

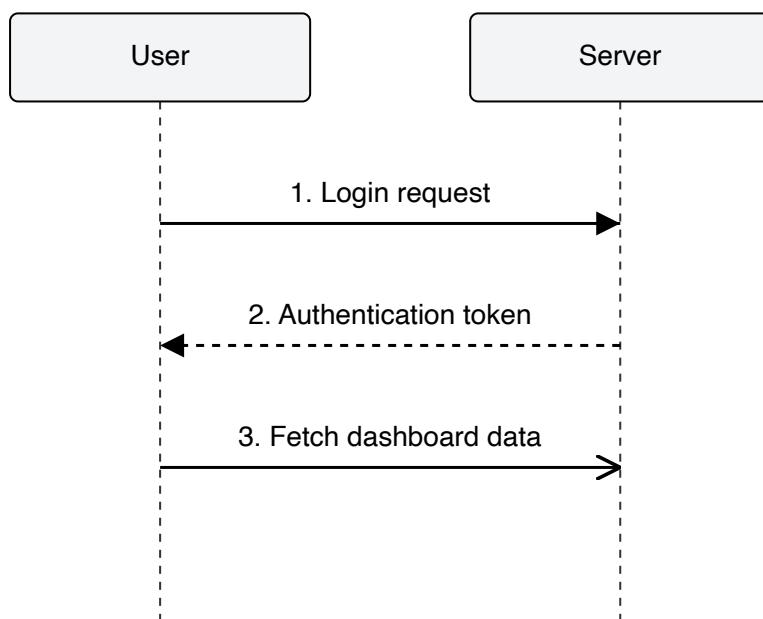
# Mermaid Sequence Diagrams Documentation

*Important Note: All Mermaid sequence diagram code blocks must start with `sequenceDiagram` on the first line to be properly parsed and rendered.*

## Basic Example

### sequenceDiagram

```
User->>Server: Login request
Server-->>User: Authentication token
User-)Server: Fetch dashboard data
```



## Participants

Participants may be declared implicitly, as shown in the basic example. By default, participants and actors appear in the order they first appear in your diagram code. However, you may want to display them in a different sequence than their initial message order. You can control the display order by explicitly declaring participants:

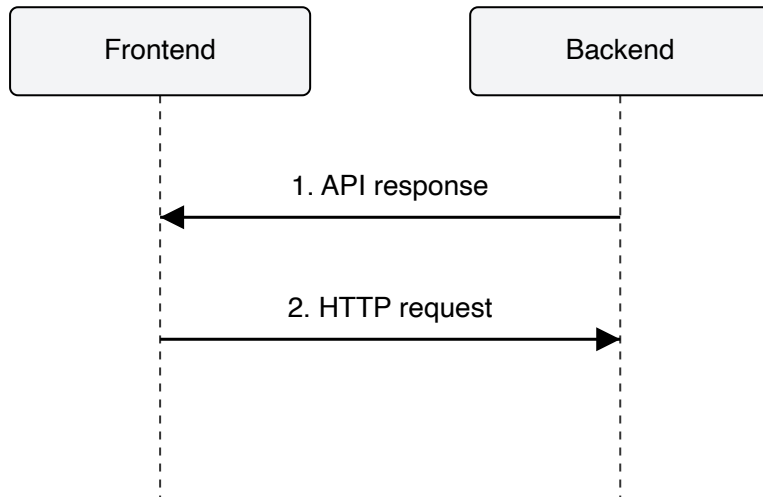
### sequenceDiagram

**participant** Frontend

**participant** Backend

Backend->>Frontend: API response

Frontend->>Backend: HTTP request



## Actors

To display actors using the stick figure symbol rather than a rectangular box with text, use the actor declaration syntax shown below.

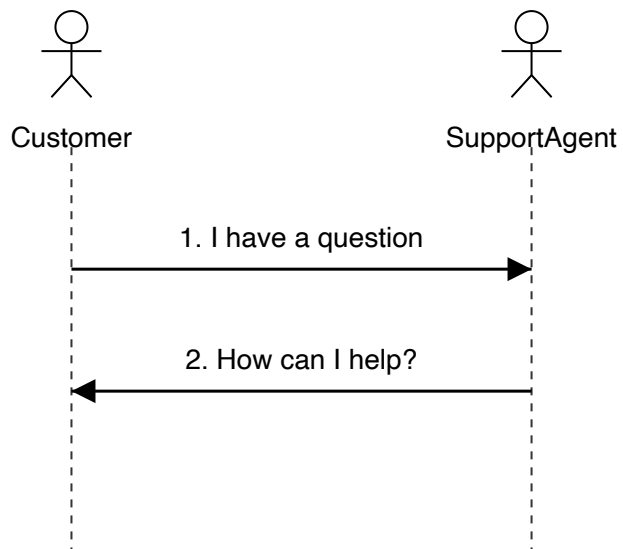
### sequenceDiagram

**actor** Customer

**actor** SupportAgent

Customer->>SupportAgent: I have a question

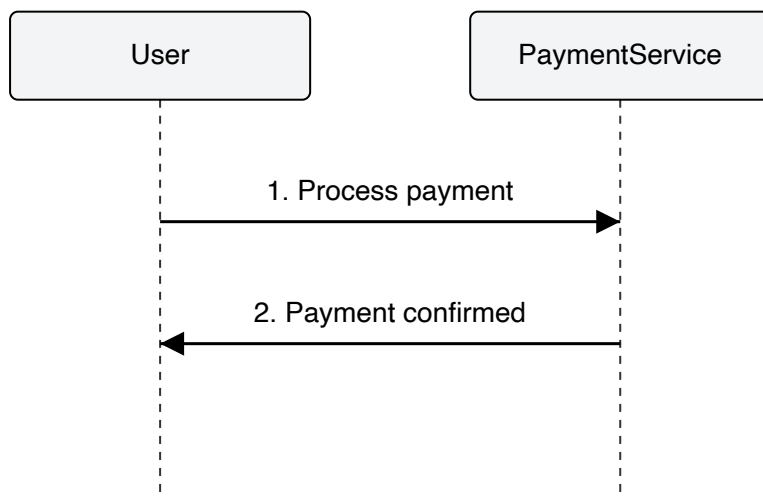
SupportAgent->>Customer: How can I help?



