Software Engineering Theory and Practice

School of Computing	UNIVERSITY OF PORTSMOUTH
Title	Software Engineering Theory and Practice
Module Coordinator	Steven Ossont
Email	steven.ossont@port.ac.uk
Code	M30819
Moodle	https://moodle.port.ac.uk/course/view.php?id=11429

U30819: Software Engineering Theory and Practice

System Modeling / UML (Chapter 5)

https://moodle.port.ac.uk/course/view.php?id=11429

System modeling

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML) v2.
- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

Models of existing systems

Models of the existing system are:

- used during requirements engineering
- clarify what the existing system does
- a basis for discussing its strengths and weaknesses
- Then lead to requirements for the new system

Models of planned systems

Models of the new system are:

- used during requirements engineering
- used to explain the proposed requirements to stakeholders
- used to discuss design proposals
- used to document the system for implementation

> In a model-driven engineering process, it is possible to generate a complete or partial system implementation from the system model

System perspectives

An external perspective

- where you model the context or environment of the system

An interaction perspective,

- where you model the interactions between a system and its environment, or between the components of a system

A structural perspective

- where you model the organization of a system or the structure of the data that is processed by the system

A behavioral perspective

- where you model the dynamic behavior of the system and how it responds to events

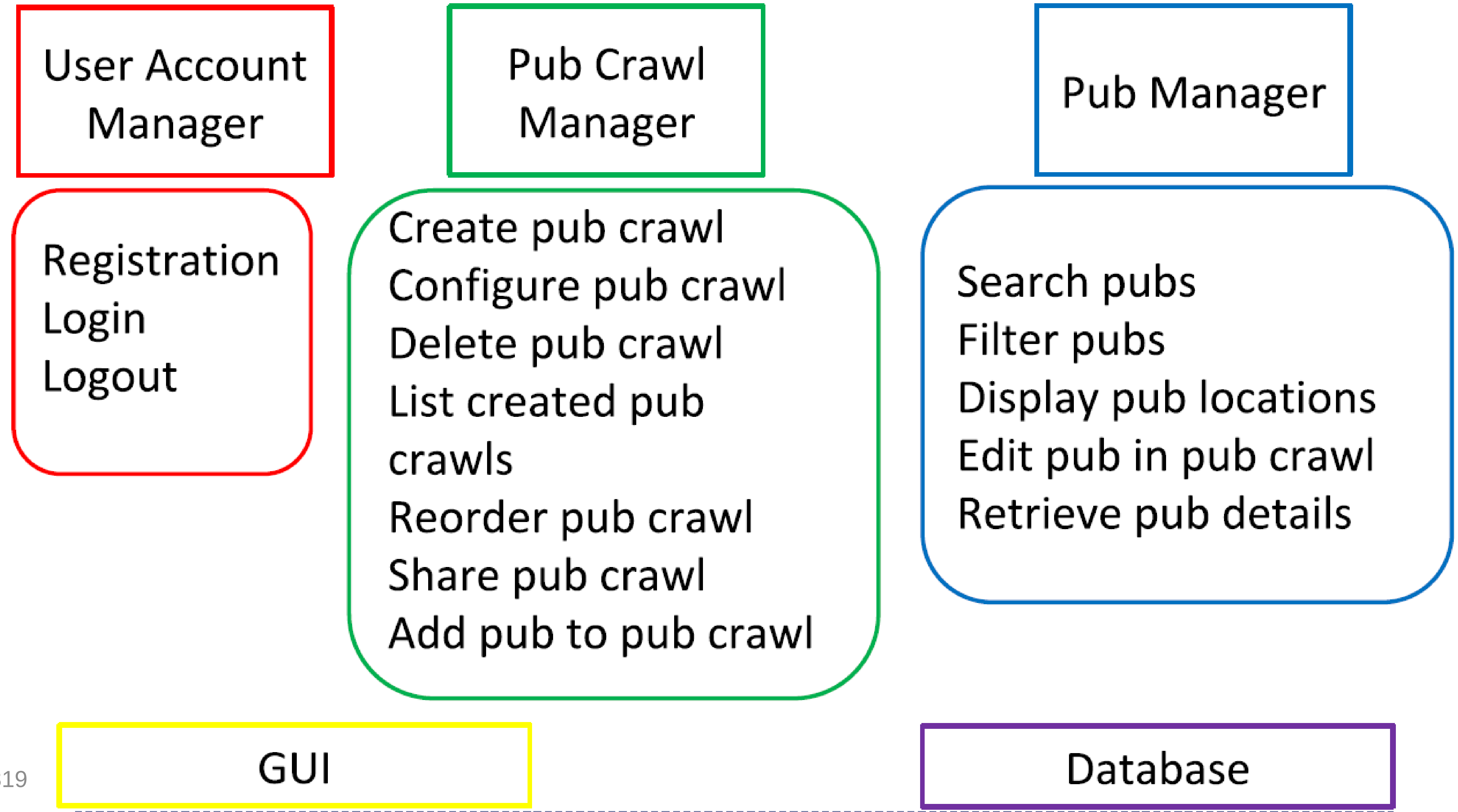
Pub Crawler Case Study

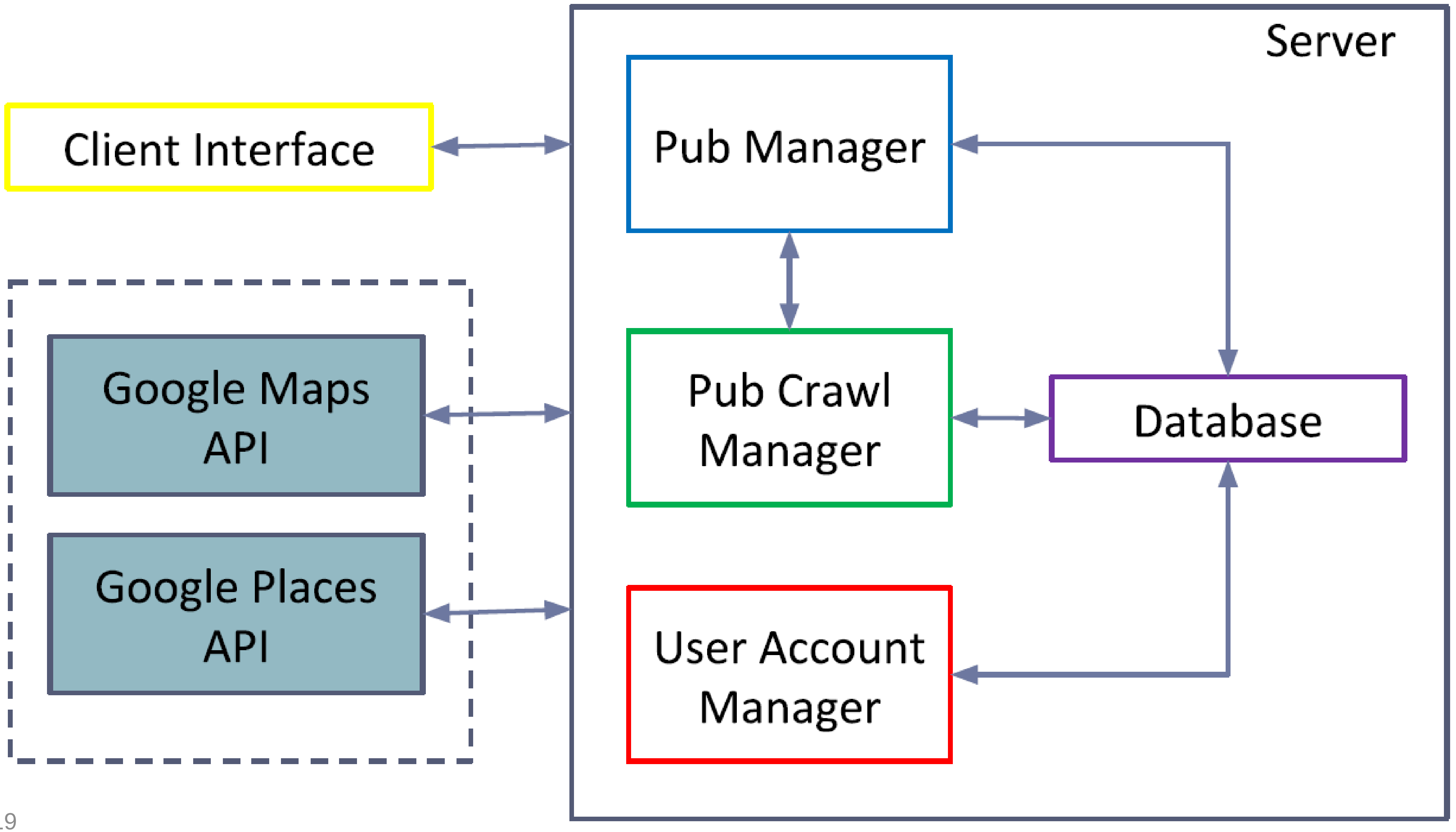
- Registration
- Login
- Create pub crawl
- Configure pub crawl parameters
- Delete pub crawl
- List created pub crawls
- Reorder pub crawls
- Share pub crawl
- Search, filter, display pubs

Pub Crawler Case Study

- Add pub to pub crawl
- Display current location and pub locations
- Edit pub in pub crawl
- Retrieve pub details
- Logout
- Access help page

Case study


User Account Manager
Registration
Login
Logout
Pub Crawl Manager
Create pub crawl
Configure pub crawl
Delete pub crawl
List created pub crawls
Reorder pub crawl
Share pub crawl
Add pub to pub crawl
Pub Manager
Search pubs
Filter pubs
Display pub locations
Edit pub in pub crawl
Retrieve pub details
GUI
Database


Server
Client Interface
Pub Manager
Google Maps API
Pub Crawl Manager
Database
Google Places API
User Account Manager

UML 2

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language it is intended to provide a standard way to visualize the design of a system

