Problem 1797: Design Authentication Manager

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

There is an authentication system that works with authentication tokens. For each session, the user will receive a new authentication token that will expire

timeToLive

seconds after the

currentTime

. If the token is renewed, the expiry time will be

extended

to expire

timeToLive

seconds after the (potentially different)

currentTime

.

Implement the

AuthenticationManager

class:

AuthenticationManager(int timeToLive)

constructs the

AuthenticationManager

and sets the

timeToLive

.

generate(string tokenId, int currentTime)

generates a new token with the given

tokenId

at the given

currentTime

in seconds.

renew(string tokenId, int currentTime)

renews the

unexpired

token with the given

tokenId

at the given

currentTime

in seconds. If there are no unexpired tokens with the given

tokenId

, the request is ignored, and nothing happens.

countUnexpiredTokens(int currentTime)

returns the number of

unexpired

tokens at the given currentTime.

Note that if a token expires at time

t

, and another action happens on time

t

(

renew

or

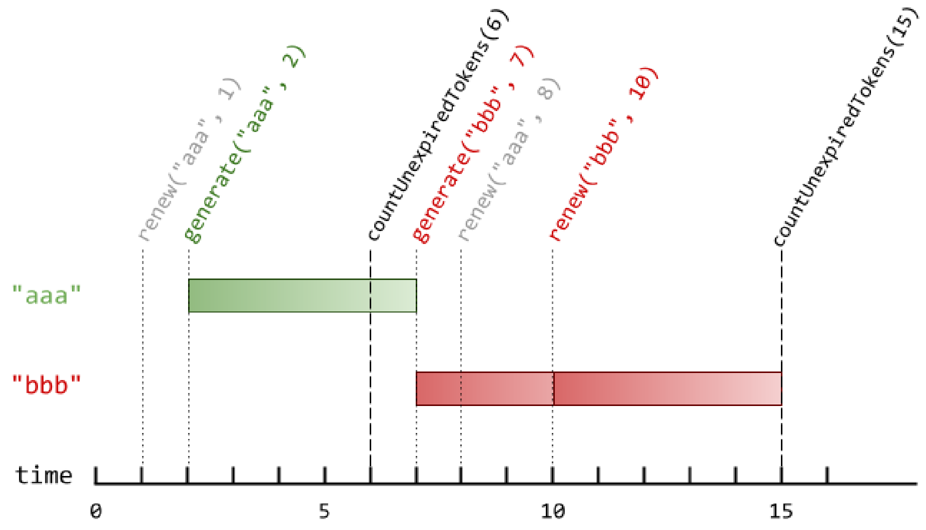
countUnexpiredTokens

), the expiration takes place

before

the other actions.

Example 1:



Input

["AuthenticationManager", "

renew

", "generate", "

countUnexpiredTokens

", "generate", "

renew

", "

renew

", "

countUnexpiredTokens

"] [[5], ["aaa", 1], ["aaa", 2], [6], ["bbb", 7], ["aaa", 8], ["bbb", 10], [15]]

Output

[null, null, null, 1, null, null, null, 0]

Explanation

AuthenticationManager authenticationManager = new AuthenticationManager(5); // Constructs the AuthenticationManager with

timeToLive

= 5 seconds. authenticationManager.

renew

("aaa", 1); // No token exists with tokenId "aaa" at time 1, so nothing happens. authenticationManager.generate("aaa", 2); // Generates a new token with tokenId "aaa" at time 2. authenticationManager.

countUnexpiredTokens

(6); // The token with tokenId "aaa" is the only unexpired one at time 6, so return 1. authenticationManager.generate("bbb", 7); // Generates a new token with tokenId "bbb" at time 7. authenticationManager.

renew

("aaa", 8); // The token with tokenId "aaa" expired at time 7, and 8 >= 7, so at time 8 the

renew

request is ignored, and nothing happens. authenticationManager.

renew

("bbb", 10); // The token with tokenId "bbb" is unexpired at time 10, so the

renew

request is fulfilled and now the token will expire at time 15. authenticationManager.

countUnexpiredTokens

(15); // The token with tokenId "bbb" expires at time 15, and the token with tokenId "aaa" expired at time 7, so currently no token is unexpired, so return 0.

Constraints:

1 <= timeToLive <= 10

8

1 <= currentTime <= 10

8

1 <= tokenId.length <= 5

tokenId

consists only of lowercase letters.