## Aliases

You can assign a short identifier to an actor while displaying a more descriptive label.

```
sequenceDiagram
    participant U as User
    participant S as PaymentService
    U->>S: Process payment
    S->>U: Payment confirmed
```

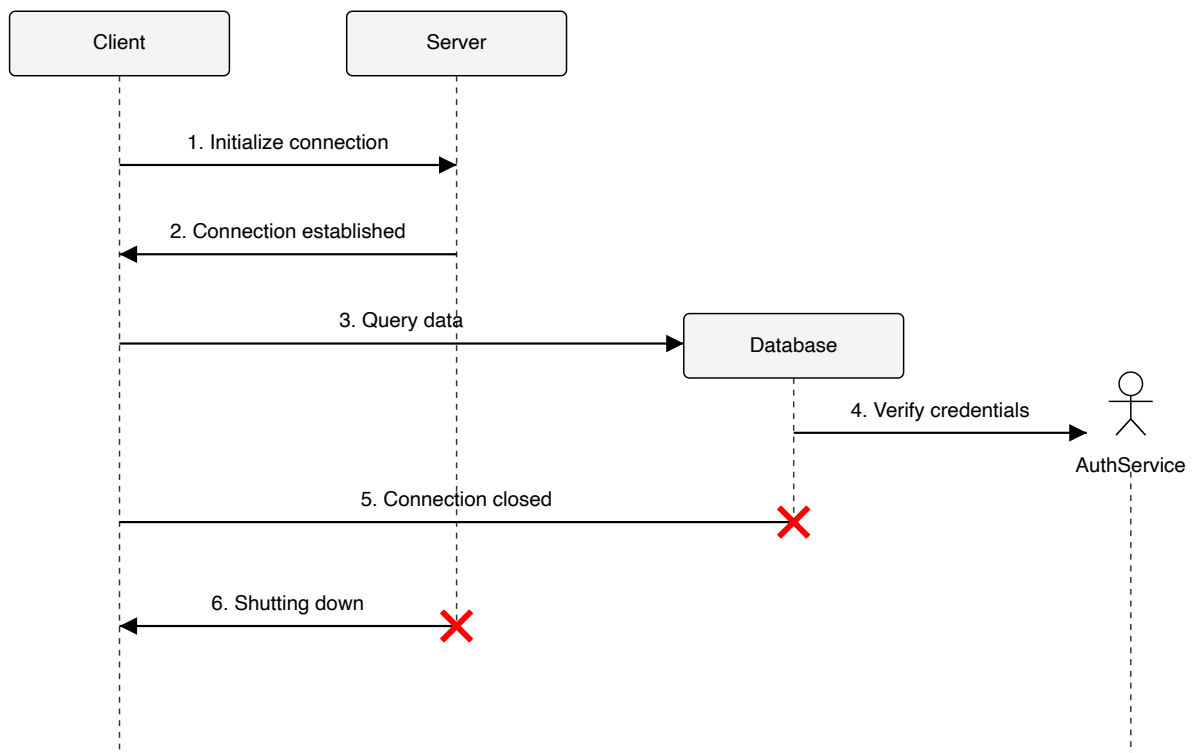


# Actor Creation and Destruction

The create directive works with both actors and participants, and supports aliases. You can destroy either the sender or receiver of a message, but only the recipient can be created dynamically.

## sequenceDiagram

```
Client->>Server: Initialize connection
Server->>Client: Connection established
create participant Database
Client->>Database: Query data
create actor A as AuthService
Database->>A: Verify credentials
destroy Database
Client-xDatabase: Connection closed
destroy Server
Server->>Client: Shutting down
```



## Grouping / Box

Actors and participants can be visually grouped together using vertical boxes. These boxes can include an optional color (defaults to transparent if omitted) and a descriptive label, as demonstrated here:

sequenceDiagram

box Purple Frontend Services

participant WebApp

participant MobileApp

end

box Another Group

participant API

participant Cache

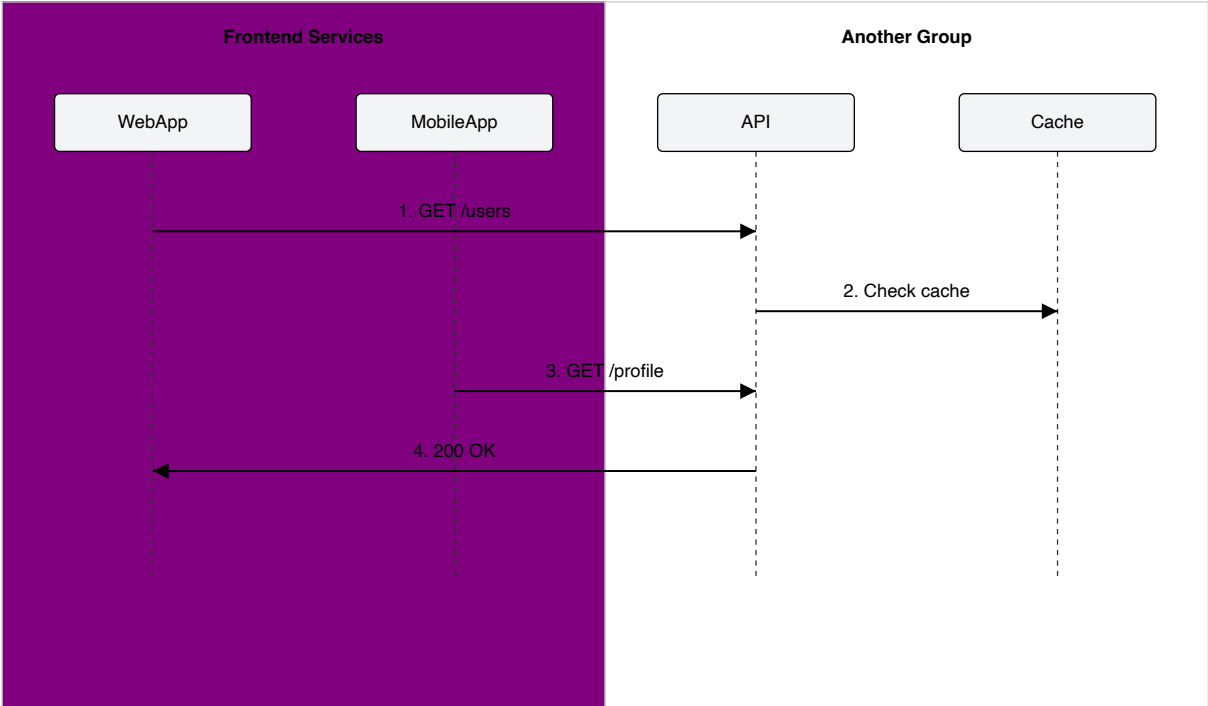
end

WebApp->>API: GET /users

API->>Cache: Check cache

MobileApp->>API: GET /profile

API->>WebApp: 200 OK



Messages

Message arrows can be rendered using either solid or dotted line styles.

