```
─────────────── MODULE storagecleanernaive ───────────────
EXTENDS Naturals, Sequences, FiniteSets

CONSTANTS
    USERIDS,
    SERVERS,
    METADATAS,
    IMAGES,
    UUIDS,
     Added to
    CLEANERS

VARIABLES
    databaseState,
    blobStoreState,
    serverStates,
    cleanerStates,   CleanerStates[storageCleanerId]

    operations


vars ≜ ⟨databaseState, blobStoreState,
              serverStates, operations, cleanerStates⟩

cleanerVars ≜ ⟨cleanerStates⟩

Strong Typing

UserIdVal ≜ USERIDS ∪ {"UNSET"}
MetadataVal ≜ METADATAS ∪ {"UNSET"}
ImageVal ≜ IMAGES ∪ {"UNSET"}
UUIDVal ≜ UUIDS ∪ {"UNSET"}  added UUID type

DatabaseRecord ≜ [
    metadata : MetadataVal,
    imageId : UUIDVal
]

Describes all possible states a server can be in. Unchanged since last example)
ServerStateVal ≜
    [
        state : {
             current:
            "waiting",   next: ServerStartWrite or ServerStartRead
             after: ServerStartWrite
            "started_write",   next: ServerWriteBlob or ServerFailWrite
             after: ServerWriteBlob
```

"wrote_blob", next: *ServerWriteMetadataAndReturn* or *ServerFailWrite*

after: *ServerStartRead*

"started_read", next: *ServerReadMetadata*

after: *ServerReadMetadata*, *ServerReadMetadataAndReturnEmpty*

"read_metadata" next: *ServerReadBlobAndReturn*

},

$userId : UserIdVal$,

$metadata : MetadataVal$,

$imageId : UUIDVal$, Need to track *imageId* to perform a lookup

$image : ImageVal$

]

Describes all possible states a server can be in. Unchanged since last example)

$CleanerStateVal \triangleq$

[

$state$: {

current:

"waiting", next: *CleanerStartGetBlobKeys*

after: waiting

"got_blob_keys", next: *CleanerGetUnusedKeys* or *CleanerFail*

after: *got_blob_keys*

"got_unused_keys", next: *CleanerDeleteKeys* or *CleanerFail*

after: *got_unused_keys*

next: *CleanerDeleteKeys*, *CleanerFinished*, or waiting

"deleting_keys"

},

$blobKeys : \textsc{subset}\ UUIDS$,

$unusedBlobKeys : \textsc{subset}\ UUIDS$

]

This is an observability value, and we are still measuring the same thing

No changes are needed

$OperationValue \triangleq [type : \{\text{"READ"}, \text{"WRITE"}\},$
$userId : UserIdVal,$
$metadata : MetadataVal,$
$image : ImageVal]$

$TypeOk \triangleq$

$\land\ databaseState \in [USERIDS \rightarrow DatabaseRecord]$

$\land\ blobStoreState \in [UUIDS \rightarrow ImageVal]$

$\land\ serverStates \in [SERVERS \rightarrow ServerStateVal]$

Added cleaner states to track status of cleaners

$\land\ cleanerStates \in [CLEANERS \rightarrow CleanerStateVal]$

$\land\ operations \in Seq(OperationValue)$

```
Init ≜
    ∧ databaseState =
            [u ∈ USERIDS ↦ [metadata ↦ "UNSET", imageId ↦ "UNSET"]]
    ∧ blobStoreState = [u ∈ UUIDS  ↦  "UNSET"]
    ∧ serverStates = [s ∈ SERVERS ↦  [state ↦ "waiting",
                                       userId ↦ "UNSET",
                                       metadata ↦ "UNSET",
                                       imageId ↦ "UNSET",
                                       image ↦ "UNSET"
                                      ]]
    ∧ cleanerStates = [c ∈ CLEANERS ↦ [
                                           state ↦ "waiting",
                                           blobKeys ↦ {},
                                           unusedBlobKeys ↦ {}
                                         ]]
    ∧ operations = ⟨⟩
```

State Machine: All of the states are functions of s (server), because the only actively modelled actors in this system are our servers, but there can be multiple working simultainiously.