- Behavior Diagrams (Including interactions)
- Structure diagrams

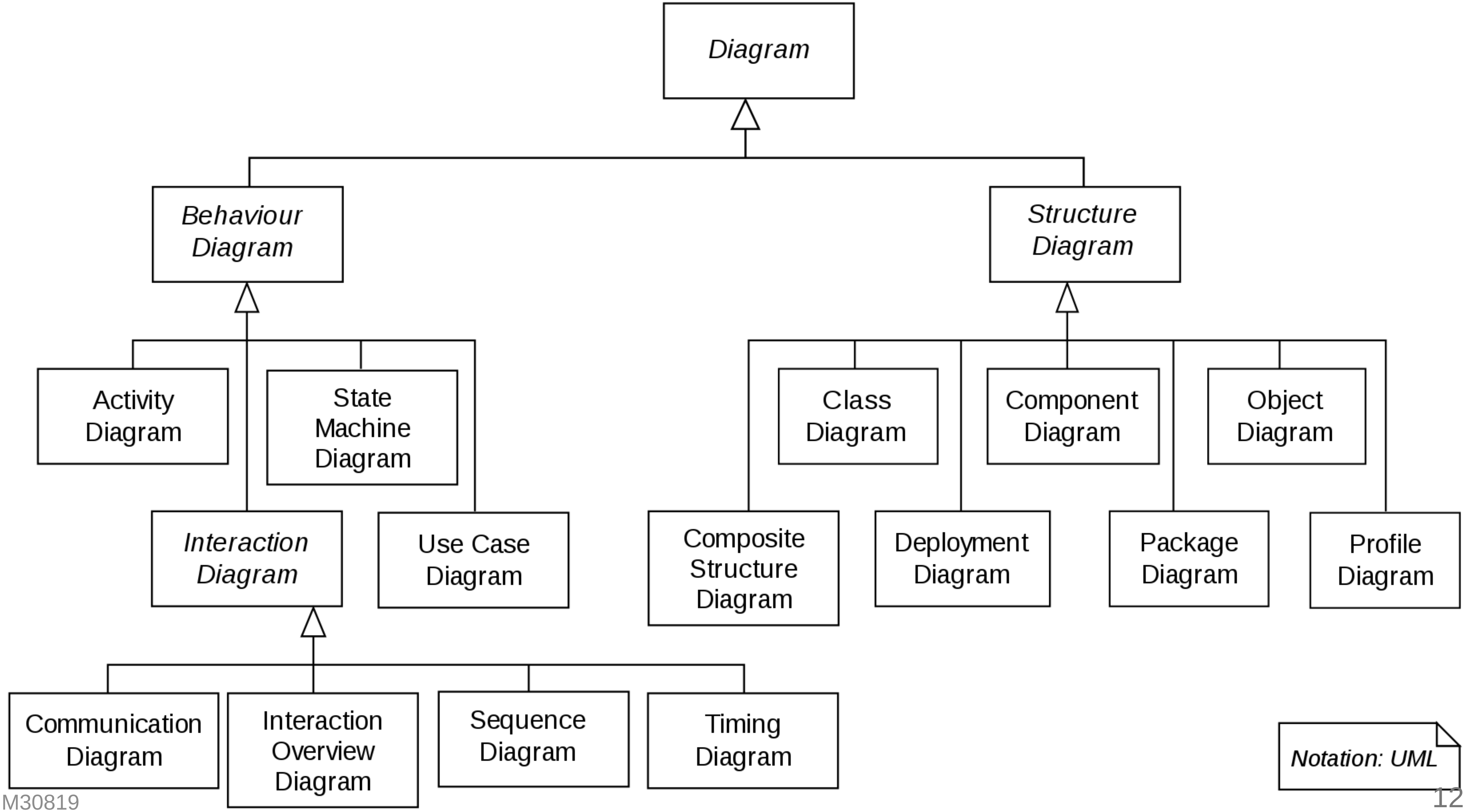


Diagram
Behaviour Diagram
Structure Diagram
Activity Diagram
State Machine Diagram
Interaction Diagram
Use Case Diagram
Communication Diagram
Interaction Overview Diagram
Sequence Diagram
Timing Diagram
Class Diagram
Component Diagram
Object Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Profile Diagram
Notation: UML

UML diagrams

A handy tool to draw UML in VSCode

- https://github.com/jkeys089/vscode-plantuml

UML - Class Diagram

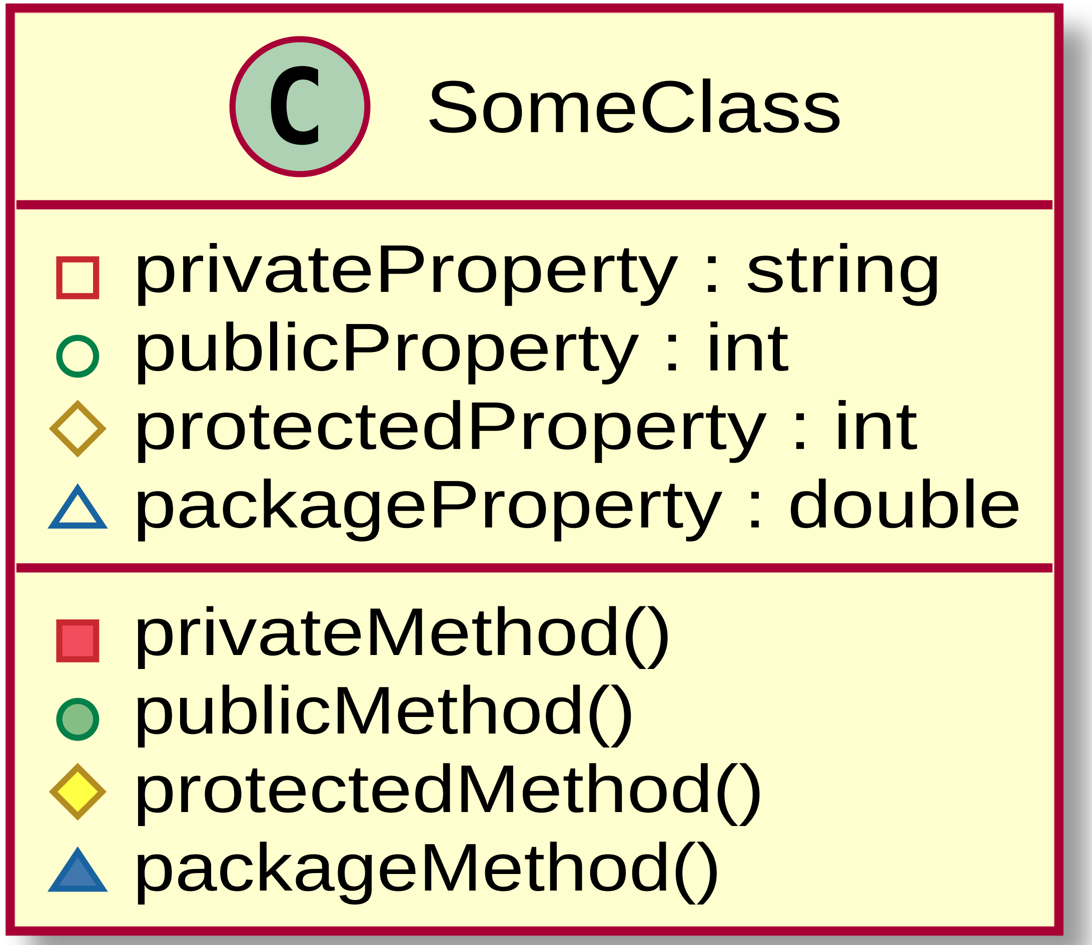
Class Diagrams show the object classes in the system and the associations between these classes.

https://www.youtube.com/watch?v=UI6lqHOVHic

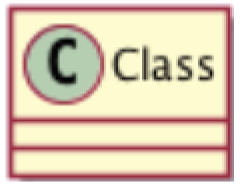
Example class

```
@startuml exampleClass
class SomeClass {
 -privateProperty : string
 +publicProperty : int
 #protectedProperty : int
 ~packageProperty : double
 -privateMethod()
 +publicMethod()
 #protectedMethod()
 ~packageMethod()
}
@enduml
```

- Order by visibility

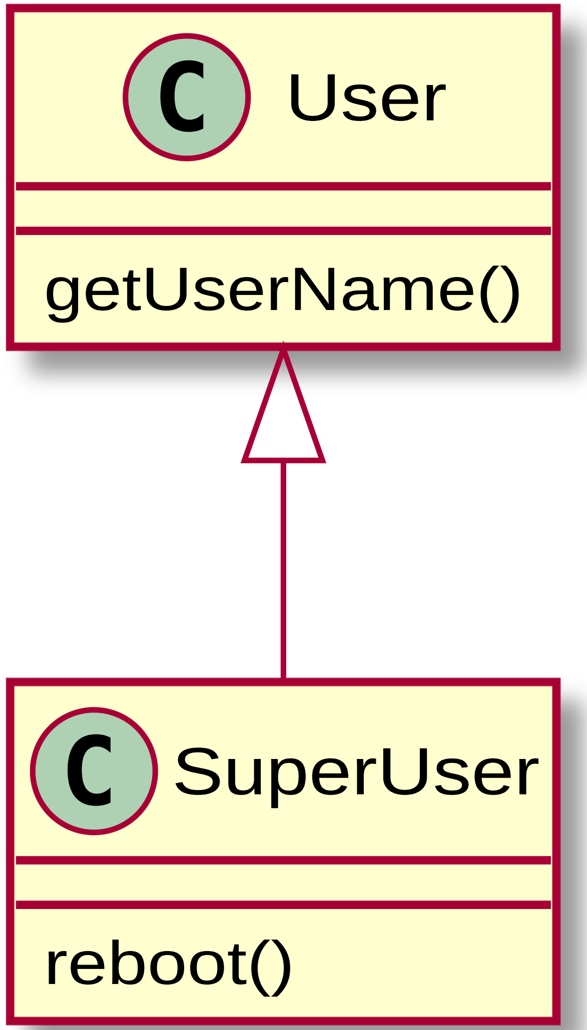


PlantUML Class syntax

keyword	usage	image
`class`	Class	
`+`	Public	○
`-`	Private	□
`#`	Protected	◇
`~`	Package	△

Class Inheritance

```
@startuml Inheritance
User <|-- SuperUser

User : getUserName()

SuperUser : reboot()

@enduml
```