All calls to

generate

will contain unique values of

tokenId

.

The values of

currentTime

across all the function calls will be

strictly increasing

.

At most

2000

calls will be made to all functions combined.

Code Snippets

C++:

```
class AuthenticationManager {
public:
AuthenticationManager(int timeToLive) {

}

void generate(string tokenId, int currentTime) {

}

void renew(string tokenId, int currentTime) {

}

int countUnexpiredTokens(int currentTime) {

}
};

/**
* Your AuthenticationManager object will be instantiated and called as such:
* AuthenticationManager* obj = new AuthenticationManager(timeToLive);
* obj->generate(tokenId,currentTime);
* obj->renew(tokenId,currentTime);
* int param_3 = obj->countUnexpiredTokens(currentTime);
*/
```

Java:

```
class AuthenticationManager {

public AuthenticationManager(int timeToLive) {

}

public void generate(String tokenId, int currentTime) {

}

public void renew(String tokenId, int currentTime) {

}

public int countUnexpiredTokens(int currentTime) {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* AuthenticationManager obj = new AuthenticationManager(timeToLive);
* obj.generate(tokenId,currentTime);
* obj.renew(tokenId,currentTime);
* int param_3 = obj.countUnexpiredTokens(currentTime);
*/
```

Python3:

```
class AuthenticationManager:

def __init__(self, timeToLive: int):



def generate(self, tokenId: str, currentTime: int) -> None:



def renew(self, tokenId: str, currentTime: int) -> None:



def countUnexpiredTokens(self, currentTime: int) -> int:
```

```
# Your AuthenticationManager object will be instantiated and called as such:
# obj = AuthenticationManager(timeToLive)
# obj.generate(tokenId,currentTime)
# obj.renew(tokenId,currentTime)
# param_3 = obj.countUnexpiredTokens(currentTime)
```

Python:

```
class AuthenticationManager(object):

def __init__(self, timeToLive):
"""
:type timeToLive: int
"""



def generate(self, tokenId, currentTime):
"""
:type tokenId: str
:type currentTime: int
:rtype: None
"""



def renew(self, tokenId, currentTime):
"""
:type tokenId: str
:type currentTime: int
:rtype: None
"""



def countUnexpiredTokens(self, currentTime):
"""
:type currentTime: int
:rtype: int
"""




# Your AuthenticationManager object will be instantiated and called as such:
```

```
# obj = AuthenticationManager(timeToLive)
# obj.generate(tokenId,currentTime)
# obj.renew(tokenId,currentTime)
# param_3 = obj.countUnexpiredTokens(currentTime)
```

JavaScript:

```
/**
* @param {number} timeToLive
*/
var AuthenticationManager = function(timeToLive) {

};

/**
* @param {string} tokenId
* @param {number} currentTime
* @return {void}
*/
AuthenticationManager.prototype.generate = function(tokenId, currentTime) {

};

/**
* @param {string} tokenId
* @param {number} currentTime
* @return {void}
*/
AuthenticationManager.prototype.renew = function(tokenId, currentTime) {

};

/**
* @param {number} currentTime
* @return {number}
*/
AuthenticationManager.prototype.countUnexpiredTokens = function(currentTime)
{

};

/**
```

```
* Your AuthenticationManager object will be instantiated and called as such:
* var obj = new AuthenticationManager(timeToLive)
* obj.generate(tokenId,currentTime)
* obj.renew(tokenId,currentTime)
* var param_3 = obj.countUnexpiredTokens(currentTime)
*/
```

TypeScript:

```
class AuthenticationManager {
constructor(timeToLive: number) {

}

generate(tokenId: string, currentTime: number): void {

}

renew(tokenId: string, currentTime: number): void {

}

countUnexpiredTokens(currentTime: number): number {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* var obj = new AuthenticationManager(timeToLive)
* obj.generate(tokenId,currentTime)
* obj.renew(tokenId,currentTime)
* var param_3 = obj.countUnexpiredTokens(currentTime)
*/
```

C#:

```
public class AuthenticationManager {

public AuthenticationManager(int timeToLive) {

}
public void Generate(string tokenId, int currentTime) {

}

public void Renew(string tokenId, int currentTime) {

}

public int CountUnexpiredTokens(int currentTime) {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* AuthenticationManager obj = new AuthenticationManager(timeToLive);
* obj.Generate(tokenId,currentTime);
* obj.Renew(tokenId,currentTime);
* int param_3 = obj.CountUnexpiredTokens(currentTime);
*/
```