Server Writes

```
ServerStartWrite(s) ≜
    ∧ serverStates[s].state = "waiting"
    ∧ ∃ u ∈ USERIDS, m ∈ METADATAS, i ∈ IMAGES :
        ∧ serverStates′ = [serverStates EXCEPT
                                  ![s].state = "started_write",
                                  ![s].userId = u,
                                  ![s].metadata = m,
                                  ![s].image = i]
        ∧ operations′ = Append(operations,
                                  [
                                      type ↦ "WRITE",
                                      userId ↦ u,
                                      metadata ↦ m,
                                      image ↦ i
                                  ])
```

Cleaner state needs to be added as unchanged for all server operations

```
    ∧ UNCHANGED ⟨databaseState, blobStoreState, cleanerStates⟩


ServerWriteBlob(s) ≜
    LET currentState ≜ serverStates[s]
    IN
    ∧ currentState.state = "started_write"
    ∧ ∃ id ∈ UUIDS :
        $\land\ blobStoreState[id] = \text{“UNSET”}$
        $\land\ blobStoreState' = [blobStoreState\ \textsc{except}$
                                 $![id] = currentState.image]$
        $\land\ serverStates' = [serverStates\ \textsc{except}$
                                 $![s].state = \text{“wrote\_blob”},$
                                 $![s].imageId = id]$
    $\land\ \textsc{unchanged}\ \langle databaseState,\ operations \rangle$
    $\land\ \textsc{unchanged}\ cleanerVars$
```

```
$ServerWriteMetadataAndReturn(s)\ \triangleq$
    $\textsc{let}\ currentState\ \triangleq\ serverStates[s]$
    $\textsc{in}$
        $\land\ currentState.state = \text{“wrote\_blob”}$
        $\land\ databaseState' = [databaseState\ \textsc{except}$
                                 $![currentState.userId] = [$
                                     $metadata \mapsto currentState.metadata,$
                                     $imageId \mapsto currentState.imageId]]$

        $\land\ serverStates' = [serverStates\ \textsc{except}$
                                 $![s].state = \text{“waiting”},$
                                 $![s].userId = \text{“UNSET”},$
                                 $![s].metadata = \text{“UNSET”},$
                                 $![s].image = \text{“UNSET”},$
                                 $![s].imageId = \text{“UNSET”}]$
        $\land\ \textsc{unchanged}\ \langle blobStoreState,\ operations \rangle$
        $\land\ \textsc{unchanged}\ cleanerVars$
```

```
$ServerFailWrite(s)\ \triangleq$
    $\land\ serverStates[s].state \in \{\text{“started\_write”},\ \text{“wrote\_blob”}\}$
    $\land\ serverStates' = [serverStates\ \textsc{except}$
                             $![s].state = \text{“waiting”},$
                             $![s].userId = \text{“UNSET”},$
                             $![s].metadata = \text{“UNSET”},$
                             $![s].image = \text{“UNSET”},$
                             $![s].imageId = \text{“UNSET”}]$
    $\land\ \textsc{unchanged}\ \langle databaseState,\ blobStoreState,\ operations \rangle$
    $\land\ \textsc{unchanged}\ cleanerVars$
```