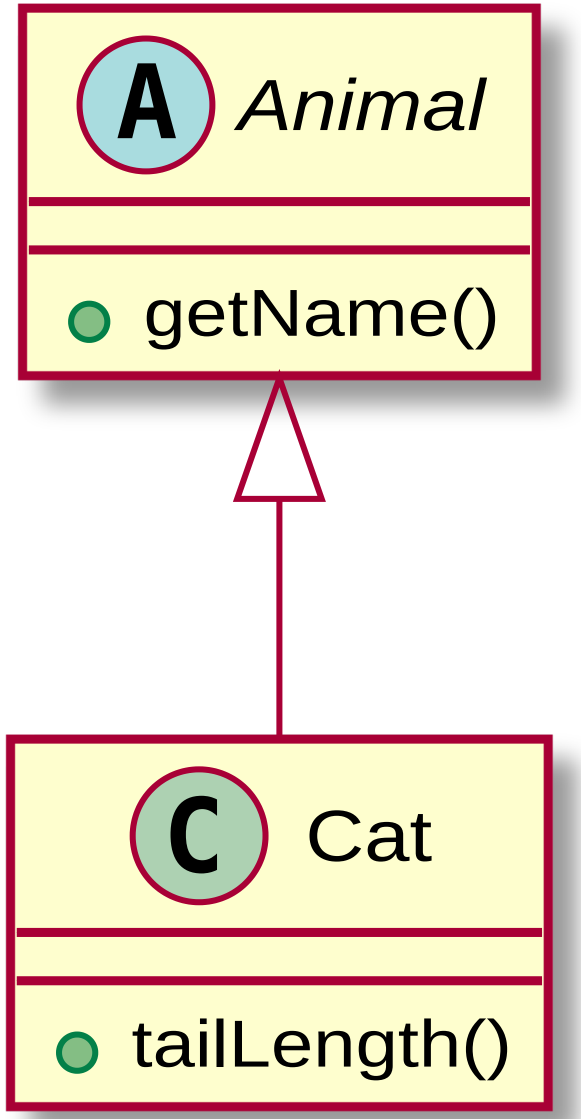


Relations

```
@startuml AbstractClass
abstract class Animal {
    + getName()
}
class Cat {
    + tailLength()
}
Animal <|-- Cat
@enduml
```

◇— Aggregation: Collections (can exist inside an an aggregation or outside)

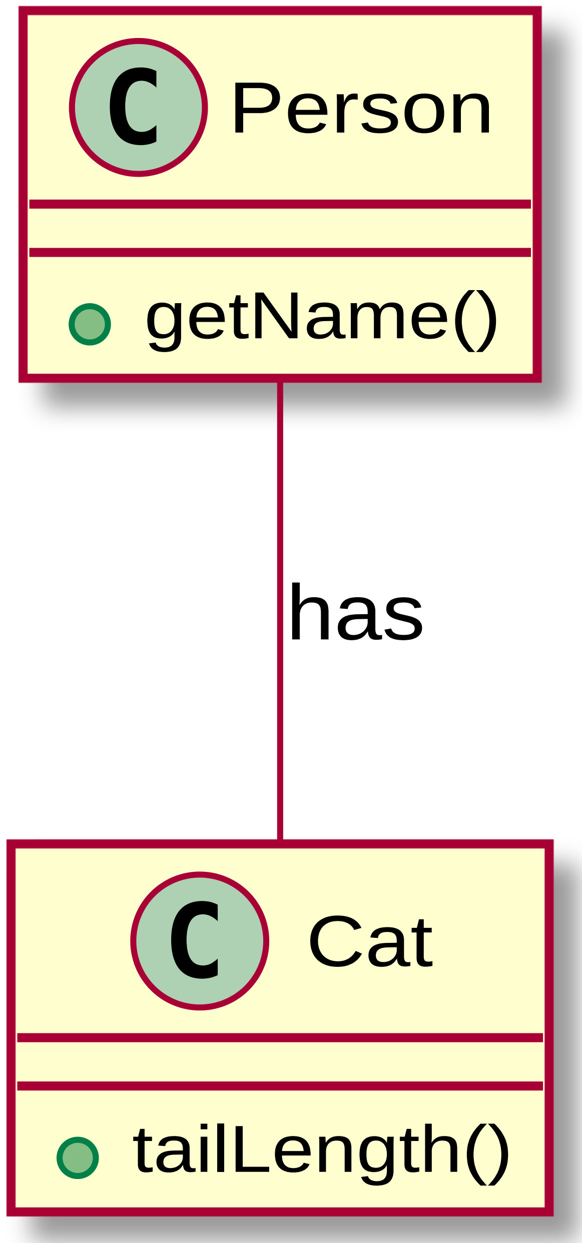
◆— Composition: Part of and cannot exist outside the collection



Class Association

```
@startuml AssociationClass
class Person {
    + getName()
}
class Cat {
    + tailLength()
}
Person -- Cat : has
@enduml
```

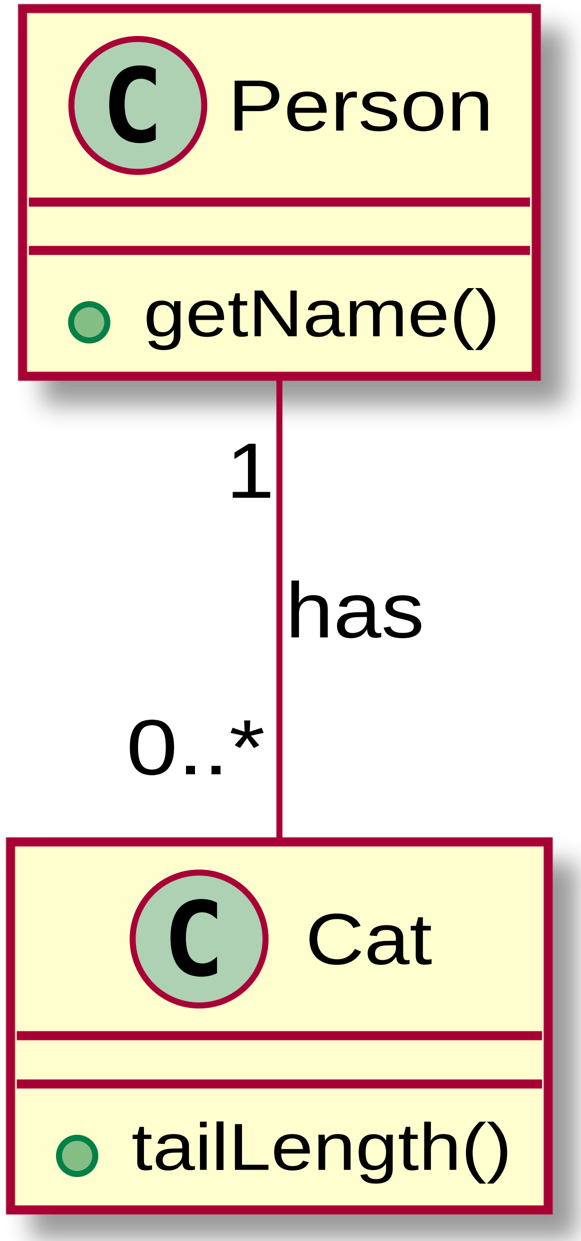
> Description is optional



Class Multiplicity

```
@startuml AssociationClass2
class Person {
    + getName()
}
class Cat {
    + tailLength()
}
Person "1" -- "0..*"Cat  : has
@enduml
```

- `0..1` zero to one
- `n` specific number
- `0..*` zero to many
- `1..*` one to many
- `m..n` number range



UML - Use case

https://plantuml.com/use-case-diagram A high level overview of how to interact with software

- the interactions between a system and its environment.
- What does the user expect from the system
- Excellent as describing systems to third parties
- You need **both** a diagram and text description (table)

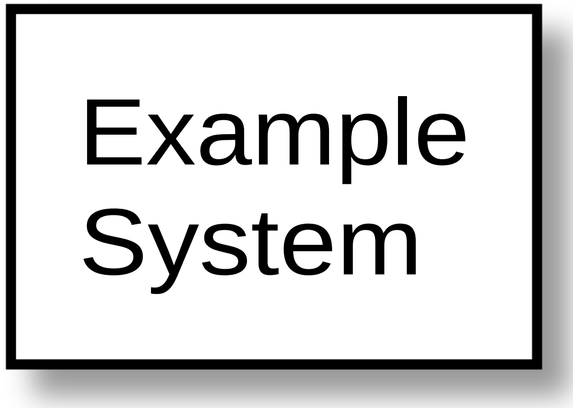
Reading

- Recap: https://www.youtube.com/watch?v=zid-MVo7M-E
- Drawing: http://ogom.github.io/draw_uml/plantuml/
- Reference: https://www.uml-diagrams.org/use-case-reference.html

Systems

```
@startuml SystemExample
rectangle Example\nSystem as S1 {
}
@enduml
```

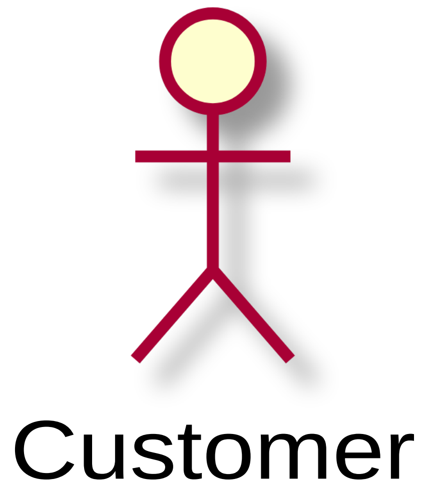
- The system or application that you are developing
- Rectangle with name at the top



Actors

```
@startuml ActorExample
:Customer: as A1
@enduml
```

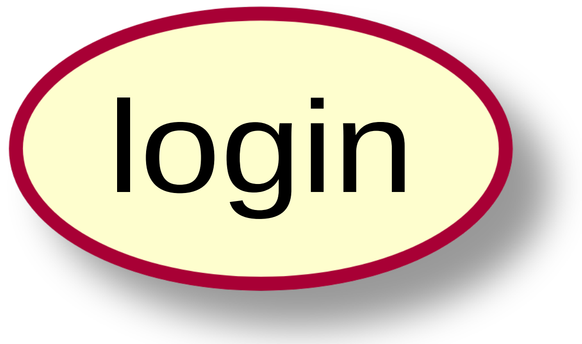
- Someone or something that uses the system
    - External objects to the system
- The people, systems, device or other stakeholders
- Primary actor (Left of System) - initiates action
- Secondary actor (Right of System) - reacts to the system



