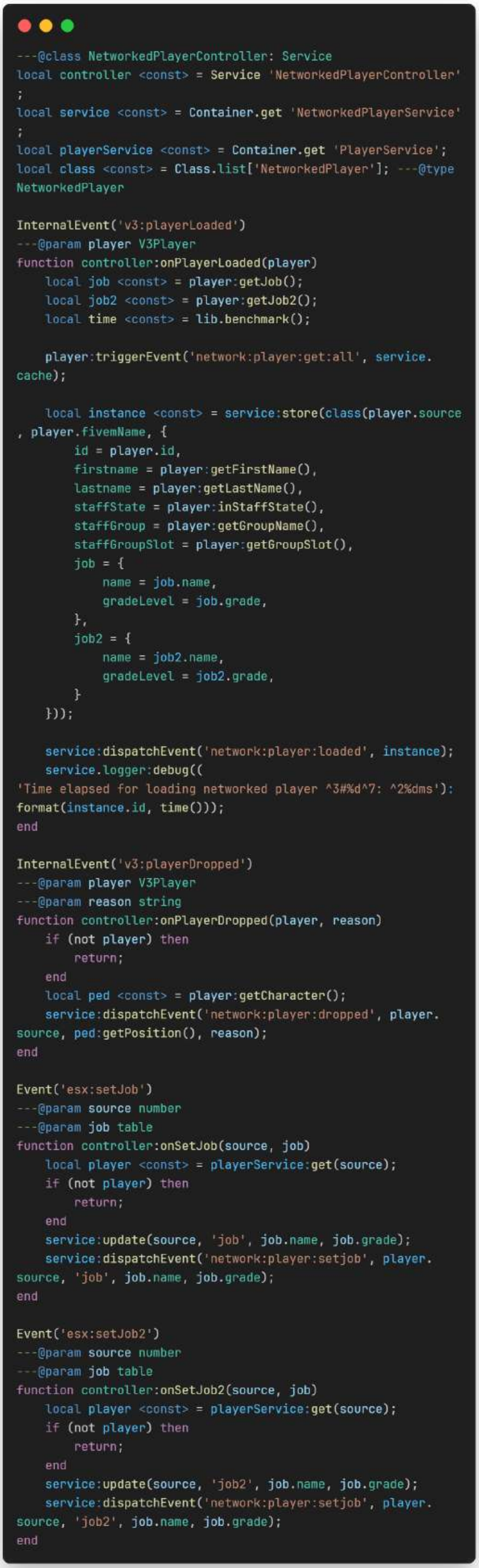
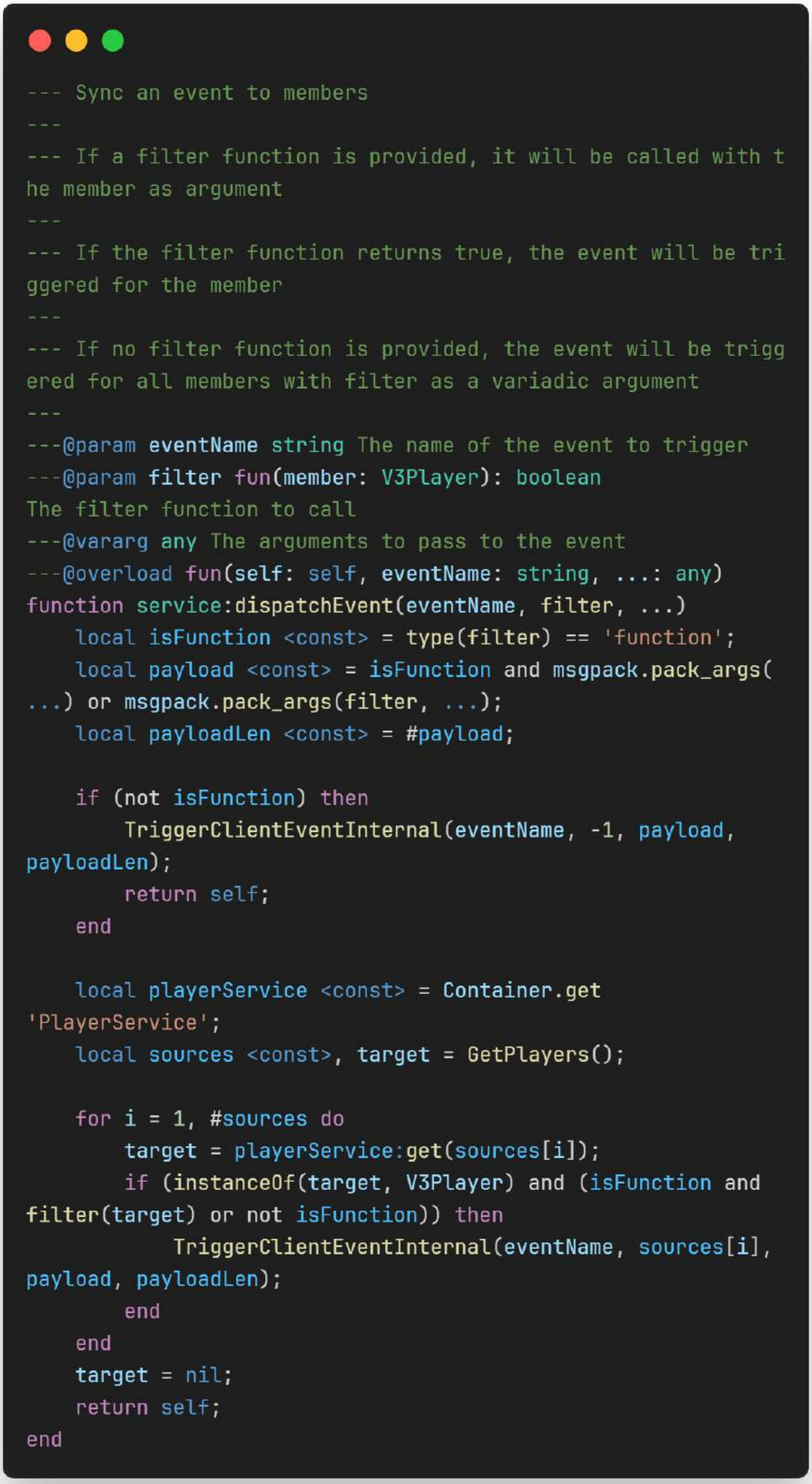
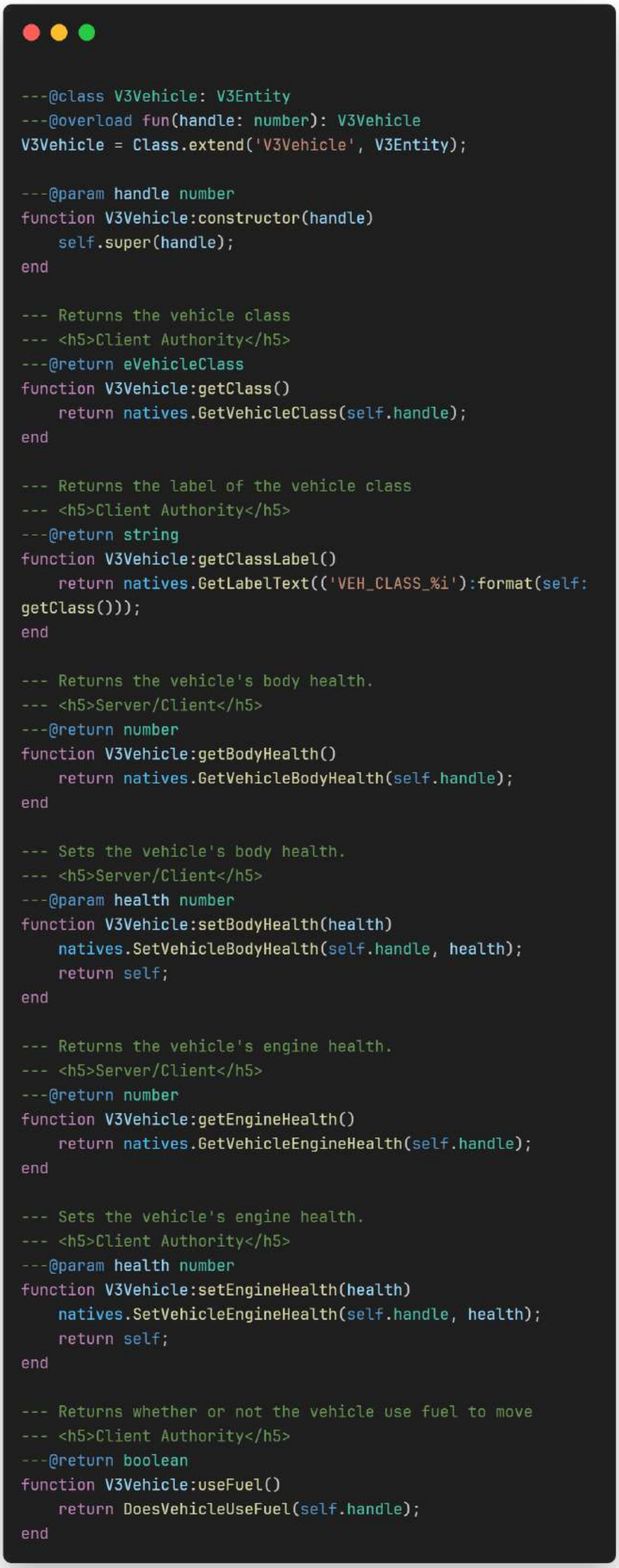


```lua
---@class NetworkedPlayerController: Service
local controller <const> = Service 'NetworkedPlayerController'
;
local service <const> = Container.get 'NetworkedPlayerService'
;
local playerService <const> = Container.get 'PlayerService';
local class <const> = Class.list['NetworkedPlayer']; ---@type
NetworkedPlayer

InternalEvent('v3:playerLoaded')
---@param player V3Player
function controller:onPlayerLoaded(player)
    local job <const> = player:getJob();
    local job2 <const> = player:getJob2();
    local time <const> = lib.benchmark();

    player:triggerEvent('network:player:get:all', service.