Type	Description
->	Solid line without arrow
-->	Dotted line without arrow
-->>	Solid line with arrowhead

Type	Description
<code>--&gt;&gt;</code>	Dotted line with arrowhead
<code>&lt;&lt;--&gt;&gt;</code>	Solid line with bidirectional arrowheads
<code>&lt;&lt;--&gt;&gt;</code>	Dotted line with bidirectional arrowheads
<code>-x</code>	Solid line with a cross at the end
<code>--x</code>	Dotted line with a cross at the end
<code>-)</code>	Solid line with an open arrow at the end (async)
<code>--)</code>	Dotted line with an open arrow at the end (async)

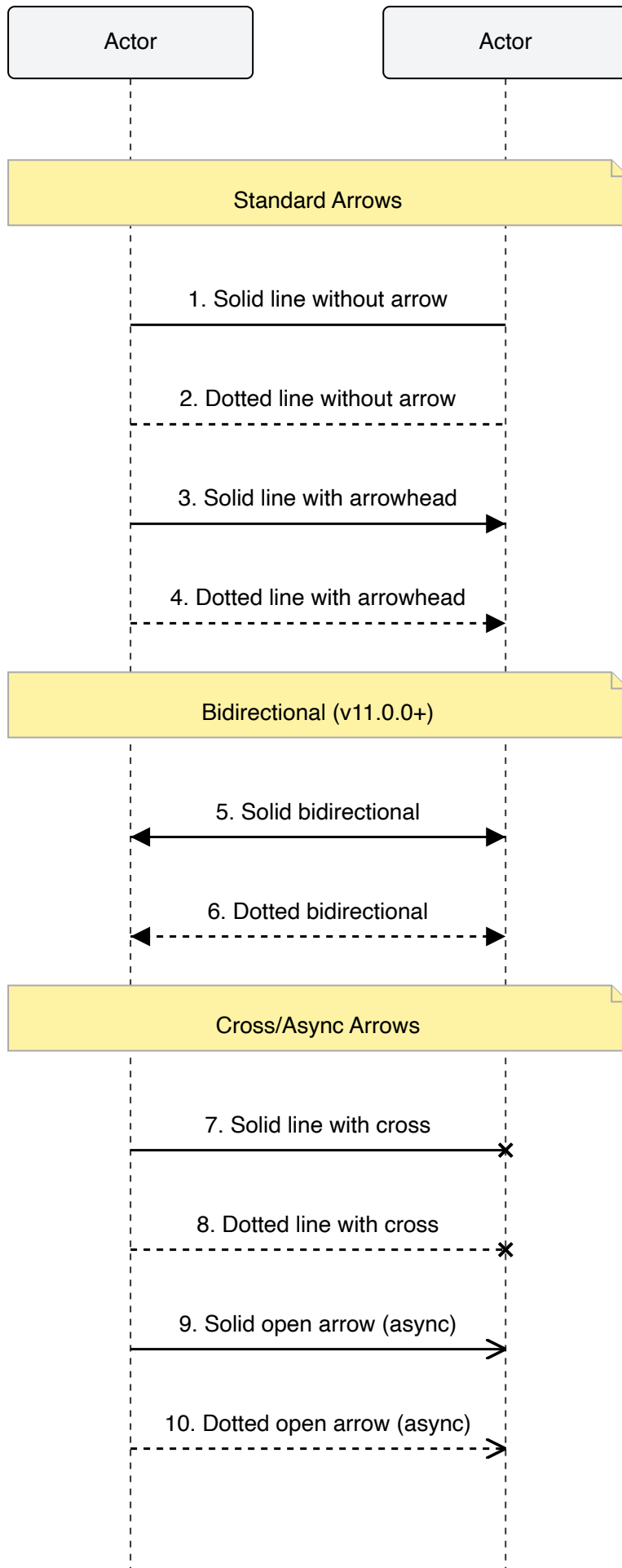
**sequenceDiagram**

```
participant A as Actor A
participant B as Actor B
```

```
Note over A,B: Standard Arrows
A -> B: Solid line without arrow
A --> B: Dotted line without arrow
A ->> B: Solid line with arrowhead
A -->> B: Dotted line with arrowhead
```

```
Note over A,B: Bidirectional (v11.0.0+)
A <<-->> B: Solid bidirectional
A <<-->> B: Dotted bidirectional
```

```
Note over A,B: Cross/Async Arrows
A -x B: Solid line with cross
A --x B: Dotted line with cross
A -) B: Solid open arrow (async)
A --) B: Dotted open arrow (async)
```

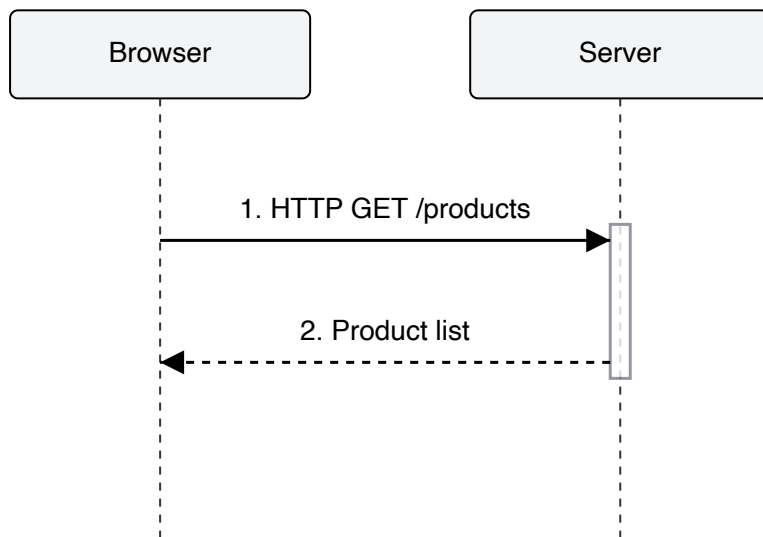


# Activations

You can activate and deactivate actors to show when they are active. Activation and deactivation can be declared explicitly:

## sequenceDiagram

```
Browser->>Server: HTTP GET /products
activate Server
Server-->>Browser: Product list
deactivate Server
```