Server Reads

```
$ServerStartRead(s)\ \triangleq$
    $\land\ serverStates[s].state = \text{“waiting”}$
    $\land\ \exists\, u \in USERIDS :$
            $serverStates' = [serverStates\ \textsc{except}$
                              $![s].state = \text{“started\_read”},$
                              ![s].userId = u]

    ∧ UNCHANGED ⟨databaseState, blobStoreState⟩
    ∧ UNCHANGED operations
    ∧ UNCHANGED cleanerVars

If database record is present
ServerReadMetadata(s) ≜
   LET currentState ≜ serverStates[s]
   IN
    ∧ currentState.state = "started_read"
     Represents reading the metadata while the database record is set
    ∧ databaseState[currentState.userId].metadata ≠ "UNSET"
    ∧ serverStates′ =
        [serverStates EXCEPT
            ![s].state = "read_metadata",
            ![s].metadata = databaseState[currentState.userId].metadata,
             Reads imageId from database
            ![s].imageId = databaseState[currentState.userId].imageId]
    ∧ UNCHANGED ⟨databaseState, blobStoreState⟩
    ∧ UNCHANGED operations
    ∧ UNCHANGED cleanerVars

If database record is not present
ServerReadMetadataAndReturnEmpty(s) ≜
   LET currentState ≜ serverStates[s]
   IN
    ∧ currentState.state = "started_read"
     Represents reading the metadata while the database record is unset
    ∧ databaseState[currentState.userId].metadata = "UNSET"
    ∧ serverStates′ = [serverStates EXCEPT
                              ![s].state = "waiting",
                              ![s].userId = "UNSET",
                              ![s].metadata = "UNSET",
                              ![s].image = "UNSET",
                              ![s].imageId = "UNSET"]

    ∧ operations′ = Append(operations,
                             Returns an empty record
                             [
                                type ↦ "READ",
                                userId ↦ currentState.userId,
                                metadata ↦ "UNSET",
                                image ↦ "UNSET"
                             ])
    ∧ UNCHANGED ⟨databaseState, blobStoreState⟩
```

```
     ∧ UNCHANGED cleanerVars

ServerReadBlobAndReturn(s) ≜
    LET currentState ≜ serverStates[s]
    IN
     ∧ currentState.state = "read_metadata"
     ∧ operations′ = Append(operations,
                            [
                               type ↦ "READ",
                               userId ↦ currentState.userId,
                               metadata ↦ currentState.metadata,
                               image ↦ blobStoreState[currentState.imageId]
                            ])
     ∧ serverStates′ = [serverStates EXCEPT
                        ![s].state = "waiting",
                        ![s].userId = "UNSET",
                        ![s].metadata = "UNSET",
                        ![s].image = "UNSET",
                        ![s].imageId = "UNSET"]
     ∧ UNCHANGED ⟨databaseState, blobStoreState⟩
     ∧ UNCHANGED cleanerVars

Cleaner States

CleanerStartGetBlobKeys(c) ≜
    LET current ≜ cleanerStates[c]IN
     Starts only from waiting
     ∧ current.state = "waiting"
     ∧ cleanerStates′ = [
        cleanerStates EXCEPT
            ![c].state = "got_blob_keys",
             All keys that are set in blockstore
            ![c].blobKeys = {k ∈ UUIDS : blobStoreState[k] ≠ "UNSET"}
        ]
     ∧ UNCHANGED ⟨serverStates, databaseState, blobStoreState, operations⟩

CleanerGetUnusedKeys(c) ≜
    LET current ≜ cleanerStates[c]IN
     From blob keys, get unused keys from database
     ∧ current.state = "got_blob_keys"
     ∧ cleanerStates′ = [
        cleanerStates EXCEPT
            ![c].state = "got_unused_keys",
            ![c].unusedBlobKeys =
                {k ∈ current.blobKeys :  Keys in blob keys
                    ∀ u ∈ USERIDS :  That are not in the database
                    databaseState[u].imageId ≠ k}
        ]
    ∧ UNCHANGED ⟨serverStates, databaseState, blobStoreState, operations⟩

CleanerDeletingKeys(c) ≜
    LET current ≜ cleanerStates[c]IN
    When we have unused keys, keep deleting
    ∧ current.state ∈ {"got_unused_keys", "deleting_keys"}
    ∧ Cardinality(current.unusedBlobKeys) ≠ 0
    ∧ ∃ k ∈ current.unusedBlobKeys :  pick a key to delete
        ∧ blobStoreState′ = [blobStoreState EXCEPT ![k] = "UNSET"]
        ∧ cleanerStates′ = [
            cleanerStates EXCEPT
                remove the key from set
                ![c].unusedBlobKeys = current.unusedBlobKeys \ {k}
            ]
    ∧ UNCHANGED ⟨serverStates, databaseState, operations⟩

CleanerFinished(c) ≜
    LET current ≜ cleanerStates[c]IN
    When we have no more unused keys to delete, finish
    ∧ current.state = "deleting_keys"
    ∧ Cardinality(current.unusedBlobKeys) = 0
    ∧ cleanerStates′ = [
        cleanerStates EXCEPT
            ![c].state = "waiting",
            ![c].blobKeys = {},
            ![c].unusedBlobKeys = {}
        ]
    ∧ UNCHANGED ⟨serverStates, databaseState, blobStoreState, operations⟩

CleanerFail(c) ≜
    LET current ≜ cleanerStates[c]IN
    Cleaner can fail from any active state
    ∧ current.state ∈ {"got_blob_keys", "got_unused_keys", "deleting_keys"}
    Failure represented by cleaner losing state. Any partial operations stay partially finished.
    ∧ cleanerStates′ = [
        cleanerStates EXCEPT
            ![c].state = "waiting",
            ![c].blobKeys = {},
            ![c].unusedBlobKeys = {}
        ]
    ∧ UNCHANGED ⟨serverStates, databaseState, blobStoreState, operations⟩
```

