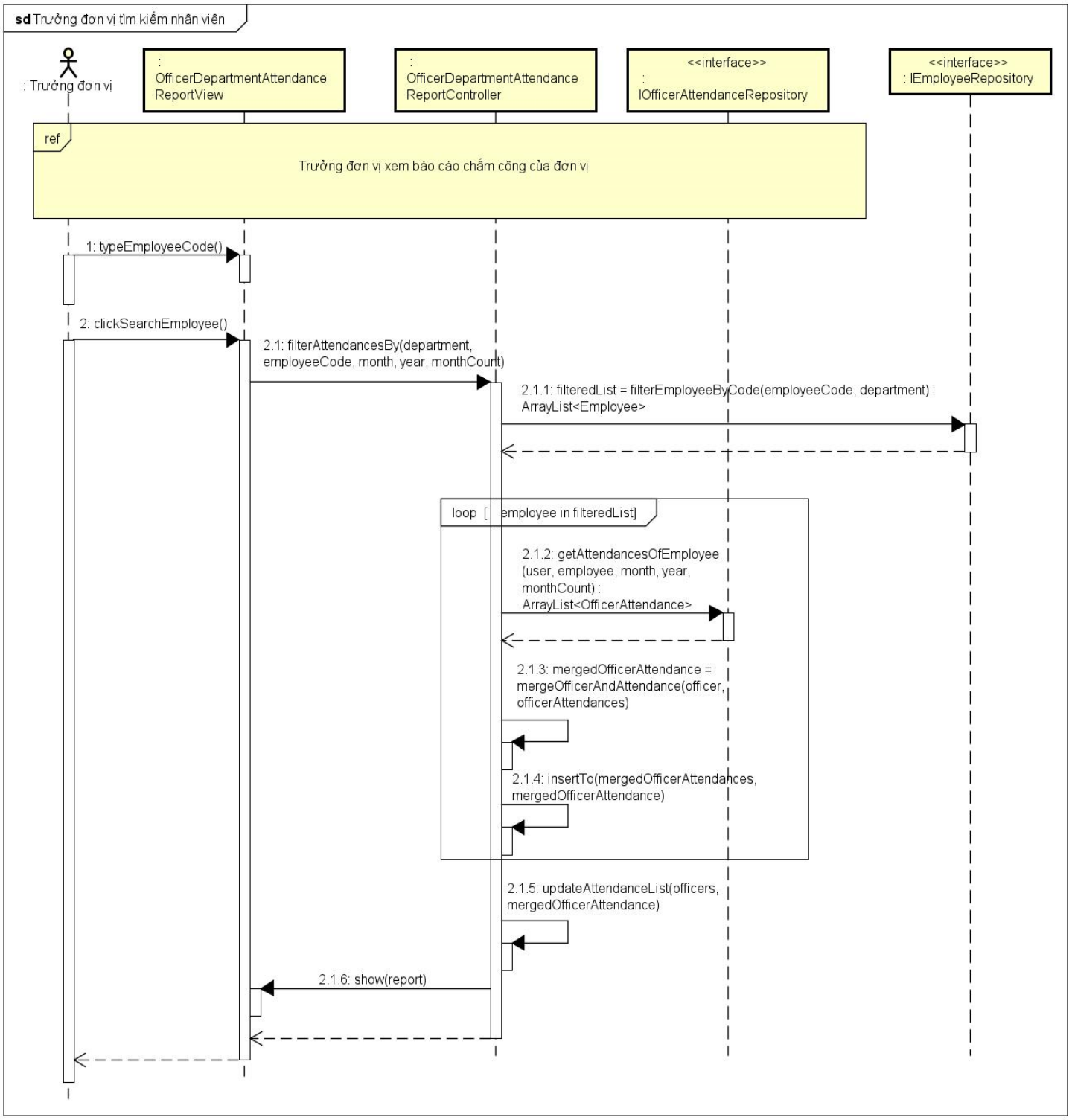
* Scenario 2: Trưởng đơn vị xuất báo cáo chấm công của đơn vị thành file báo cáo



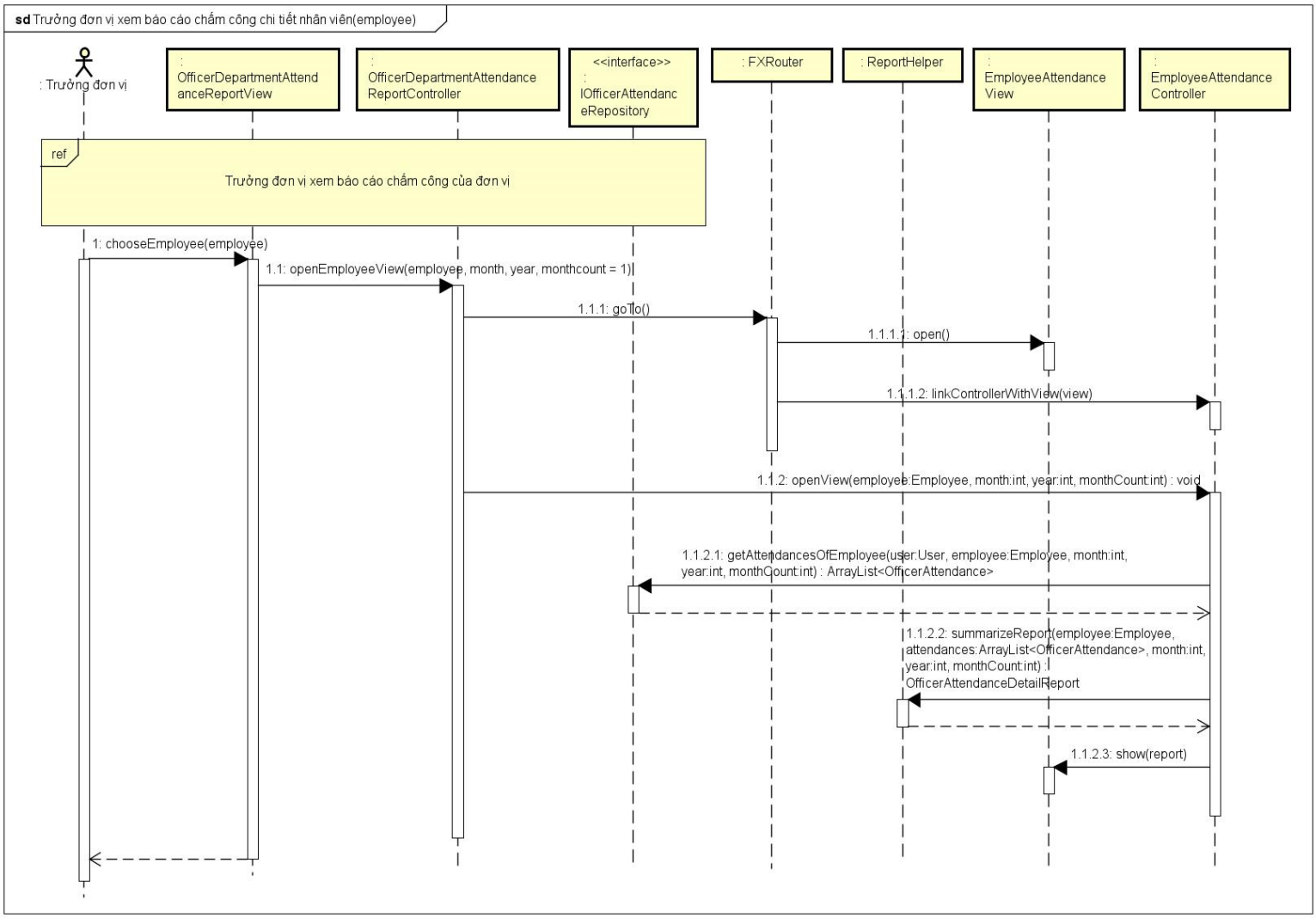
* Scenario 3: Trưởng đơn vị xem báo cáo chấm công của đơn vị vào tháng khác (tương tự với quý, năm)



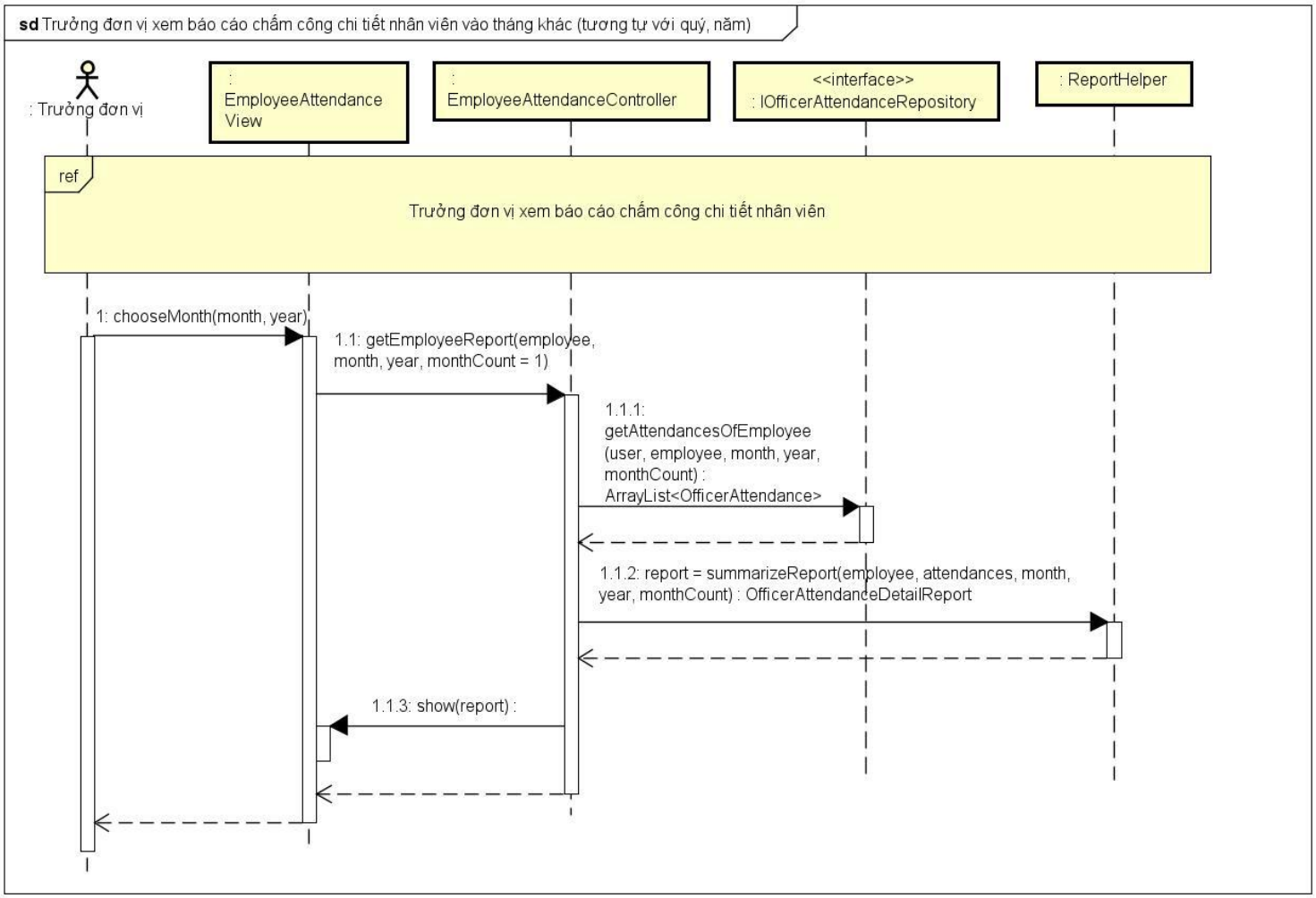
* Scenario 4: Trưởng đơn vị tìm kiếm nhân viên



* Scenario 5: Trưởng đơn vị xem báo cáo chấm công chi tiết nhân viên

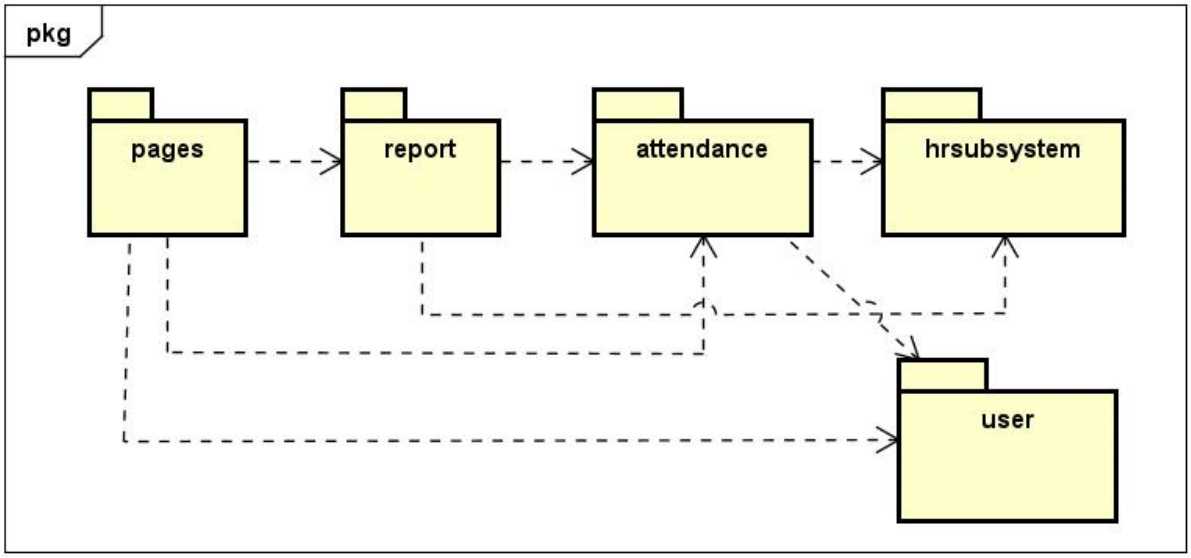


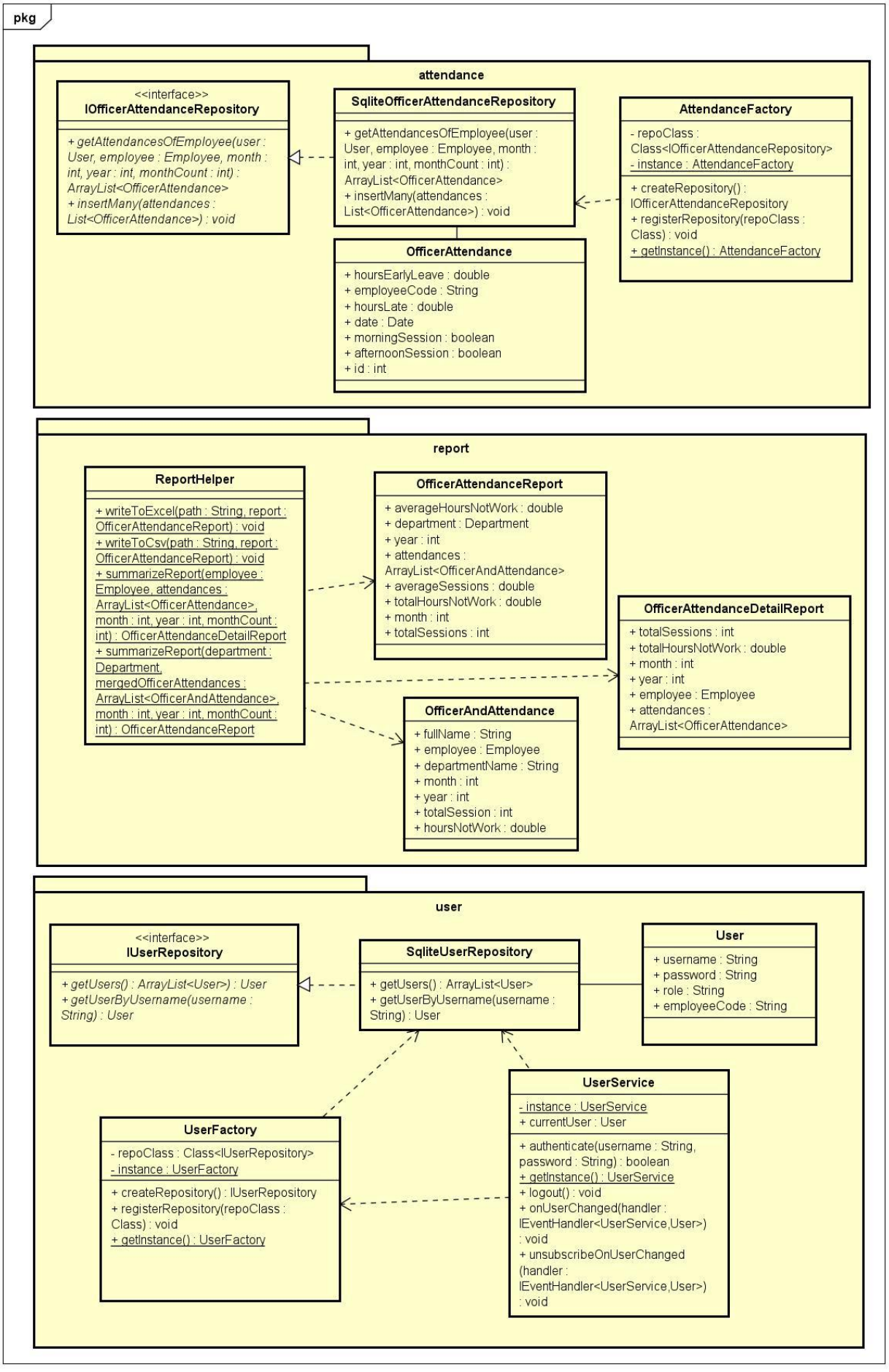
* Scenario 6: Trưởng đơn vị xem báo cáo chấm công chi tiết nhân viên vào tháng khác (tương tự với quý, năm)

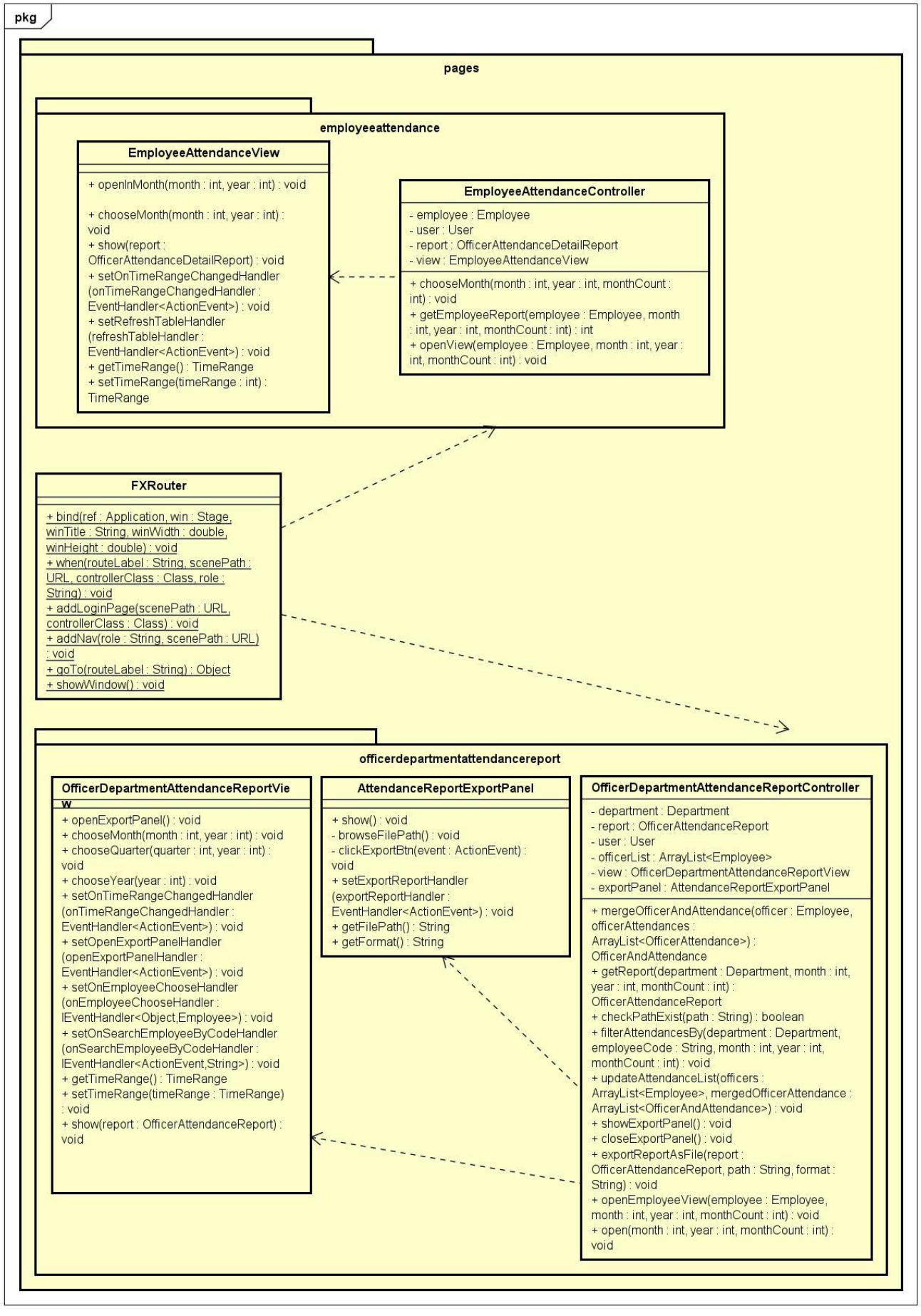


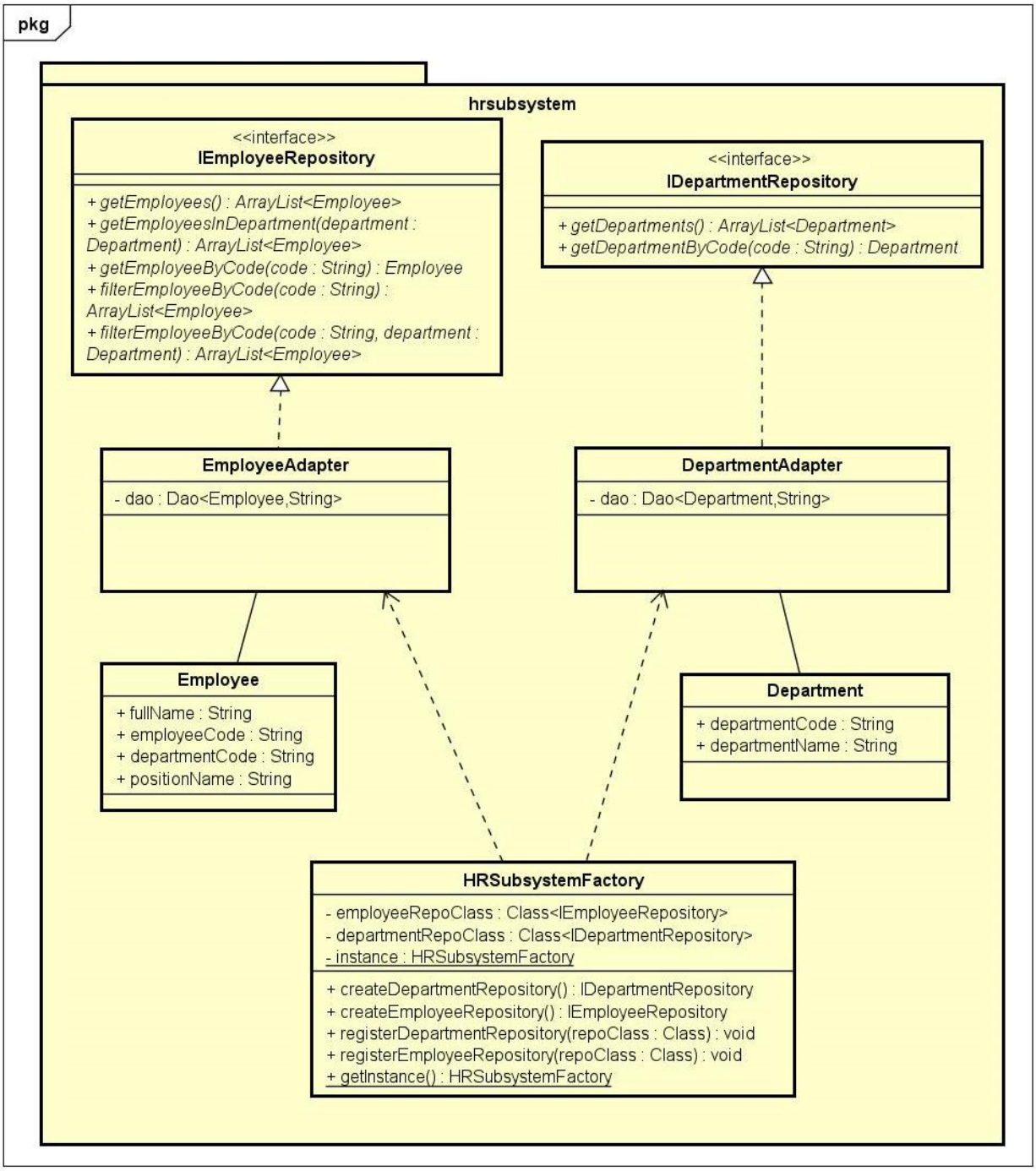
3 Biểu đồ lớp mức thiết kế

3.1 Use case “Xem báo cáo chấm công của đơn vị nhân viên văn phòng”




pkg
attendance
<<interface>>
IOfficerAttendanceRepository
+ getAttendancesOfEmployee(user : User, employee : Employee, month : int, year : int, monthCount : int) : ArrayList<OfficerAttendance>