Use Cases

```
@startuml UseCaseOnly
(login) as (Ex1)
@enduml
```

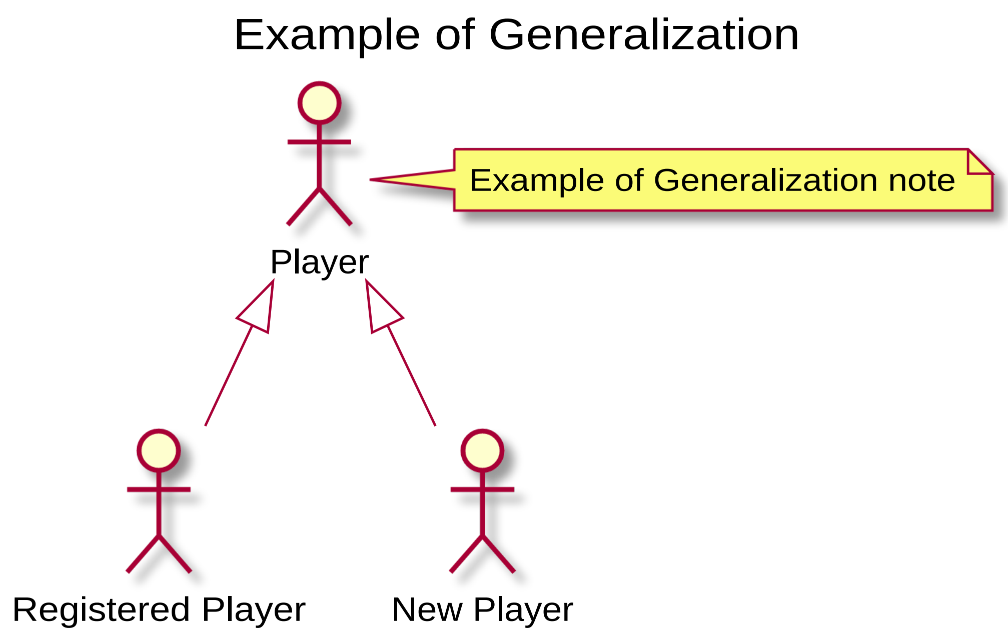
- Represents a task within the system
- Consider a name with a verb
- Sort them if possible (Order of usage)
- Every actor must interact with a usecase



Relationships -- Generalization

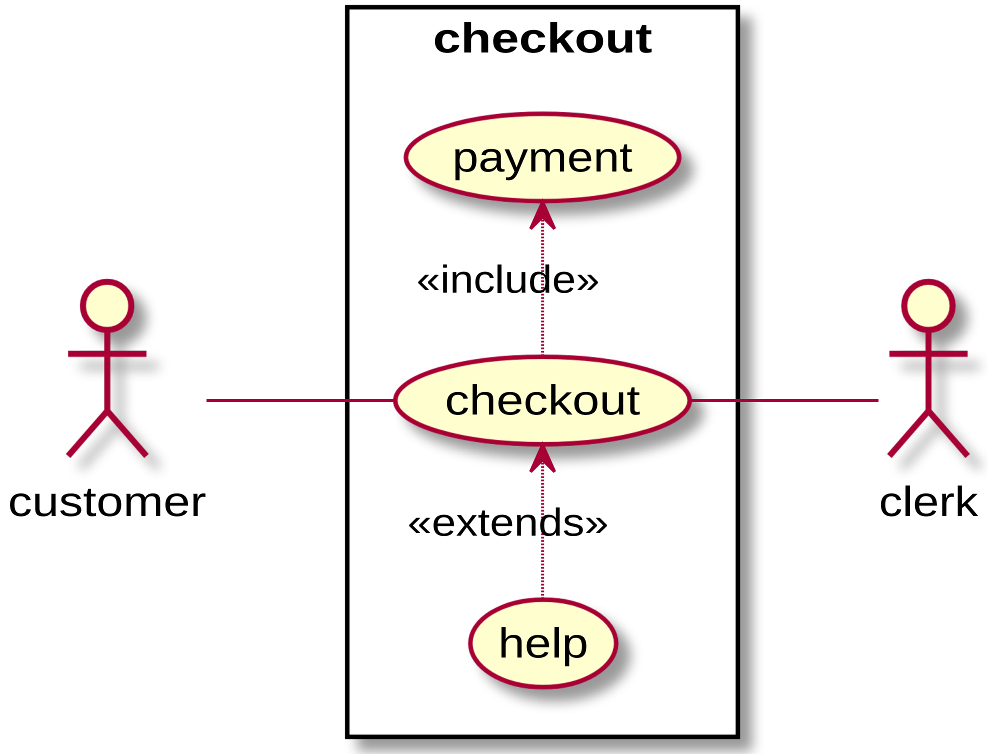
```
@startuml Generalization
title Example of Generalization
:Player: as A1
note right: Example of Generalization note
:Registered Player: as A2
:New Player: as A3
A1 <|--  A2
A1 <|--  A3
@enduml
```

- Association / Use (Solid line)
- Generalization (dash , triangle)



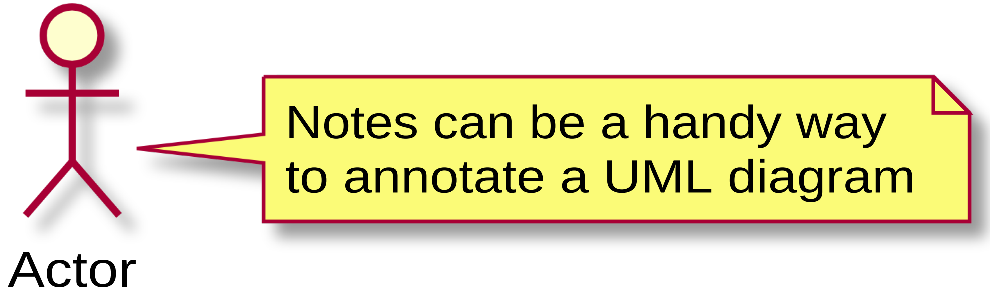
Relationships: Include & extend

- Includes (dash, arrow)
  - Base usecase **will** execute included Usecase - ->
- Extends (dash, arrow)
  - Base usecase **may** execute Extended Usecase - ->



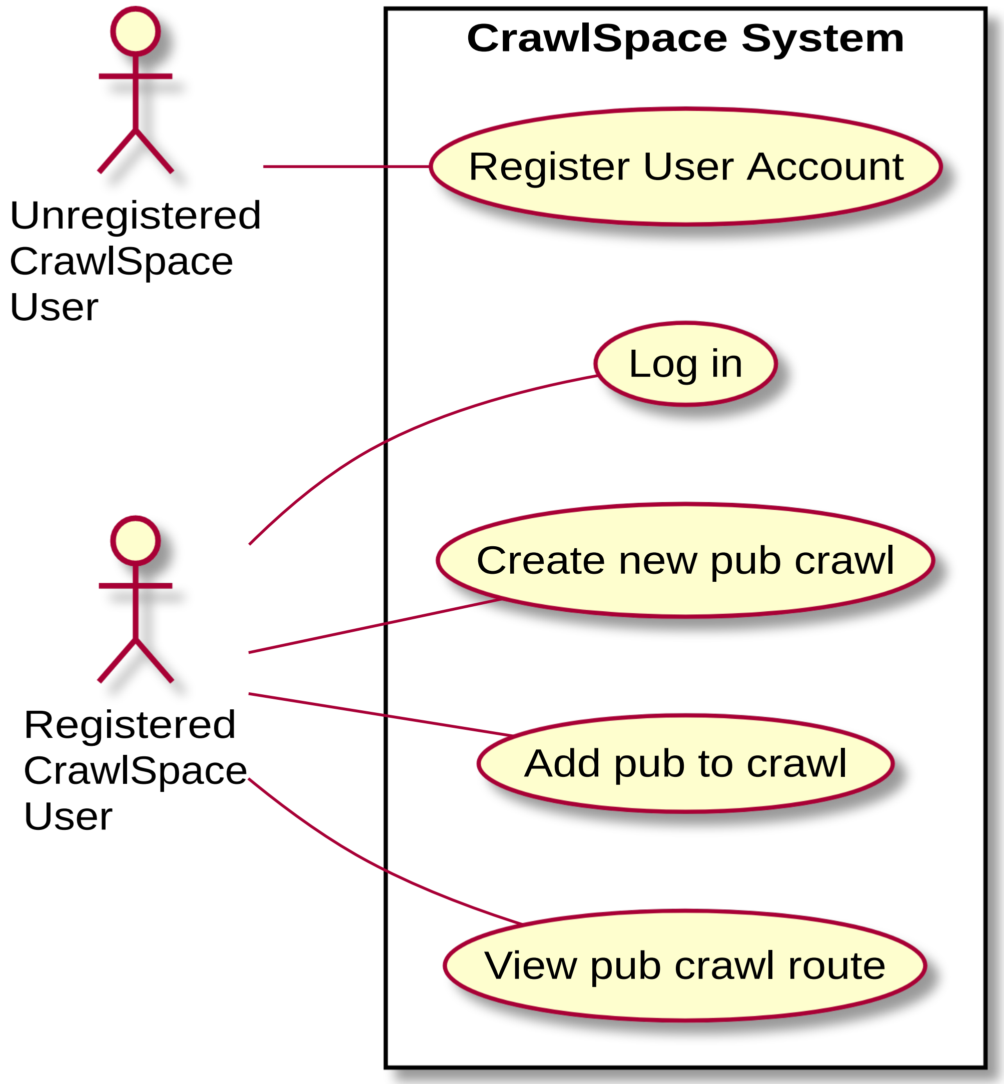
PlantUML Notes

```
@startuml UseCaseNote
:Actor: as a
note right of (a)
  Notes can be a handy way
  to annotate a UML diagram
end note
@enduml
```