cache);

    local instance <const> = service:store(class(player.source
, player.fivemName, {
        id = player.id,
        firstname = player:getFirstName(),
        lastname = player:getLastName(),
        staffState = player:inStaffState(),
        staffGroup = player:getGroupName(),
        staffGroupSlot = player:getGroupSlot(),
        job = {
            name = job.name,
            gradeLevel = job.grade,
        },
        job2 = {
            name = job2.name,
            gradeLevel = job2.grade,
        }
    }));

    service:dispatchEvent('network:player:loaded', instance);
    service.logger:debug((
'Time elapsed for loading networked player ^3#%d^7: ^2%dms'):
format(instance.id, time()));
end

InternalEvent('v3:playerDropped')
---@param player V3Player
---@param reason string
function controller:onPlayerDropped(player, reason)
    if (not player) then
        return;
    end
    local ped <const> = player:getCharacter();
    service:dispatchEvent('network:player:dropped', player.
source, ped:getPosition(), reason);
end

Event('esx:setJob')
---@param source number
---@param job table
function controller:onSetJob(source, job)
    local player <const> = playerService:get(source);
    if (not player) then
        return;
    end
    service:update(source, 'job', job.name, job.grade);
    service:dispatchEvent('network:player:setjob', player.
source, 'job', job.name, job.grade);
end

Event('esx:setJob2')
---@param source number
---@param job table
function controller:onSetJob2(source, job)
    local player <const> = playerService:get(source);
    if (not player) then
        return;
    end
    service:update(source, 'job2', job.name, job.grade);
    service:dispatchEvent('network:player:setjob', player.
source, 'job2', job.name, job.grade);
end
```



```lua
--- Sync an event to members
---
--- If a filter function is provided, it will be called with t
he member as argument
---
--- If the filter function returns true, the event will be tri
ggered for the member
---
--- If no filter function is provided, the event will be trigg
ered for all members with filter as a variadic argument
---
---@param eventName string The name of the event to trigger
---@param filter fun(member: V3Player): boolean
The filter function to call
---@vararg any The arguments to pass to the event
---@overload fun(self: self, eventName: string, ...: any)
function service:dispatchEvent(eventName, filter, ...)
    local isFunction <const> = type(filter) == 'function';
    local payload <const> = isFunction and msgpack.pack_args(
...) or msgpack.pack_args(filter, ...);
    local payloadLen <const> = #payload;

    if (not isFunction) then
        TriggerClientEventInternal(eventName, -1, payload,
payloadLen);
        return self;
    end

    local playerService <const> = Container.get
'PlayerService';
    local sources <const>, target = GetPlayers();

    for i = 1, #sources do
        target = playerService:get(sources[i]);
        if (instanceOf(target, V3Player) and (isFunction and
filter(target) or not isFunction)) then
            TriggerClientEventInternal(eventName, sources[i],
payload, payloadLen);
        end
    end
    target = nil;
    return self;
end
```



```lua
---@class V3Vehicle: V3Entity
---@overload fun(handle: number): V3Vehicle
V3Vehicle = Class.extend('V3Vehicle', V3Entity);

---@param handle number
function V3Vehicle:constructor(handle)
    self.super(handle);
end

--- Returns the vehicle class
--- <h5>Client Authority</h5>
---@return eVehicleClass
function V3Vehicle:getClass()
    return natives.GetVehicleClass(self.handle);
end

--- Returns the label of the vehicle class
--- <h5>Client Authority</h5>
---@return string
function V3Vehicle:getClassLabel()
    return natives.GetLabelText(('VEH_CLASS_%i'):format(self:
getClass()));
end

--- Returns the vehicle's body health.
--- <h5>Server/Client</h5>
---@return number
function V3Vehicle:getBodyHealth()
    return natives.GetVehicleBodyHealth(self.handle);
end

--- Sets the vehicle's body health.
--- <h5>Server/Client</h5>
---@param health number
function V3Vehicle:setBodyHealth(health)
    natives.SetVehicleBodyHealth(self.handle, health);
    return self;
end

--- Returns the vehicle's engine health.
--- <h5>Server/Client</h5>
---@return number
function V3Vehicle:getEngineHealth()
    return natives.GetVehicleEngineHealth(self.handle);
end

--- Sets the vehicle's engine health.
--- <h5>Client Authority</h5>
---@param health number
function V3Vehicle:setEngineHealth(health)
    natives.SetVehicleEngineHealth(self.handle, health);
    return self;
end

--- Returns whether or not the vehicle use fuel to move
--- <h5>Client Authority</h5>
---@return boolean
function V3Vehicle:useFuel()
    return DoesVehicleUseFuel(self.handle);
end
```