+ insertMany(attendances : List<OfficerAttendance>) : void
SqliteOfficerAttendanceRepository
+ getAttendancesOfEmployee(user : User, employee : Employee, month : int, year : int, monthCount : int) : ArrayList<OfficerAttendance>
+ insertMany(attendances : List<OfficerAttendance>) : void
AttendanceFactory
- repoClass : Class<IOfficerAttendanceRepository>
- instance : AttendanceFactory
+ createRepository() : IOfficerAttendanceRepository
+ registerRepository(repoClass : Class) : void
+ getInstance() : AttendanceFactory
OfficerAttendance
+ hoursEarlyLeave : double
+ employeeCode : String
+ hoursLate : double
+ date : Date
+ morningSession : boolean
+ afternoonSession : boolean
+ id : int
report
ReportHelper
+ writeToExcel(path : String, report : OfficerAttendanceReport) : void
+ writeToCsv(path : String, report : OfficerAttendanceReport) : void
+ summarizeReport(employee : Employee, attendances : ArrayList<OfficerAttendance>, month : int, year : int, monthCount : int) : OfficerAttendanceDetailReport
+ summarizeReport(department : Department, mergedOfficerAttendances : ArrayList<OfficerAndAttendance>, month : int, year : int, monthCount : int) : OfficerAttendanceReport
OfficerAttendanceReport
+ averageHoursNotWork : double
+ department : Department
+ year : int
+ attendances : ArrayList<OfficerAndAttendance>
+ averageSessions : double
+ totalHoursNotWork : double
+ month : int
+ totalSessions : int
OfficerAttendanceDetailReport
+ totalSessions : int
+ totalHoursNotWork : double
+ month : int
+ year : int
+ employee : Employee
+ attendances : ArrayList<OfficerAttendance>
OfficerAndAttendance
+ fullName : String
+ employee : Employee
+ departmentName : String
+ month : int
+ year : int
+ totalSession : int
+ hoursNotWork : double
user
<<interface>>
IUserRepository
+ getUsers() : ArrayList<User>) : User
+ getUserByUsername(username : String) : User
SqliteUserRepository
+ getUsers() : ArrayList<User>
+ getUserByUsername(username : String) : User
User