C:

```
typedef struct {

} AuthenticationManager;


AuthenticationManager* authenticationManagerCreate(int timeToLive) {

}

void authenticationManagerGenerate(AuthenticationManager* obj, char* tokenId,
int currentTime) {

}

void authenticationManagerRenew(AuthenticationManager* obj, char* tokenId,
int currentTime) {

}

int authenticationManagerCountUnexpiredTokens(AuthenticationManager* obj, int
currentTime) {

}

void authenticationManagerFree(AuthenticationManager* obj) {

}

/**
* Your AuthenticationManager struct will be instantiated and called as such:
* AuthenticationManager* obj = authenticationManagerCreate(timeToLive);
* authenticationManagerGenerate(obj, tokenId, currentTime);

* authenticationManagerRenew(obj, tokenId, currentTime);

* int param_3 = authenticationManagerCountUnexpiredTokens(obj, currentTime);

* authenticationManagerFree(obj);
*/
```

Go:

```
type AuthenticationManager struct {

}


func Constructor(timeToLive int) AuthenticationManager {

}


func (this *AuthenticationManager) Generate(tokenId string, currentTime int)
{

}

```

```
func (this *AuthenticationManager) Renew(tokenId string, currentTime int) {

}


func (this *AuthenticationManager) CountUnexpiredTokens(currentTime int) int
{

}


/**
* Your AuthenticationManager object will be instantiated and called as such:
* obj := Constructor(timeToLive);
* obj.Generate(tokenId,currentTime);
* obj.Renew(tokenId,currentTime);
* param_3 := obj.CountUnexpiredTokens(currentTime);
*/
```

Kotlin:

```
class AuthenticationManager(timeToLive: Int) {

fun generate(tokenId: String, currentTime: Int) {

}

fun renew(tokenId: String, currentTime: Int) {

}

fun countUnexpiredTokens(currentTime: Int): Int {

}

}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* var obj = AuthenticationManager(timeToLive)
* obj.generate(tokenId,currentTime)
* obj.renew(tokenId,currentTime)
* var param_3 = obj.countUnexpiredTokens(currentTime)
*/
```

Swift:

```
class AuthenticationManager {

init(_ timeToLive: Int) {

}

func generate(_ tokenId: String, _ currentTime: Int) {

}

func renew(_ tokenId: String, _ currentTime: Int) {

}

func countUnexpiredTokens(_ currentTime: Int) -> Int {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* let obj = AuthenticationManager(timeToLive)
* obj.generate(tokenId, currentTime)
* obj.renew(tokenId, currentTime)
* let ret_3: Int = obj.countUnexpiredTokens(currentTime)
*/
```

Rust:

```
struct AuthenticationManager {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl AuthenticationManager {

fn new(timeToLive: i32) -> Self {

}

fn generate(&self, token_id: String, current_time: i32) {

}

fn renew(&self, token_id: String, current_time: i32) {

}

fn count_unexpired_tokens(&self, current_time: i32) -> i32 {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* let obj = AuthenticationManager::new(timeToLive);
* obj.generate(tokenId, currentTime);
* obj.renew(tokenId, currentTime);
* let ret_3: i32 = obj.count_unexpired_tokens(currentTime);
*/
```

Ruby:

```
class AuthenticationManager

=begin
:type time_to_live: Integer
=end
def initialize(time_to_live)

end



=begin
```

```
:type token_id: String
:type current_time: Integer
:rtype: Void
=end
def generate(token_id, current_time)

end



=begin
:type token_id: String
:type current_time: Integer
:rtype: Void
=end
def renew(token_id, current_time)

end



=begin
:type current_time: Integer
:rtype: Integer
=end
def count_unexpired_tokens(current_time)

end



end

# Your AuthenticationManager object will be instantiated and called as such:
# obj = AuthenticationManager.new(time_to_live)
# obj.generate(token_id, current_time)
# obj.renew(token_id, current_time)
# param_3 = obj.count_unexpired_tokens(current_time)
```