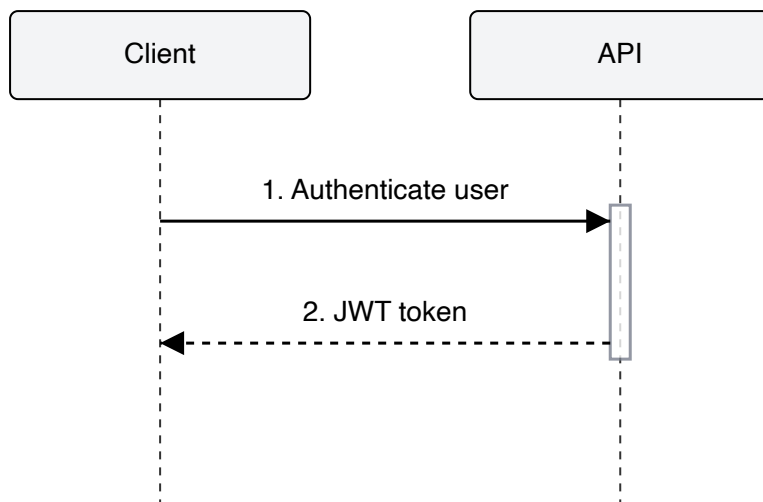


Alternatively, you can use a shorthand by adding a +/- suffix directly to the message arrow:

## sequenceDiagram

```
Client->>+API: Authenticate user
API-->>-Client: JWT token
```

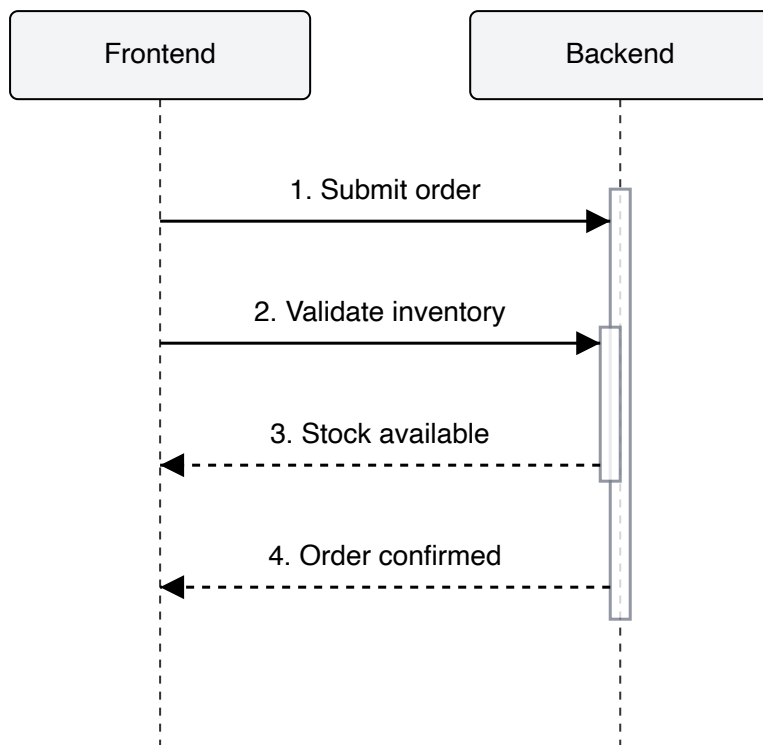




Multiple activations can be stacked for the same actor:

#### sequenceDiagram

```
Frontend->>+Backend: Submit order
Frontend->>+Backend: Validate inventory
Backend-->>-Frontend: Stock available
Backend-->>-Frontend: Order confirmed
```

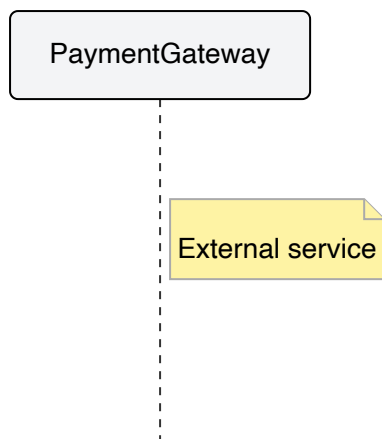


## Notes

You can add annotations to sequence diagrams using notes. The syntax is **Note [ right of | left of | over ] [Actor]: Text in note content**

Here's an example:

```
sequenceDiagram
    participant PaymentGateway
    note right of PaymentGateway: External service
```

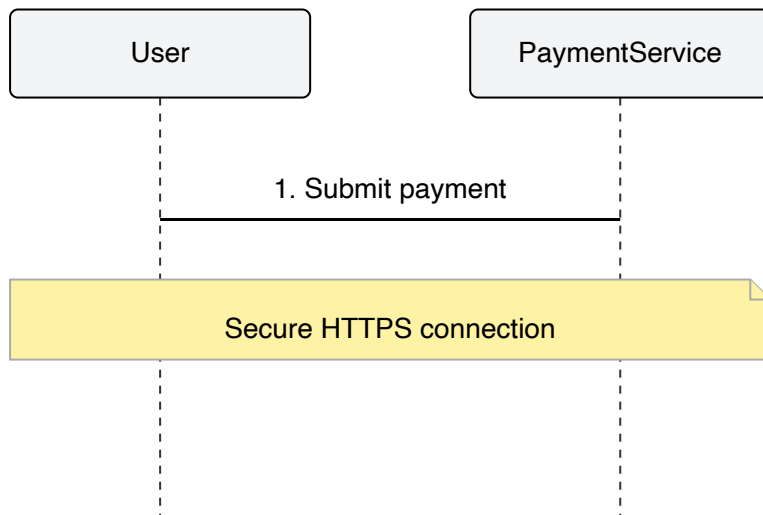


Notes can also span across two participants:

#### sequenceDiagram

User->>PaymentService: Submit payment

Note over User,PaymentService: Secure HTTPS connection



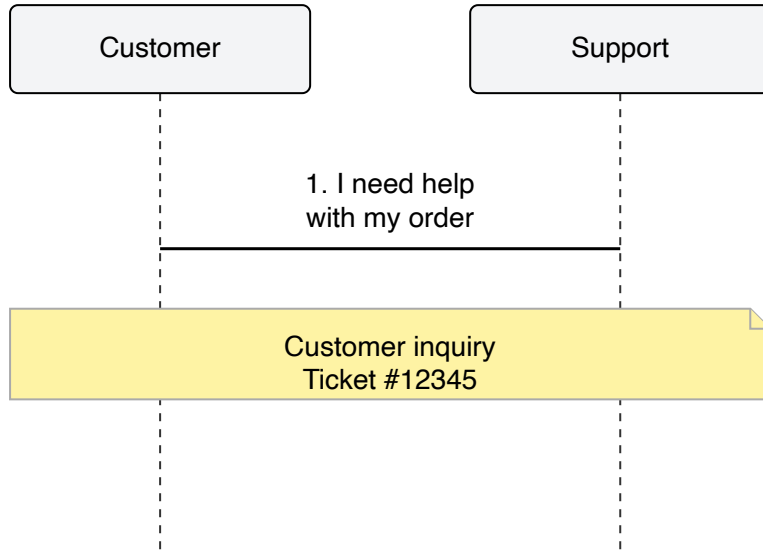
## Line Breaks

You can insert line breaks within Notes and Messages:

### sequenceDiagram

Customer->>Support: I need help<br/>with my order

Note over Customer,Support: Customer inquiry<br/>Ticket #12345



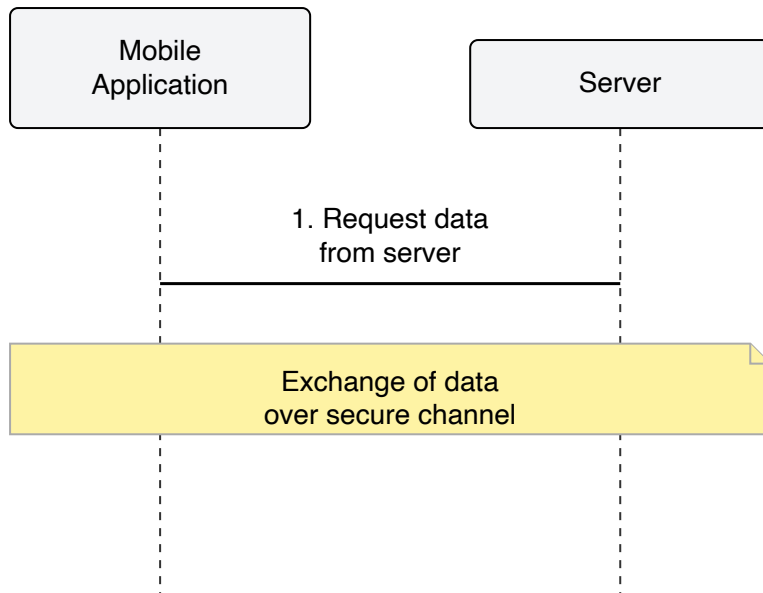
To include line breaks in actor names, you must use aliases:

### sequenceDiagram

participant User as Mobile<br/>Application

User->>Server: Request data<br/>from server

Note over User,Server: Exchange of data<br/>over secure channel



## Loops

You can represent iterative processes in sequence diagrams using the loop syntax:

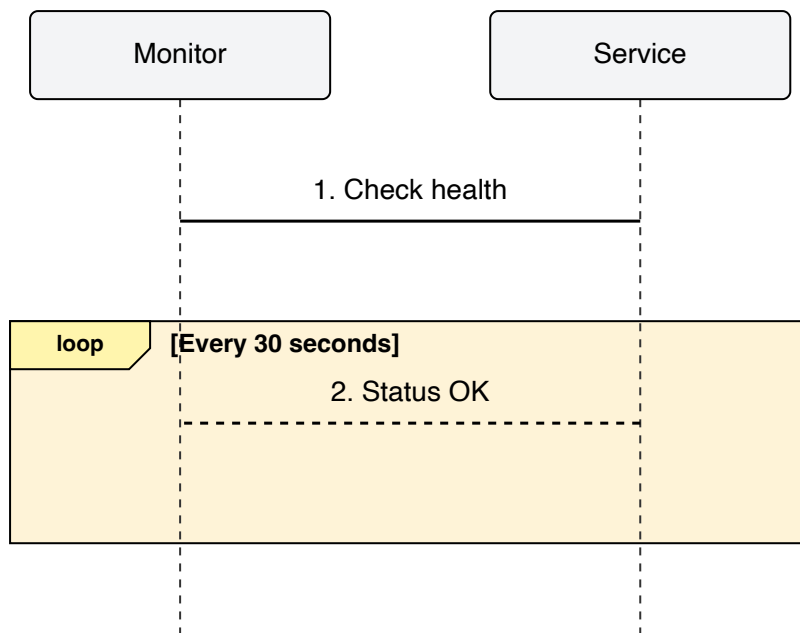
### **sequenceDiagram**

Monitor->>Service: Check health

**loop** Every 30 seconds

Service-->>Monitor: Status OK

**end**

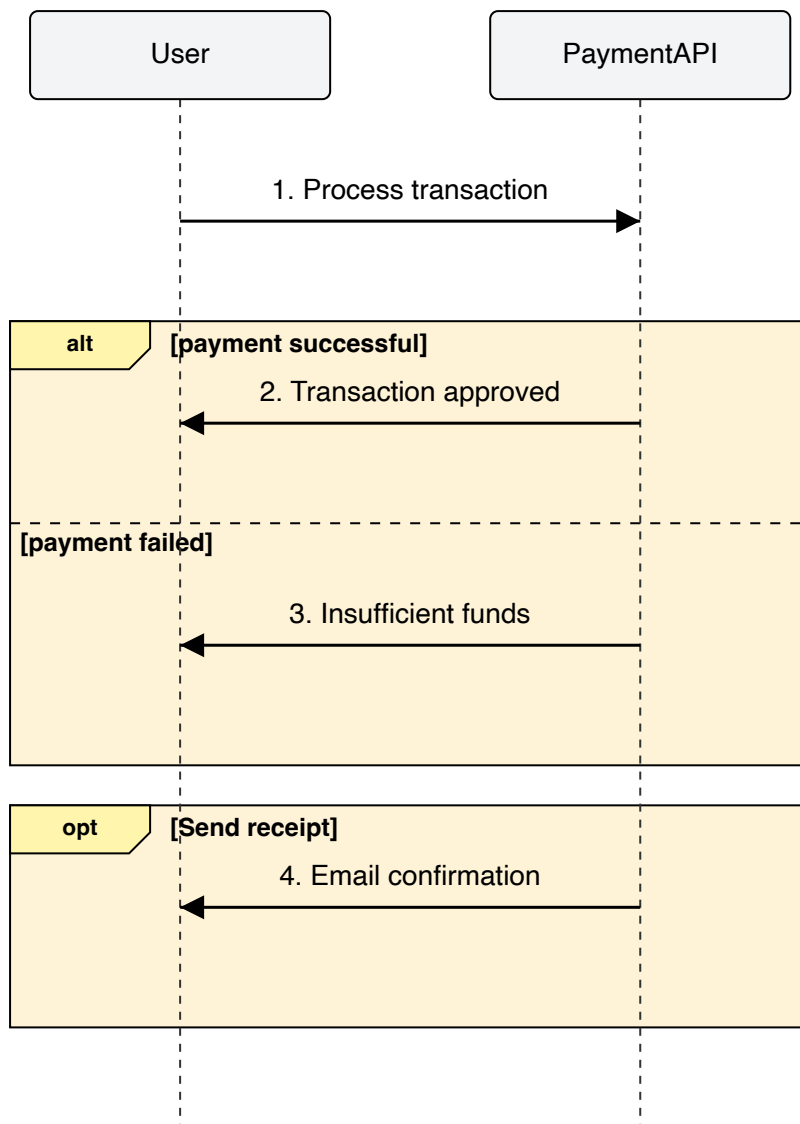


## Alt

Sequence diagrams support conditional logic through alternative paths. Use the alt notation for if-else scenarios, or opt for optional sequences (if without else).