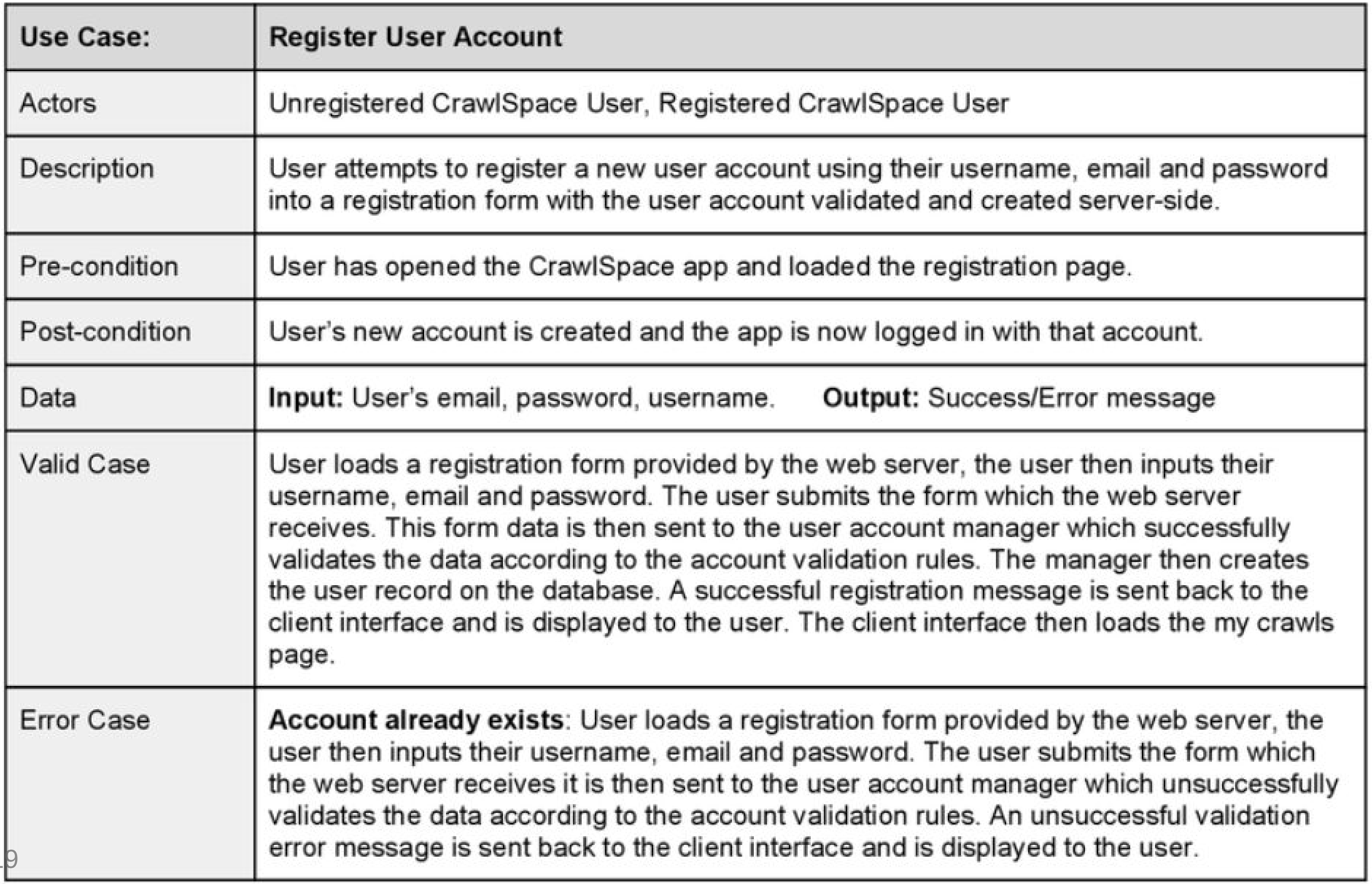


Pub Crawler Case Study

```
@startuml PubCrawlUseCase
left to right direction
:Unregistered\nCrawlSpace\nUser: as u1
:Registered\nCrawlSpace \nUser: as u2
rectangle "CrawlSpace System" as S1 {
    (Register User Account) as F1
    (Log in) as F2
    (Create new pub crawl) as F3
    (Add pub to crawl) as F4
    (View pub crawl route) as F5
    u1 -- F1
    u2 -- F2
    u2 -- F3
    u2 -- F4
    u2 -- F5
}

@enduml
```

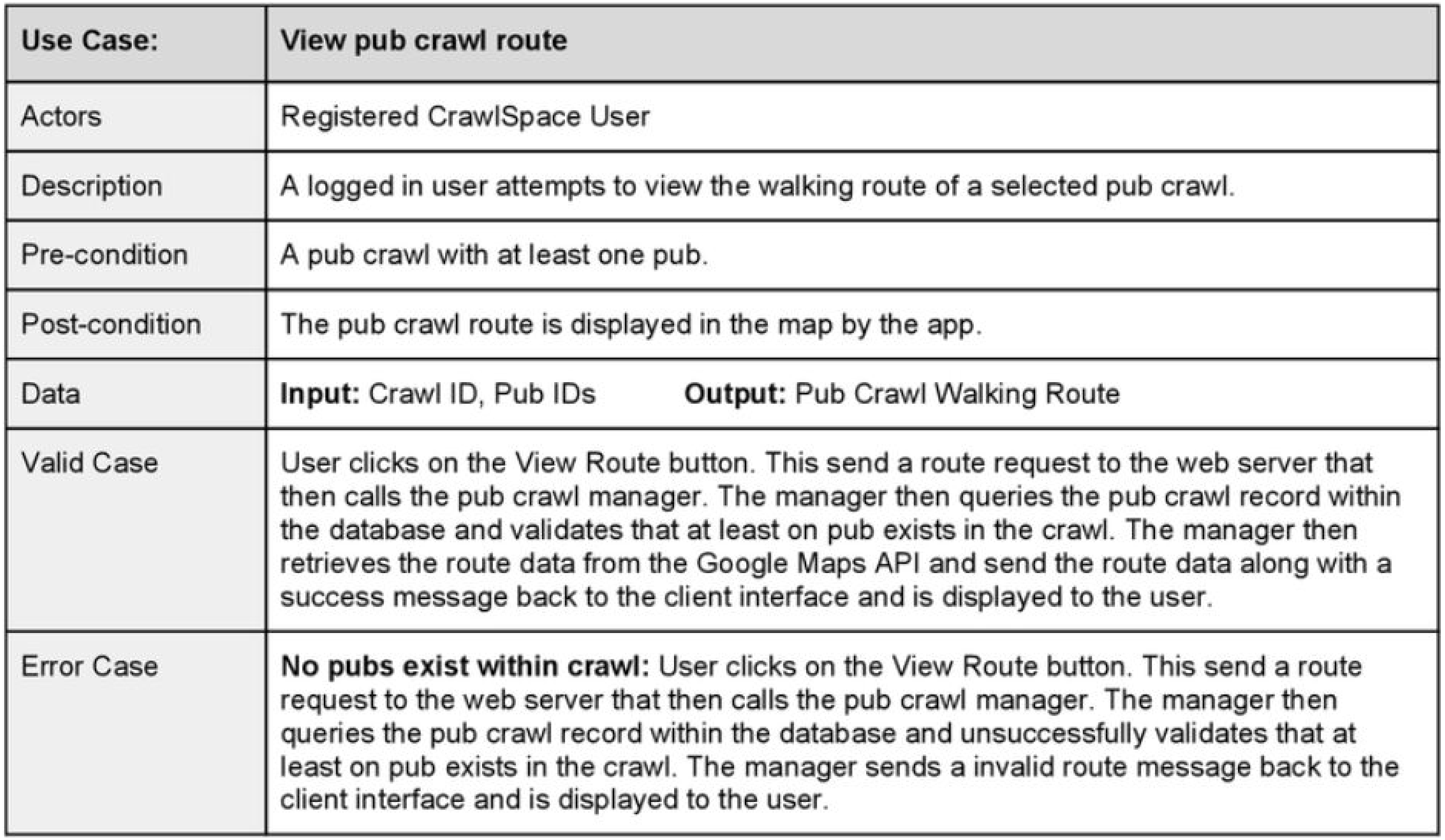


Register user Account



Use Case:	Register User Account
Actors	Unregistered CrawlSpace User, Registered CrawlSpace User
Description	User attempts to register a new user account using their username, email and password into a registration form with the user account validated and created server-side.
Pre-condition	User has opened the CrawlSpace app and loaded the registration page.
Post-condition	User's new account is created and the app is now logged in with that account.
Data	**Input:** User's email, password, username. **Output:** Success/Error message
Valid Case	User loads a registration form provided by the web server, the user then inputs their username, email and password. The user submits the form which the web server receives. This form data is then sent to the user account manager which successfully validates the data according to the account validation rules. The manager then creates the user record on the database. A successful registration message is sent back to the client interface and is displayed to the user. The client interface then loads the my crawls page.
Error Case	**Account already exists**: User loads a registration form provided by the web server, the user then inputs their username, email and password. The user submits the form which the web server receives it is then sent to the user account manager which unsuccessfully validates the data according to the account validation rules. An unsuccessful validation error message is sent back to the client interface and is displayed to the user.

View pub crawl route



Use Case:	**View pub crawl route**
Actors	Registered CrawlSpace User
Description	A logged in user attempts to view the walking route of a selected pub crawl.
Pre-condition	A pub crawl with at least one pub.
Post-condition	The pub crawl route is displayed in the map by the app.
Data	**Input:** Crawl ID, Pub IDs **Output:** Pub Crawl Walking Route
Valid Case	User clicks on the View Route button. This send a route request to the web server that then calls the pub crawl manager. The manager then queries the pub crawl record within the database and validates that at least on pub exists in the crawl. The manager then retrieves the route data from the Google Maps API and send the route data along with a success message back to the client interface and is displayed to the user.
Error Case	**No pubs exist within crawl:** User clicks on the View Route button. This send a route request to the web server that then calls the pub crawl manager. The manager then queries the pub crawl record within the database and unsuccessfully validates that at least on pub exists in the crawl. The manager sends a invalid route message back to the client interface and is displayed to the user.