PHP:

```
class AuthenticationManager {
/**
* @param Integer $timeToLive
*/
function __construct($timeToLive) {

}

/**
* @param String $tokenId
* @param Integer $currentTime
* @return NULL
*/
function generate($tokenId, $currentTime) {

}

/**
* @param String $tokenId
* @param Integer $currentTime
* @return NULL
*/
function renew($tokenId, $currentTime) {

}

/**
* @param Integer $currentTime
* @return Integer
*/
function countUnexpiredTokens($currentTime) {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* $obj = AuthenticationManager($timeToLive);
* $obj->generate($tokenId, $currentTime);
* $obj->renew($tokenId, $currentTime);
* $ret_3 = $obj->countUnexpiredTokens($currentTime);
*/
```

Dart:

```
class AuthenticationManager {

AuthenticationManager(int timeToLive) {

}

void generate(String tokenId, int currentTime) {

}

void renew(String tokenId, int currentTime) {

}

int countUnexpiredTokens(int currentTime) {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* AuthenticationManager obj = AuthenticationManager(timeToLive);
* obj.generate(tokenId,currentTime);
* obj.renew(tokenId,currentTime);
* int param3 = obj.countUnexpiredTokens(currentTime);
*/
```

Scala:

```
class AuthenticationManager(_timeToLive: Int) {

def generate(tokenId: String, currentTime: Int): Unit = {

}

def renew(tokenId: String, currentTime: Int): Unit = {

}

def countUnexpiredTokens(currentTime: Int): Int = {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* val obj = new AuthenticationManager(timeToLive)
* obj.generate(tokenId,currentTime)
* obj.renew(tokenId,currentTime)
* val param_3 = obj.countUnexpiredTokens(currentTime)
*/
```

Elixir:

```
defmodule AuthenticationManager do
@spec init_(time_to_live :: integer) :: any
def init_(time_to_live) do

end

@spec generate(token_id :: String.t, current_time :: integer) :: any
def generate(token_id, current_time) do

end

@spec renew(token_id :: String.t, current_time :: integer) :: any
def renew(token_id, current_time) do

end

@spec count_unexpired_tokens(current_time :: integer) :: integer
def count_unexpired_tokens(current_time) do

end
end

# Your functions will be called as such:
# AuthenticationManager.init_(time_to_live)
# AuthenticationManager.generate(token_id, current_time)
# AuthenticationManager.renew(token_id, current_time)
# param_3 = AuthenticationManager.count_unexpired_tokens(current_time)

# AuthenticationManager.init_ will be called before every test case, in which
you can do some necessary initializations.
```

Erlang:

```
-spec authentication_manager_init_(TimeToLive :: integer()) -> any().
authentication_manager_init_(TimeToLive) ->
.

-spec authentication_manager_generate(TokenId :: unicode:unicode_binary(),
CurrentTime :: integer()) -> any().
authentication_manager_generate(TokenId, CurrentTime) ->
.

-spec authentication_manager_renew(TokenId :: unicode:unicode_binary(),
CurrentTime :: integer()) -> any().
authentication_manager_renew(TokenId, CurrentTime) ->
.

-spec authentication_manager_count_unexpired_tokens(CurrentTime :: integer())
-> integer().
authentication_manager_count_unexpired_tokens(CurrentTime) ->
.



%% Your functions will be called as such:
%% authentication_manager_init_(TimeToLive),
%% authentication_manager_generate(TokenId, CurrentTime),
%% authentication_manager_renew(TokenId, CurrentTime),
%% Param_3 = authentication_manager_count_unexpired_tokens(CurrentTime),

%% authentication_manager_init_ will be called before every test case, in
which you can do some necessary initializations.
```

Racket:

```
(define authentication-manager%
(class object%
(super-new)

; time-to-live : exact-integer?
(init-field
time-to-live)

; generate : string? exact-integer? -> void?
(define/public (generate token-id current-time)
)
; renew : string? exact-integer? -> void?
(define/public (renew token-id current-time)
)
; count-unexpired-tokens : exact-integer? -> exact-integer?
(define/public (count-unexpired-tokens current-time)
)))

;; Your authentication-manager% object will be instantiated and called as
such:
;; (define obj (new authentication-manager% [time-to-live time-to-live]))
;; (send obj generate token-id current-time)
;; (send obj renew token-id current-time)
;; (define param_3 (send obj count-unexpired-tokens current-time))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Authentication Manager
* Difficulty: Medium
* Tags: string, hash, linked_list
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class AuthenticationManager {
public:
AuthenticationManager(int timeToLive) {

}

void generate(string tokenId, int currentTime) {

}
void renew(string tokenId, int currentTime) {

}

int countUnexpiredTokens(int currentTime) {

}
};

/**
* Your AuthenticationManager object will be instantiated and called as such:
* AuthenticationManager* obj = new AuthenticationManager(timeToLive);
* obj->generate(tokenId,currentTime);
* obj->renew(tokenId,currentTime);
* int param_3 = obj->countUnexpiredTokens(currentTime);
*/
```