### sequenceDiagram

```
User->>PaymentAPI: Process transaction
alt payment successful
    PaymentAPI->>User: Transaction approved
else payment failed
    PaymentAPI->>User: Insufficient funds
end
opt Send receipt
    PaymentAPI->>User: Email confirmation
end
```



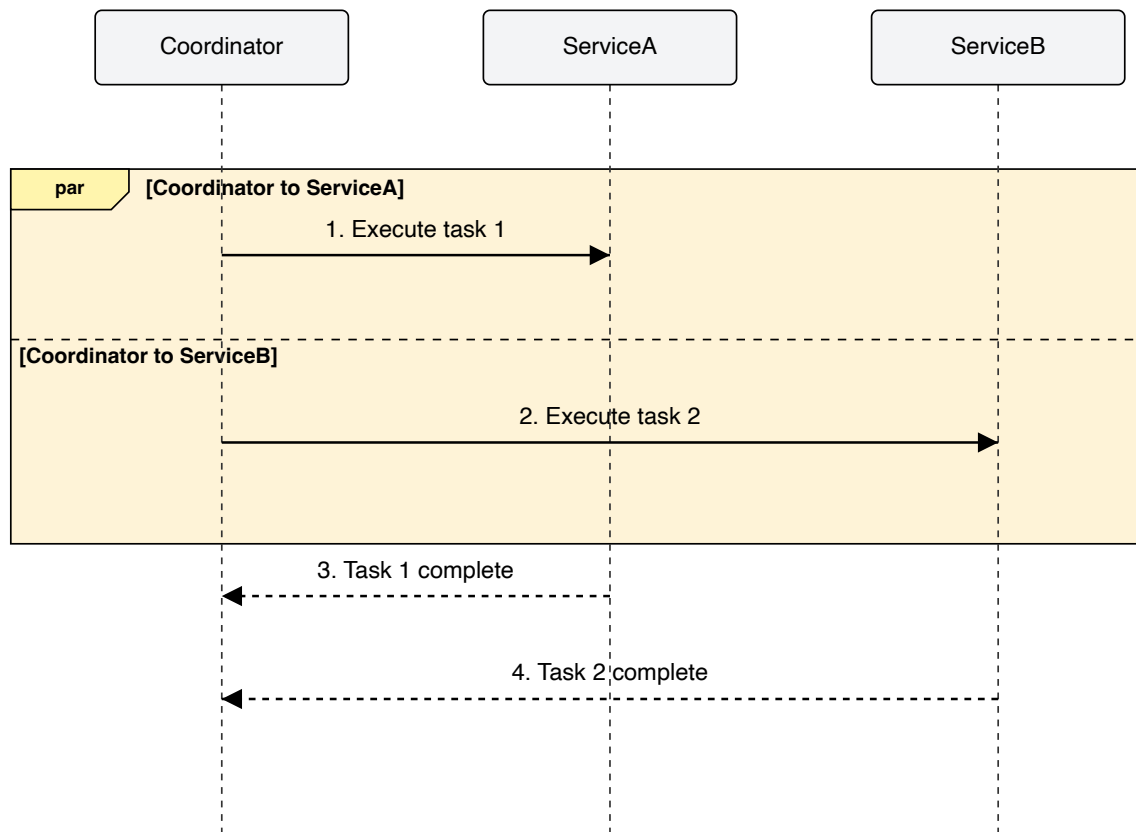
## Parallel

You can illustrate concurrent actions using parallel blocks.

Example:

```

sequenceDiagram
    par Coordinator to ServiceA
        Coordinator->>ServiceA: Execute task 1
    and Coordinator to ServiceB
        Coordinator->>ServiceB: Execute task 2
    end
    ServiceA-->>Coordinator: Task 1 complete
    ServiceB-->>Coordinator: Task 2 complete
  
```



Parallel blocks can be nested within each other:

```

sequenceDiagram
    par Manager to Developer
        Manager->>Developer: Review PR #123
    and Manager to QA
        Manager->>QA: Test feature branch
        par QA to TestEnv
            QA->>TestEnv: Deploy to staging
        and QA to AutomationSuite
            QA->>AutomationSuite: Run integration tests
        end
    end
  
```

![[Nested Parallel Block]](<http://localhost:3000/markdown-to-pdf/marmaid-images/17-nested-parallel-block.svg>)

## Critical Region

Critical regions represent actions that must execute automatically, with conditional error handling.



Example:

### sequenceDiagram

**critical** Establish a connection to the DB

Service-->>DB: connect

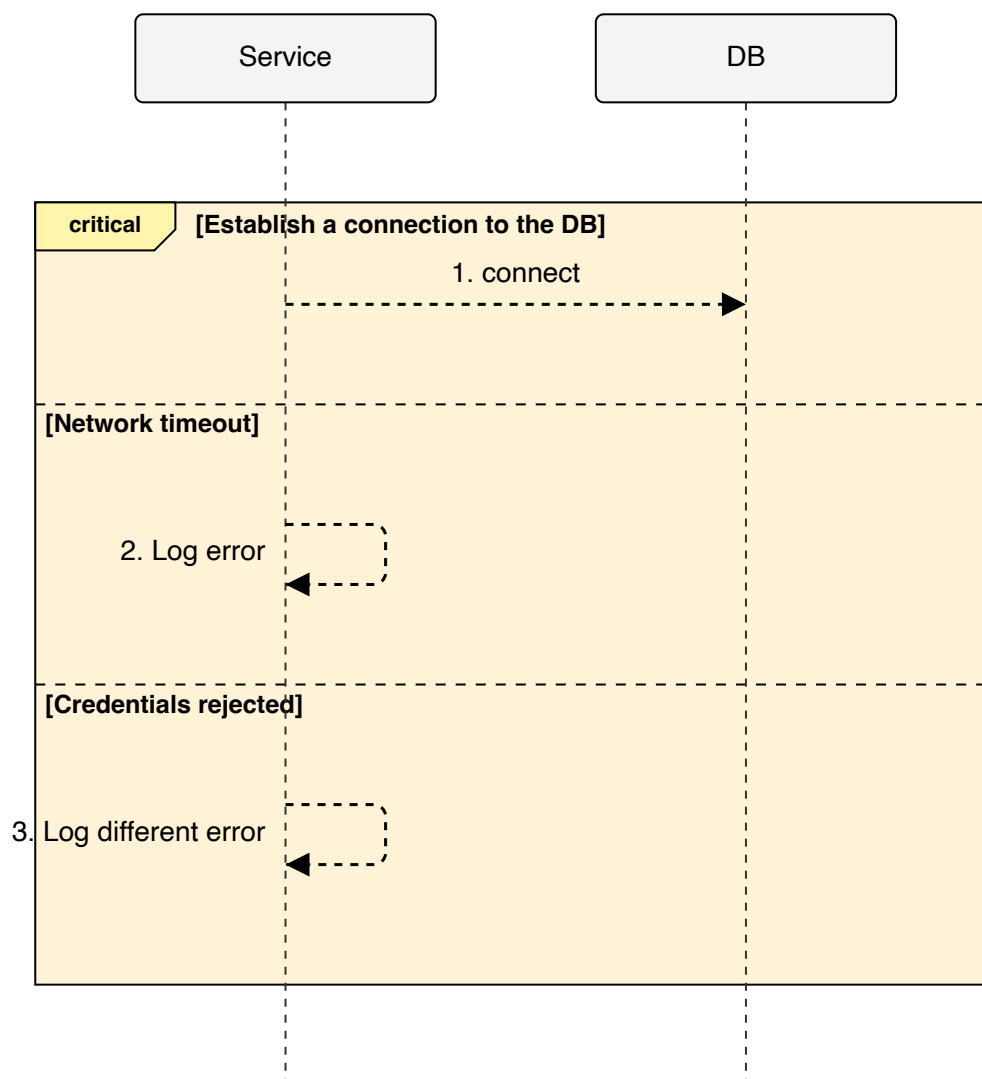
**option** Network timeout

Service-->>Service: Log error

**option** Credentials rejected

Service-->>Service: Log different error

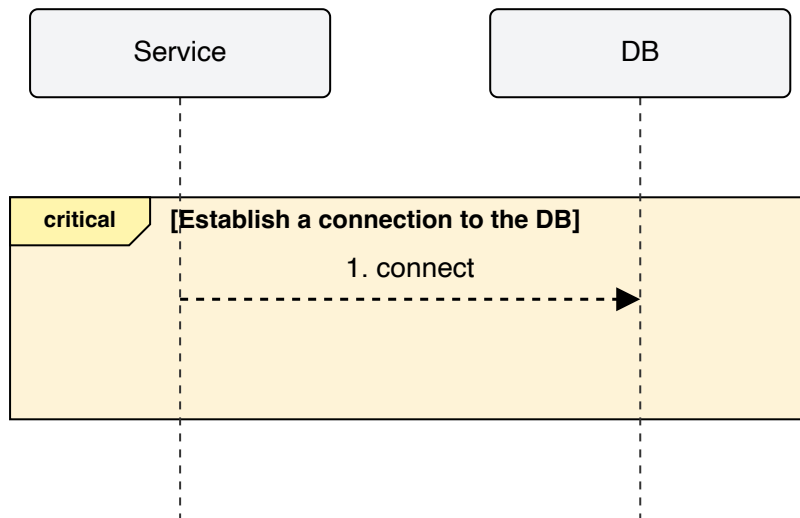
**end**



You can also define a critical region without any error handling options:

### sequenceDiagram

```
critical Establish a connection to the DB
    Service-->>DB: connect
end
```



Critical blocks support nesting, similar to parallel blocks as demonstrated earlier.

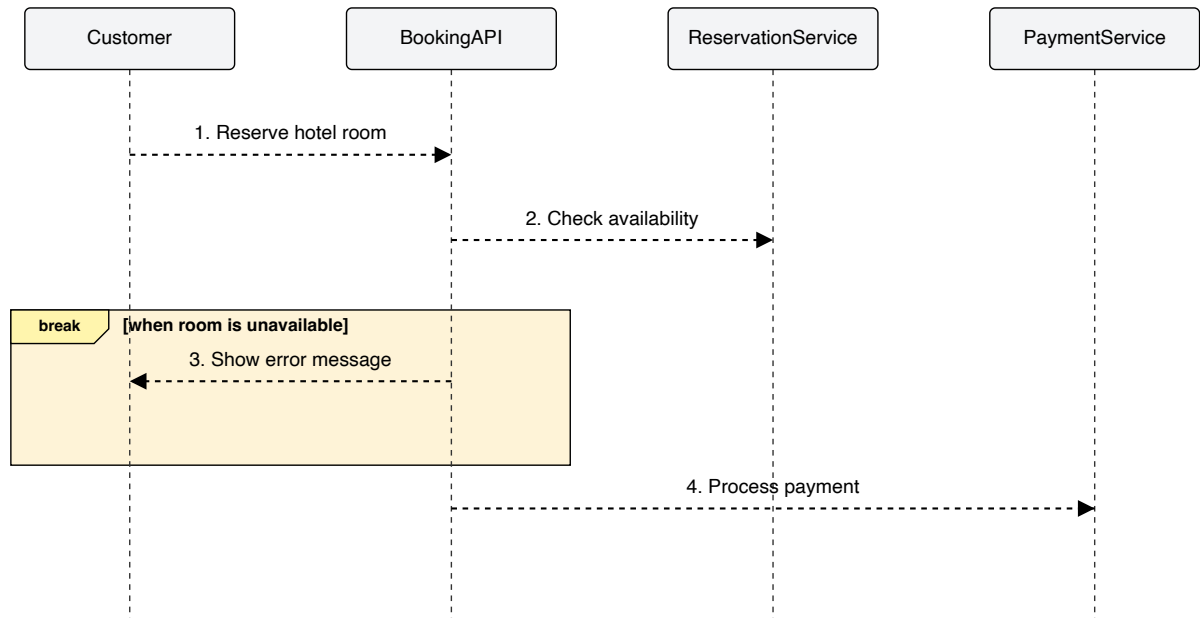
## Break

Break statements allow you to interrupt the normal flow of a sequence, typically used to model exception handling.