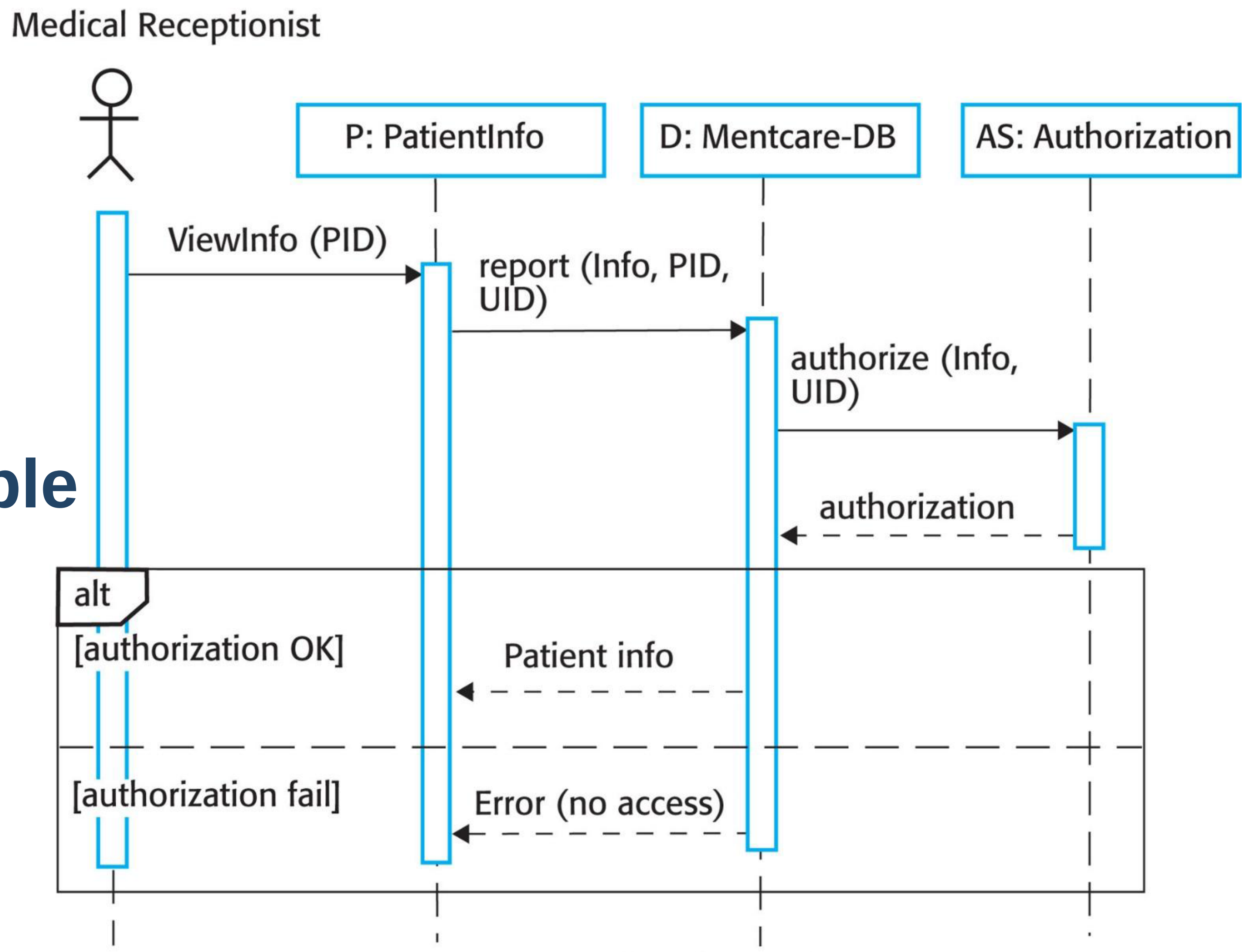
UML - Sequence diagrams

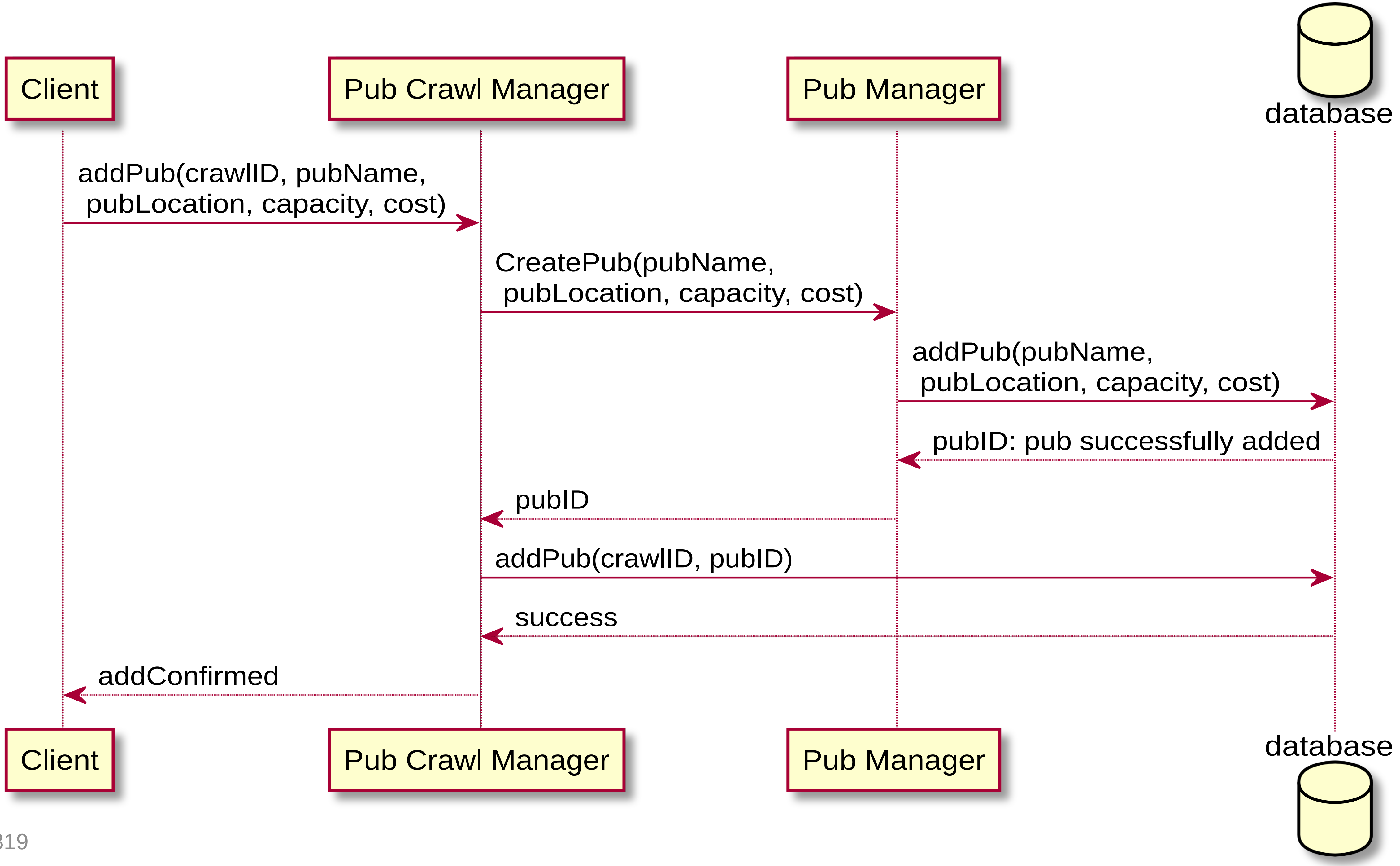
- Sequence diagrams show interactions between actors and the system and between system components.
    - Internal and external
- Shows the sequence of interactions that take place during a particular use case
- Time passes the further down the sequence diagram you go
- Return messages are dashed lines

Reading

- Recap: https://www.youtube.com/watch?v=pCK6prSq8a
- PlantUML: https://plantuml.com/sequence-diagram