Specification / *Next*

```
$Next \triangleq$
    For every step, we either trigger a server or cleaner to take a step
    $\lor\ \exists s \in SERVERS :$
            $\lor\ ServerStartWrite(s)$
            $\lor\ ServerWriteBlob(s)$
            $\lor\ ServerWriteMetadataAndReturn(s)$
            $\lor\ ServerFailWrite(s)$
            $\lor\ ServerStartRead(s)$
            $\lor\ ServerReadMetadata(s)$
            $\lor\ ServerReadMetadataAndReturnEmpty(s)$
            $\lor\ ServerReadBlobAndReturn(s)$
    $\lor\ \exists c \in CLEANERS :$  all the steps a cleaner can take
           $\lor\ CleanerStartGetBlobKeys(c)$
           $\lor\ CleanerGetUnusedKeys(c)$
           $\lor\ CleanerDeletingKeys(c)$
           $\lor\ CleanerFinished(c)$
           $\lor\ CleanerFail(c)$
```

```
$Spec \triangleq Init \land \Box[Next]_{vars}$
```

Invariants

Note that the success criteria hasn't changed this whole time

```
$ConsistentReads \triangleq$
    If there are no operations, they are consistent
    $\lor\ operations = \langle\rangle$
    $\lor\ \forall i \in 1 \mathrel{..} Len(operations) :$  For every read operation
      LET $readOp \triangleq operations[i]$ IN
      $\lor$    $\land\ readOp.type = \text{"READ"}$
                There must exists a write operation
            $\land$   $\lor\ \exists j \in 1 \mathrel{..} i :$
                         LET $writeOp \triangleq operations[j]$ IN
                         $\land\ writeOp.type = \text{"WRITE"}$
                         With the same data
                         $\land\ readOp.userId = writeOp.userId$
                         $\land\ readOp.metadata = writeOp.metadata$
                         $\land\ readOp.image = writeOp.image$
                  $\lor$  Ignore unset reads
                         $\land\ readOp.metadata = \text{"UNSET"}$
                         $\land\ readOp.image = \text{"UNSET"}$
      $\lor\ readOp.type = \text{"WRITE"}$  Ignore writes
```

```
$NoOrphanFiles \triangleq$
```

```
    There does not exist a key
    $\neg \exists\, k \in \mathit{UUIDS} :$
        That is in the block store
        $\land\, \mathit{blobStoreState}[k] \neq$ “UNSET”
        And not in database
        $\land\, \forall\, u \in \mathit{USERIDS} :$
            $\mathit{databaseState}[u].\mathit{imageId} \neq k$
```

```
At some point in the future there will be no orphan files
If it’s true ever, it is True
$\mathit{EventuallyNoOrphanFiles} \triangleq \Diamond \mathit{NoOrphanFiles}$

Always, at some point in the future, there will be no orphan files
This is how we test eventual consistency. It can’t just happen once
It must always happen
$\mathit{AlwaysEventuallyNoOrphanFiles} \triangleq \Box \mathit{EventuallyNoOrphanFiles}$

$\mathit{StopAfter3Operations} \triangleq$
    $\mathit{Len}(\mathit{operations}) \leq 3$

$\mathit{StopAfter5Operations} \triangleq$
    $\mathit{Len}(\mathit{operations}) \leq 5$
```