Example:

### sequenceDiagram

```
Customer-->>BookingAPI: Reserve hotel room
BookingAPI-->>ReservationService: Check availability
break when room is unavailable
    BookingAPI-->>Customer: Show error message
end
BookingAPI-->>PaymentService: Process payment
```



## Background Highlighting

You can emphasize specific sections of your diagram by adding colored background rectangles. The syntax is:

```
rect COLOR
... content ...
end
```

Colors can be specified using either rgb or rgba format.

```
rect rgb(0, 255, 0)
... content ...
end
```

```
rect rgba(0, 0, 255, .1)
... content ...
end
```

Here are some examples:

## sequenceDiagram

participant Frontend

participant Backend

rect rgb(191, 223, 255)

note right of Frontend: User authentication flow

Frontend->>+Backend: Login credentials

rect rgb(200, 150, 255)

Frontend->>+Backend: Request session token

Backend-->>-Frontend: Session created

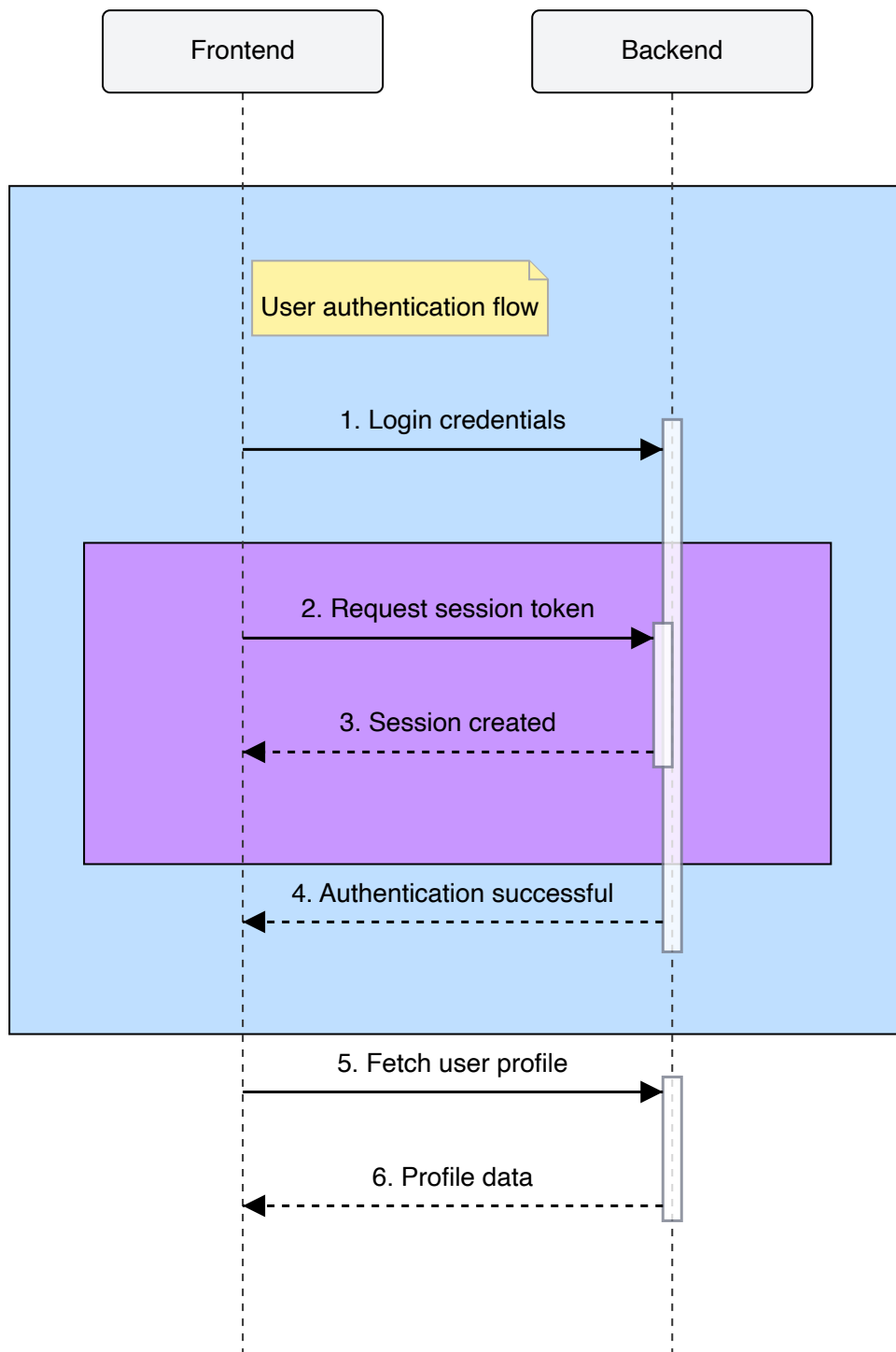
end

Backend-->>-Frontend: Authentication successful

end

Frontend ->>+ Backend: Fetch user profile

Backend -->>- Frontend: Profile data



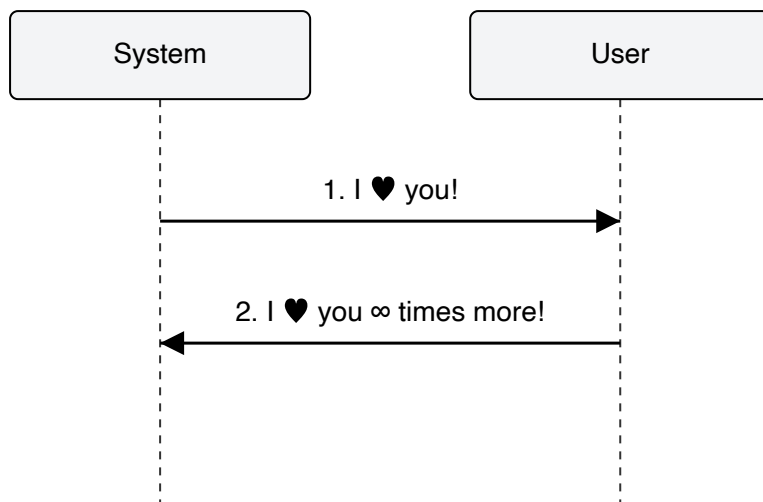
## Entity Codes to Escape Characters

Special characters can be escaped using entity codes, as shown in the following example.

### sequenceDiagram

System->>User: I #9829; you!

User->>System: I #9829; you #infin; times more!



---

## Support

For additional help or questions about SparkChart Pro:

- Website: <https://sparkchart.pro>
- Email: [sparkchartpro@gmail.com](mailto:sparkchartpro@gmail.com)

---

## Version 1.0

*SparkChart Pro - Professional UML Sequence Diagrams for macOS*