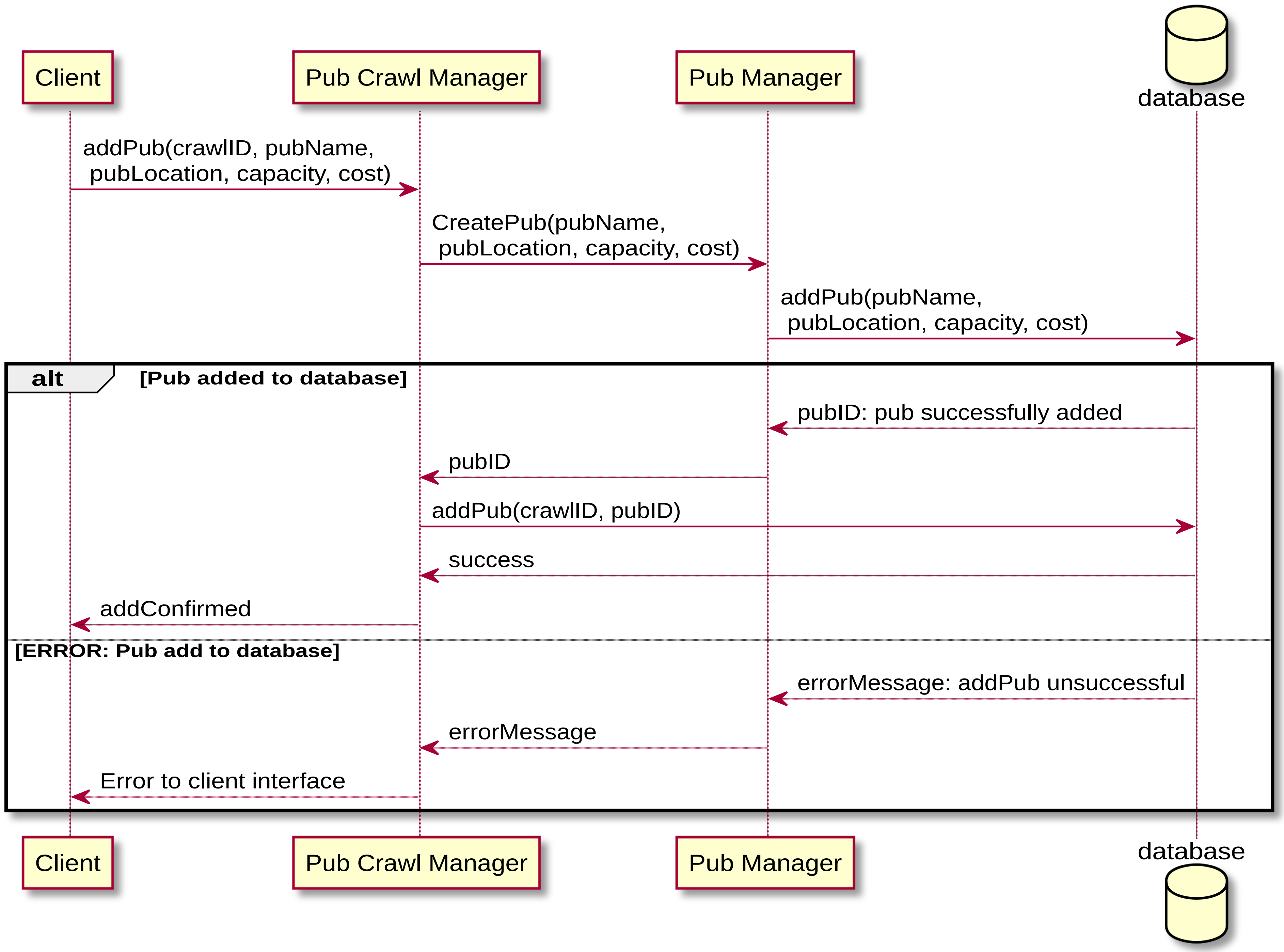
Example


Medical Receptionist
P: PatientInfo
D: Mentcare-DB
AS: Authorization
ViewInfo (PID)
report (Info, PID, UID)
authorize (Info, UID)
authorization
alt
[authorization OK]
Patient info
[authorization fail]
Error (no access)


Client
Pub Crawl Manager
Pub Manager
database
addPub(crawlID, pubName, pubLocation, capacity, cost)
CreatePub(pubName, pubLocation, capacity, cost)
addPub(pubName, pubLocation, capacity, cost)
pubID: pub successfully added
pubID
addPub(crawlID, pubID)
success
addConfirmed
Client
Pub Crawl Manager
Pub Manager
database

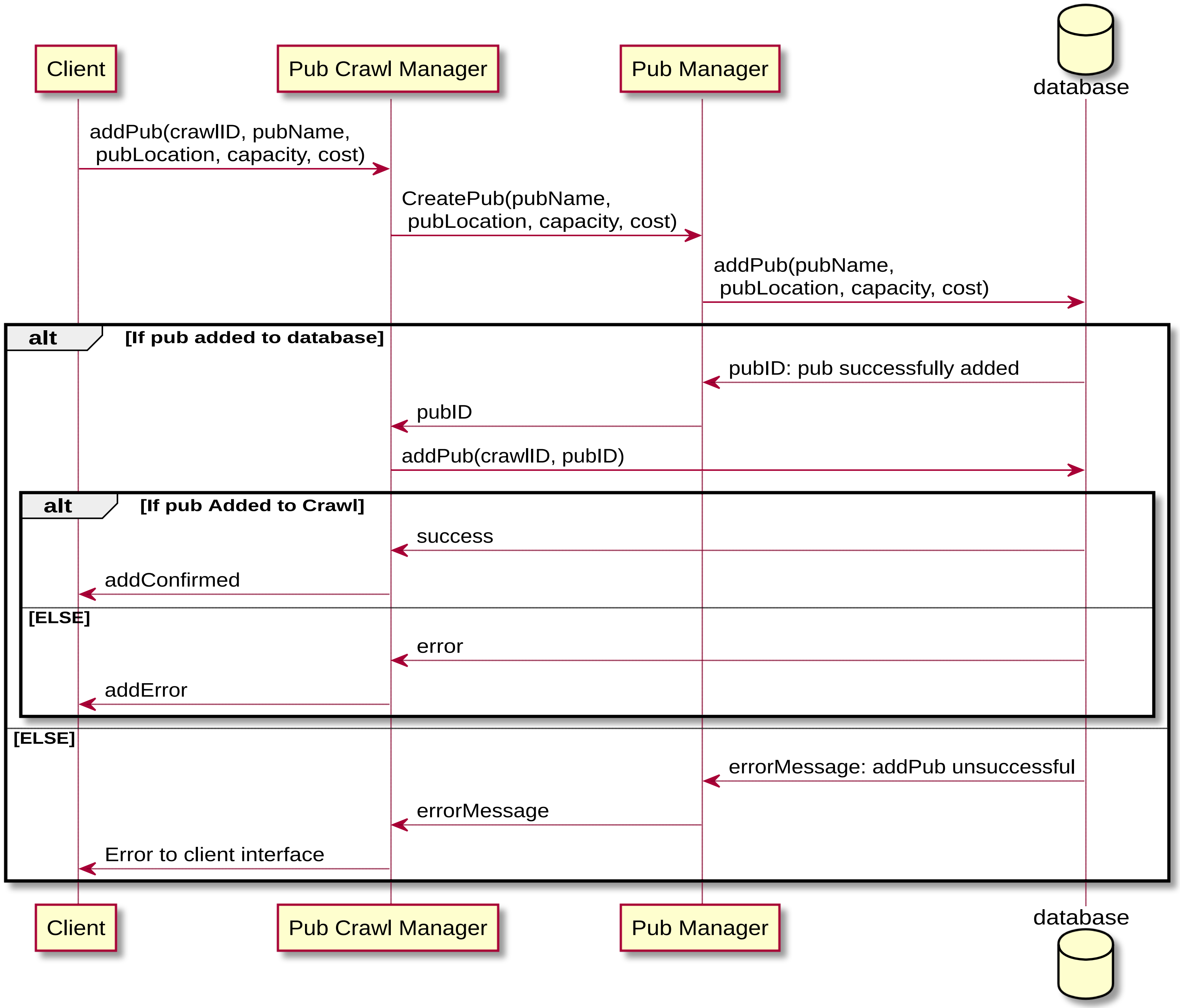
Pubcrawl

```
@startuml crawlspaceSequenceDiagram1

"Client" -> "Pub Crawl Manager"  : addPub(crawlID, pubName,\n pubLocation, capacity, cost)
 "Pub Crawl Manager" -> "Pub Manager" : CreatePub(pubName,\n pubLocation, capacity, cost)
Database database

"Pub Manager" -> database : addPub(pubName,\n pubLocation, capacity, cost)

    database -->   "Pub Manager" : pubID: pub successfully added

    "Pub Manager" -->  "Pub Crawl Manager" : pubID
    "Pub Crawl Manager" -> database : addPub(crawlID, pubID)

        database --> "Pub Crawl Manager" : success
        "Pub Crawl Manager"  --> Client : addConfirmed
@enduml
```

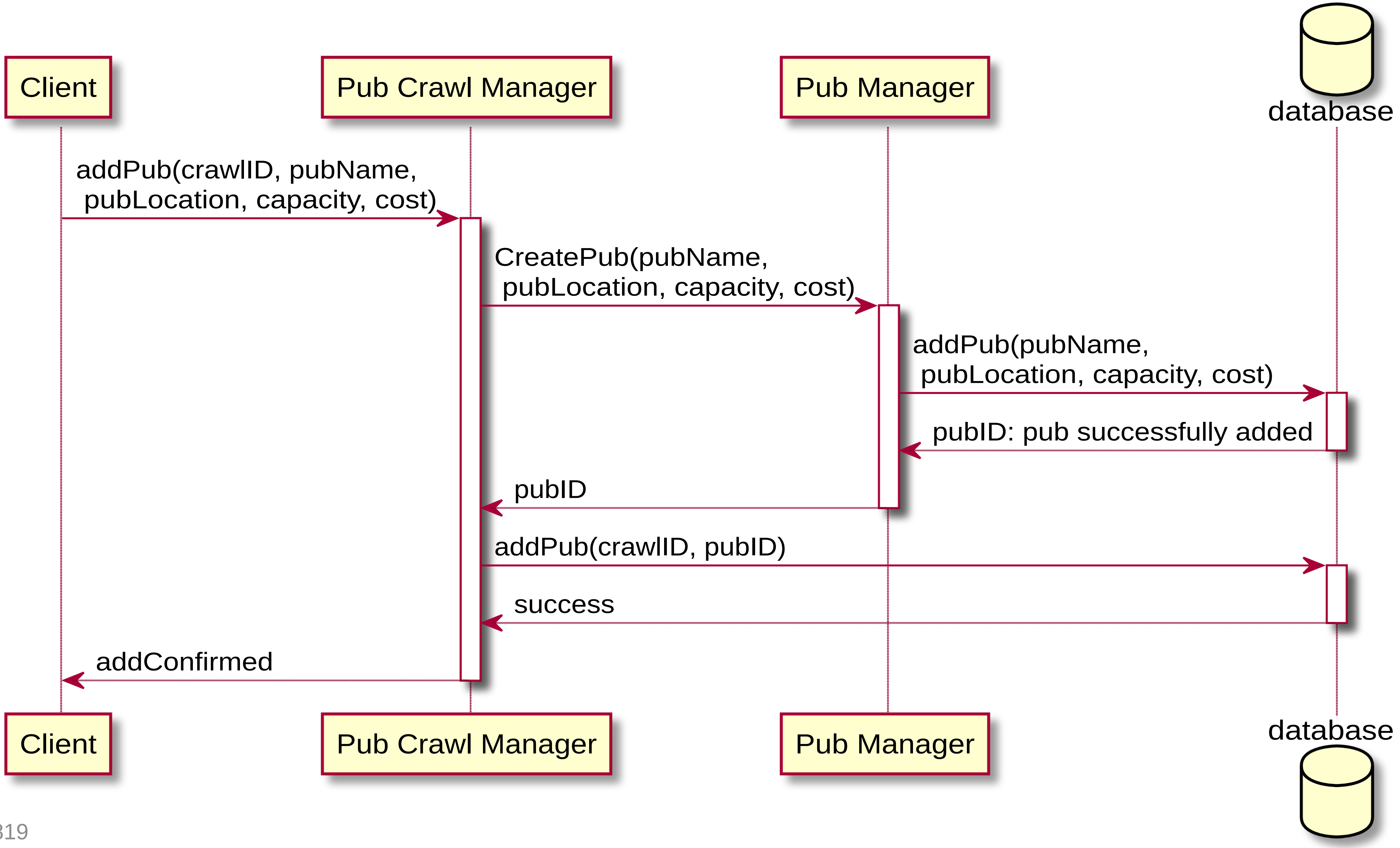

Client
Pub Crawl Manager
Pub Manager
database
addPub(crawlID, pubName, pubLocation, capacity, cost)
CreatePub(pubName, pubLocation, capacity, cost)
addPub(pubName, pubLocation, capacity, cost)
alt
[Pub added to database]
pubID: pub successfully added
pubID
addPub(crawlID, pubID)
success
addConfirmed
[ERROR: Pub add to database]
errorMessage: addPub unsuccessful
errorMessage
Error to client interface
Client
Pub Crawl Manager
Pub Manager
database