+ username : String
+ password : String
+ role : String
+ employeeCode : String
UserFactory
- repoClass : Class<IUserRepository>
- instance : UserFactory
+ createRepository() : IUserRepository
+ registerRepository(repoClass : Class) : void
+ getInstance() : UserFactory
UserService
- instance : UserService
+ currentUser : User
+ authenticate(username : String, password : String) : boolean
+ getInstance() : UserService
+ logout() : void
+ onUserChanged(handler : IEventHandler<UserService,User>) : void
+ unsubscribeOnUserChanged (handler : IEventHandler<UserService,User>) : void


pkg
pages
employeeattendance
EmployeeAttendanceView
+ openInMonth(month : int, year : int) : void
+ chooseMonth(month : int, year : int) : void
+ show(report : OfficerAttendanceDetailReport) : void
+ setOnTimeRangeChangedHandler (onTimeRangeChangedHandler : EventHandler<ActionEvent>) : void
+ setRefreshTableHandler (refreshTableHandler : EventHandler<ActionEvent>) : void
+ getTimeRange() : TimeRange
+ setTimeRange(timeRange : int) : TimeRange
EmployeeAttendanceController
- employee : Employee
- user : User
- report : OfficerAttendanceDetailReport
- view : EmployeeAttendanceView
+ chooseMonth(month : int, year : int, monthCount : int) : void
+ getEmployeeReport(employee : Employee, month : int, year : int, monthCount : int) : int
+ openView(employee : Employee, month : int, year : int, monthCount : int) : void
FXRouter
+ bind(ref : Application, win : Stage, winTitle : String, winWidth : double, winHeight : double) : void
+ when(routeLabel : String, scenePath : URL, controllerClass : Class, role : String) : void
+ addLoginPage(scenePath : URL, controllerClass : Class) : void
+ addNav(role : String, scenePath : URL) : void
+ goTo(routeLabel : String) : Object
+ showWindow() : void
officerdepartmentattendancereport
OfficerDepartmentAttendanceReportView
+ openExportPanel() : void
+ chooseMonth(month : int, year : int) : void
+ chooseQuarter(quarter : int, year : int) : void