Java Solution:

```
/**
* Problem: Design Authentication Manager
* Difficulty: Medium
* Tags: string, hash, linked_list
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class AuthenticationManager {

public AuthenticationManager(int timeToLive) {

}

public void generate(String tokenId, int currentTime) {

}

public void renew(String tokenId, int currentTime) {
}

public int countUnexpiredTokens(int currentTime) {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* AuthenticationManager obj = new AuthenticationManager(timeToLive);
* obj.generate(tokenId,currentTime);
* obj.renew(tokenId,currentTime);
* int param_3 = obj.countUnexpiredTokens(currentTime);
*/
```

Python3 Solution:

```
"""
Problem: Design Authentication Manager
Difficulty: Medium
Tags: string, hash, linked_list

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class AuthenticationManager:

def __init__(self, timeToLive: int):



def generate(self, tokenId: str, currentTime: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class AuthenticationManager(object):

def __init__(self, timeToLive):
```

```
"""
:type timeToLive: int
"""



def generate(self, tokenId, currentTime):
"""
:type tokenId: str
:type currentTime: int
:rtype: None
"""



def renew(self, tokenId, currentTime):
"""
:type tokenId: str
:type currentTime: int
:rtype: None
"""



def countUnexpiredTokens(self, currentTime):
"""
:type currentTime: int
:rtype: int
"""




# Your AuthenticationManager object will be instantiated and called as such:
# obj = AuthenticationManager(timeToLive)
# obj.generate(tokenId,currentTime)
# obj.renew(tokenId,currentTime)
# param_3 = obj.countUnexpiredTokens(currentTime)
```

JavaScript Solution:

```
/**
* Problem: Design Authentication Manager
* Difficulty: Medium
* Tags: string, hash, linked_list
```

```
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} timeToLive
*/
var AuthenticationManager = function(timeToLive) {

};

/**
* @param {string} tokenId
* @param {number} currentTime
* @return {void}
*/
AuthenticationManager.prototype.generate = function(tokenId, currentTime) {

};

/**
* @param {string} tokenId
* @param {number} currentTime
* @return {void}
*/
AuthenticationManager.prototype.renew = function(tokenId, currentTime) {

};

/**
* @param {number} currentTime
* @return {number}
*/
AuthenticationManager.prototype.countUnexpiredTokens = function(currentTime)
{

};

/**
* Your AuthenticationManager object will be instantiated and called as such:
```

```
* var obj = new AuthenticationManager(timeToLive)
* obj.generate(tokenId,currentTime)
* obj.renew(tokenId,currentTime)
* var param_3 = obj.countUnexpiredTokens(currentTime)
*/
```

TypeScript Solution:

```
/**
* Problem: Design Authentication Manager
* Difficulty: Medium
* Tags: string, hash, linked_list
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class AuthenticationManager {
constructor(timeToLive: number) {

}

generate(tokenId: string, currentTime: number): void {

}

renew(tokenId: string, currentTime: number): void {

}

countUnexpiredTokens(currentTime: number): number {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* var obj = new AuthenticationManager(timeToLive)
* obj.generate(tokenId,currentTime)
* obj.renew(tokenId,currentTime)
* var param_3 = obj.countUnexpiredTokens(currentTime)
*/
```

C# Solution:

```
/*
* Problem: Design Authentication Manager
* Difficulty: Medium
* Tags: string, hash, linked_list
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class AuthenticationManager {

public AuthenticationManager(int timeToLive) {

}

public void Generate(string tokenId, int currentTime) {

}

public void Renew(string tokenId, int currentTime) {

}

public int CountUnexpiredTokens(int currentTime) {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* AuthenticationManager obj = new AuthenticationManager(timeToLive);
* obj.Generate(tokenId,currentTime);
* obj.Renew(tokenId,currentTime);
* int param_3 = obj.CountUnexpiredTokens(currentTime);
*/
```