Pubcrawl

```
@startuml crawlspaceSequenceDiagram2

Client -> "Pub Crawl Manager"  : addPub(crawlID, pubName,\n pubLocation, capacity, cost)
 "Pub Crawl Manager" -> "Pub Manager" : CreatePub(pubName,\n pubLocation, capacity, cost)
Database database
"Pub Manager" -> database : addPub(pubName,\n pubLocation, capacity, cost)
alt Pub added to database
    database -->   "Pub Manager" : pubID: pub successfully added
    "Pub Manager" -->  "Pub Crawl Manager" : pubID
    "Pub Crawl Manager" -> database : addPub(crawlID, pubID)
        database --> "Pub Crawl Manager" : success
        "Pub Crawl Manager"  --> Client : addConfirmed
else ERROR: Pub add to database
    database -->   "Pub Manager" : errorMessage: addPub unsuccessful
    "Pub Manager"  --> "Pub Crawl Manager" :  errorMessage
    "Pub Crawl Manager"  --> Client : Error to client interface
end
@enduml
```


Client
Pub Crawl Manager
Pub Manager
database
addPub(crawlID, pubName, pubLocation, capacity, cost)
CreatePub(pubName, pubLocation, capacity, cost)
addPub(pubName, pubLocation, capacity, cost)
alt [If pub added to database]
pubID: pub successfully added
pubID
addPub(crawlID, pubID)
alt [If pub Added to Crawl]
success
addConfirmed
[ELSE]
error
addError
[ELSE]
errorMessage: addPub unsuccessful
errorMessage
Error to client interface
Client
Pub Crawl Manager
Pub Manager
database

```
@startuml crawlspaceSequenceDiagram3

Client -> "Pub Crawl Manager"  : addPub(crawlID, pubName,\n pubLocation, capacity, cost)
 "Pub Crawl Manager" -> "Pub Manager" : CreatePub(pubName,\n pubLocation, capacity, cost)
Database database
"Pub Manager" -> database : addPub(pubName,\n pubLocation, capacity, cost)
alt If pub added to database
    database -->   "Pub Manager" : pubID: pub successfully added
    "Pub Manager" -->  "Pub Crawl Manager" : pubID
    "Pub Crawl Manager" -> database : addPub(crawlID, pubID)
    alt If pub Added to Crawl
        database --> "Pub Crawl Manager" : success
        "Pub Crawl Manager"  --> Client : addConfirmed
    else ELSE
    database --> "Pub Crawl Manager" : error
    "Pub Crawl Manager"  --> Client : addError
    end
else ELSE
    database -->   "Pub Manager" : errorMessage: addPub unsuccessful
    "Pub Manager"  --> "Pub Crawl Manager" :  errorMessage
    "Pub Crawl Manager"  --> Client : Error to client interface
end
@enduml
```

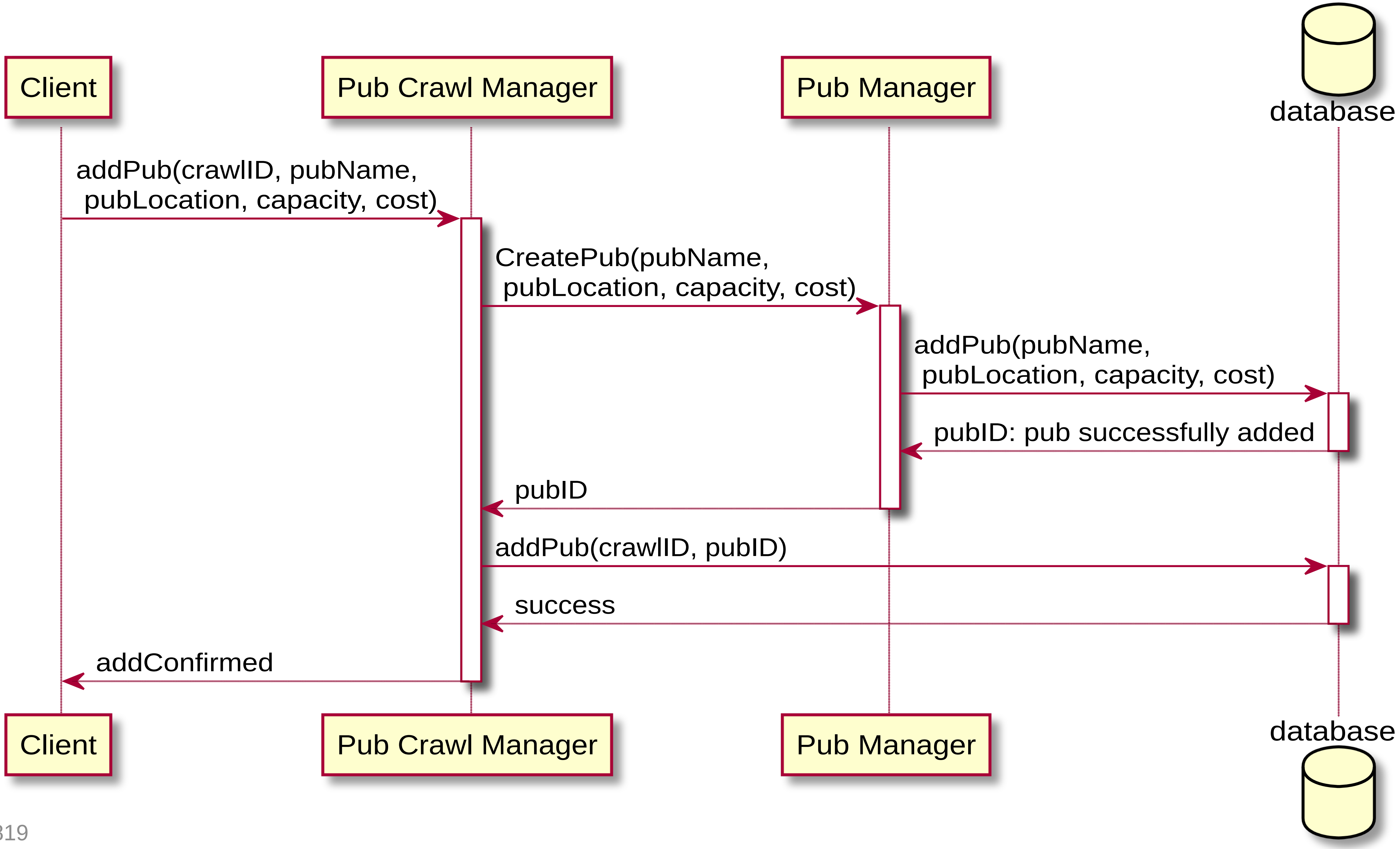

Client
Pub Crawl Manager
Pub Manager
database
addPub(crawlID, pubName, pubLocation, capacity, cost)
CreatePub(pubName, pubLocation, capacity, cost)
addPub(pubName, pubLocation, capacity, cost)
pubID: pub successfully added
pubID
addPub(crawlID, pubID)
success
addConfirmed
Client
Pub Crawl Manager
Pub Manager
database

Pubcrawl

```
@startuml crawlspaceSequenceDiagram4


Client -> "Pub Crawl Manager"  : addPub(crawlID, pubName,\n pubLocation, capacity, cost)
activate "Pub Crawl Manager"
 "Pub Crawl Manager" -> "Pub Manager" : CreatePub(pubName,\n pubLocation, capacity, cost)
 activate "Pub Manager"
Database database

"Pub Manager" -> database : addPub(pubName,\n pubLocation, capacity, cost)
    activate database
        database -->   "Pub Manager" : pubID: pub successfully added
    deactivate database
        "Pub Manager" -->  "Pub Crawl Manager" : pubID
    deactivate "Pub Manager"
    "Pub Crawl Manager" -> database : addPub(crawlID, pubID)
    activate database
        database --> "Pub Crawl Manager" : success
    deactivate database
        "Pub Crawl Manager"  --> Client : addConfirmed
        deactivate "Pub Crawl Manager"
@enduml
```

Auto activate

```
@startuml crawlspaceSequenceDiagramAuto
autoactivate on


Client -> "Pub Crawl Manager"  : addPub(crawlID, pubName,\n pubLocation, capacity, cost)
 "Pub Crawl Manager" -> "Pub Manager" : CreatePub(pubName,\n pubLocation, capacity, cost)
Database database

"Pub Manager" -> database : addPub(pubName,\n pubLocation, capacity, cost)

    database -->   "Pub Manager" : pubID: pub successfully added

    "Pub Manager" -->  "Pub Crawl Manager" : pubID
    "Pub Crawl Manager" -> database : addPub(crawlID, pubID)

        database --> "Pub Crawl Manager" : success
        "Pub Crawl Manager"  --> Client : addConfirmed
@enduml
```


Client
Pub Crawl Manager
Pub Manager
database
addPub(crawlID, pubName, pubLocation, capacity, cost)
CreatePub(pubName, pubLocation, capacity, cost)
addPub(pubName, pubLocation, capacity, cost)
pubID: pub successfully added
pubID
addPub(crawlID, pubID)
success
addConfirmed
Client
Pub Crawl Manager
Pub Manager
database

Questions?

- Tsui Frank, Karam Orlando, Bernal Barbara. Essentials of software engineering
- Sommerville Ian. 2010. Software Engineering, 10th Edition, Pearson
- Grace Lewis, Carnegie Mellon University, SEI