+ chooseYear(year : int) : void
+ setOnTimeRangeChangedHandler (onTimeRangeChangedHandler : EventHandler<ActionEvent>) : void
+ setOpenExportPanelHandler (openExportPanelHandler : EventHandler<ActionEvent>) : void
+ setOnEmployeeChooseHandler (onEmployeeChooseHandler : IEventHandler<Object,Employee>) : void
+ setOnSearchEmployeeByCodeHandler (onSearchEmployeeByCodeHandler : IEventHandler<ActionEvent,String>) : void
+ getTimeRange() : TimeRange
+ setTimeRange(timeRange : TimeRange) : void
+ show(report : OfficerAttendanceReport) : void
AttendanceReportExportPanel
+ show() : void
- browseFilePath() : void
- clickExportBtn(event : ActionEvent) : void
+ setExportReportHandler (exportReportHandler : EventHandler<ActionEvent>) : void
+ getFilePath() : String
+ getFormat() : String
OfficerDepartmentAttendanceReportController
- department : Department
- report : OfficerAttendanceReport
- user : User
- officerList : ArrayList<Employee>
- view : OfficerDepartmentAttendanceReportView
- exportPanel : AttendanceReportExportPanel
+ mergeOfficerAndAttendance(officer : Employee, officerAttendances : ArrayList<OfficerAttendance>) : OfficerAndAttendance
+ getReport(department : Department, month : int, year : int, monthCount : int) : OfficerAttendanceReport
+ checkPathExist(path : String) : boolean
+ filterAttendancesBy(department : Department, employeeCode : String, month : int, year : int, monthCount : int) : void
+ updateAttendanceList(officers : ArrayList<Employee>, mergedOfficerAttendance : ArrayList<OfficerAndAttendance>) : void
+ showExportPanel() : void
+ closeExportPanel() : void
+ exportReportAsFile(report : OfficerAttendanceReport, path : String, format : String) : void
+ openEmployeeView(employee : Employee, month : int, year : int, monthCount : int) : void
+ open(month : int, year : int, monthCount : int) : void


pkg
hrsubsystem
<<interface>>
IEmployeeRepository
+ getEmployees() : ArrayList<Employee>
+ getEmployeesInDepartment(department : Department) : ArrayList<Employee>
+ getEmployeeByCode(code : String) : Employee
+ filterEmployeeByCode(code : String) : ArrayList<Employee>
+ filterEmployeeByCode(code : String, department : Department) : ArrayList<Employee>
<<interface>>
IDepartmentRepository
+ getDepartments() : ArrayList<Department>
+ getDepartmentByCode(code : String) : Department
EmployeeAdapter
- dao : Dao<Employee,String>
DepartmentAdapter
- dao : Dao<Department,String>
Employee
+ fullName : String
+ employeeCode : String
+ departmentCode : String
+ positionName : String
Department
+ departmentCode : String
+ departmentName : String
HRSubsystemFactory
- employeeRepoClass : Class<IEmployeeRepository>
- departmentRepoClass : Class<IDepartmentRepository>
- instance : HRSubsystemFactory
+ createDepartmentRepository() : IDepartmentRepository
+ createEmployeeRepository() : IEmployeeRepository
+ registerDepartmentRepository(repoClass : Class) : void
+ registerEmployeeRepository(repoClass : Class) : void
+ getInstance() : HRSubsystemFactory