C Solution:

```c
/*
* Problem: Design Authentication Manager
* Difficulty: Medium
* Tags: string, hash, linked_list
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/




typedef struct {

} AuthenticationManager;


AuthenticationManager* authenticationManagerCreate(int timeToLive) {

}

void authenticationManagerGenerate(AuthenticationManager* obj, char* tokenId,
int currentTime) {

}

void authenticationManagerRenew(AuthenticationManager* obj, char* tokenId,
int currentTime) {

}

int authenticationManagerCountUnexpiredTokens(AuthenticationManager* obj, int
currentTime) {

}

void authenticationManagerFree(AuthenticationManager* obj) {

}
```

```
/**
* Your AuthenticationManager struct will be instantiated and called as such:
* AuthenticationManager* obj = authenticationManagerCreate(timeToLive);
* authenticationManagerGenerate(obj, tokenId, currentTime);

* authenticationManagerRenew(obj, tokenId, currentTime);

* int param_3 = authenticationManagerCountUnexpiredTokens(obj, currentTime);

* authenticationManagerFree(obj);
*/
```

Go Solution:

```
// Problem: Design Authentication Manager
// Difficulty: Medium
// Tags: string, hash, linked_list
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type AuthenticationManager struct {

}


func Constructor(timeToLive int) AuthenticationManager {

}


func (this *AuthenticationManager) Generate(tokenId string, currentTime int)
{

}


func (this *AuthenticationManager) Renew(tokenId string, currentTime int) {
}


func (this *AuthenticationManager) CountUnexpiredTokens(currentTime int) int
{

}


/**
* Your AuthenticationManager object will be instantiated and called as such:
* obj := Constructor(timeToLive);
* obj.Generate(tokenId,currentTime);
* obj.Renew(tokenId,currentTime);
* param_3 := obj.CountUnexpiredTokens(currentTime);
*/
```

Kotlin Solution:

```
class AuthenticationManager(timeToLive: Int) {

fun generate(tokenId: String, currentTime: Int) {

}

fun renew(tokenId: String, currentTime: Int) {

}

fun countUnexpiredTokens(currentTime: Int): Int {

}

}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* var obj = AuthenticationManager(timeToLive)
* obj.generate(tokenId,currentTime)
* obj.renew(tokenId,currentTime)
* var param_3 = obj.countUnexpiredTokens(currentTime)
```

```
*/
```

Swift Solution:

```
class AuthenticationManager {

init(_ timeToLive: Int) {

}

func generate(_ tokenId: String, _ currentTime: Int) {

}

func renew(_ tokenId: String, _ currentTime: Int) {

}

func countUnexpiredTokens(_ currentTime: Int) -> Int {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* let obj = AuthenticationManager(timeToLive)
* obj.generate(tokenId, currentTime)
* obj.renew(tokenId, currentTime)
* let ret_3: Int = obj.countUnexpiredTokens(currentTime)
*/
```

Rust Solution:

```
// Problem: Design Authentication Manager
// Difficulty: Medium
// Tags: string, hash, linked_list
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
struct AuthenticationManager {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl AuthenticationManager {

fn new(timeToLive: i32) -> Self {

}

fn generate(&self, token_id: String, current_time: i32) {

}

fn renew(&self, token_id: String, current_time: i32) {

}

fn count_unexpired_tokens(&self, current_time: i32) -> i32 {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* let obj = AuthenticationManager::new(timeToLive);
* obj.generate(tokenId, currentTime);
* obj.renew(tokenId, currentTime);
* let ret_3: i32 = obj.count_unexpired_tokens(currentTime);
*/
```

Ruby Solution:

```
class AuthenticationManager
```

```
=begin
:type time_to_live: Integer
=end
def initialize(time_to_live)

end



=begin
:type token_id: String
:type current_time: Integer
:rtype: Void
=end
def generate(token_id, current_time)

end



=begin
:type token_id: String
:type current_time: Integer
:rtype: Void
=end
def renew(token_id, current_time)

end



=begin
:type current_time: Integer
:rtype: Integer
=end
def count_unexpired_tokens(current_time)

end



end

# Your AuthenticationManager object will be instantiated and called as such:
# obj = AuthenticationManager.new(time_to_live)
# obj.generate(token_id, current_time)
# obj.renew(token_id, current_time)
# param_3 = obj.count_unexpired_tokens(current_time)
```

PHP Solution:

```
class AuthenticationManager {
/**
* @param Integer $timeToLive
*/
function __construct($timeToLive) {

}

/**
* @param String $tokenId
* @param Integer $currentTime
* @return NULL
*/
function generate($tokenId, $currentTime) {

}

/**
* @param String $tokenId
* @param Integer $currentTime
* @return NULL
*/
function renew($tokenId, $currentTime) {

}

/**
* @param Integer $currentTime
* @return Integer
*/
function countUnexpiredTokens($currentTime) {

}
}

/**
```

```
* Your AuthenticationManager object will be instantiated and called as such:
* $obj = AuthenticationManager($timeToLive);
* $obj->generate($tokenId, $currentTime);
* $obj->renew($tokenId, $currentTime);
* $ret_3 = $obj->countUnexpiredTokens($currentTime);
*/
```

Dart Solution:

```
class AuthenticationManager {

AuthenticationManager(int timeToLive) {

}

void generate(String tokenId, int currentTime) {

}

void renew(String tokenId, int currentTime) {

}

int countUnexpiredTokens(int currentTime) {

}
}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* AuthenticationManager obj = AuthenticationManager(timeToLive);
* obj.generate(tokenId,currentTime);
* obj.renew(tokenId,currentTime);
* int param3 = obj.countUnexpiredTokens(currentTime);
*/
```

Scala Solution:

```
class AuthenticationManager(_timeToLive: Int) {

def generate(tokenId: String, currentTime: Int): Unit = {
}

def renew(tokenId: String, currentTime: Int): Unit = {

}

def countUnexpiredTokens(currentTime: Int): Int = {

}

}

/**
* Your AuthenticationManager object will be instantiated and called as such:
* val obj = new AuthenticationManager(timeToLive)
* obj.generate(tokenId,currentTime)
* obj.renew(tokenId,currentTime)
* val param_3 = obj.countUnexpiredTokens(currentTime)
*/
```

Elixir Solution:

```
defmodule AuthenticationManager do
@spec init_(time_to_live :: integer) :: any
def init_(time_to_live) do

end

@spec generate(token_id :: String.t, current_time :: integer) :: any
def generate(token_id, current_time) do

end

@spec renew(token_id :: String.t, current_time :: integer) :: any
def renew(token_id, current_time) do

end

@spec count_unexpired_tokens(current_time :: integer) :: integer
def count_unexpired_tokens(current_time) do
```

```
end
end

# Your functions will be called as such:
# AuthenticationManager.init_(time_to_live)
# AuthenticationManager.generate(token_id, current_time)
# AuthenticationManager.renew(token_id, current_time)
# param_3 = AuthenticationManager.count_unexpired_tokens(current_time)

# AuthenticationManager.init_ will be called before every test case, in which
you can do some necessary initializations.
```

Erlang Solution:

```
-spec authentication_manager_init_(TimeToLive :: integer()) -> any().
authentication_manager_init_(TimeToLive) ->
.

-spec authentication_manager_generate(TokenId :: unicode:unicode_binary(),
CurrentTime :: integer()) -> any().
authentication_manager_generate(TokenId, CurrentTime) ->
.

-spec authentication_manager_renew(TokenId :: unicode:unicode_binary(),
CurrentTime :: integer()) -> any().
authentication_manager_renew(TokenId, CurrentTime) ->
.

-spec authentication_manager_count_unexpired_tokens(CurrentTime :: integer())
-> integer().
authentication_manager_count_unexpired_tokens(CurrentTime) ->
.


%% Your functions will be called as such:
%% authentication_manager_init_(TimeToLive),
%% authentication_manager_generate(TokenId, CurrentTime),
%% authentication_manager_renew(TokenId, CurrentTime),
%% Param_3 = authentication_manager_count_unexpired_tokens(CurrentTime),
%% authentication_manager_init_ will be called before every test case, in
which you can do some necessary initializations.
```

Racket Solution:

```
(define authentication-manager%
(class object%
(super-new)

; time-to-live : exact-integer?
(init-field
time-to-live)

; generate : string? exact-integer? -> void?
(define/public (generate token-id current-time)
)
; renew : string? exact-integer? -> void?
(define/public (renew token-id current-time)
)
; count-unexpired-tokens : exact-integer? -> exact-integer?
(define/public (count-unexpired-tokens current-time)
)))

;; Your authentication-manager% object will be instantiated and called as
such:
;; (define obj (new authentication-manager% [time-to-live time-to-live]))
;; (send obj generate token-id current-time)
;; (send obj renew token-id current-time)
;; (define param_3 (send obj count-unexpired-tokens current-time))
```