MODULE *working*

EXTENDS *Naturals*, *Sequences*

CONSTANTS
USERIDS,
SERVERS,
METADATAS,
IMAGES,
This constant is added to allow us to assign *UUIDs* as blob store keys
UUIDS

VARIABLES
databaseState,
blobStoreState,
serverStates,

operations

$vars \triangleq \langle databaseState, blobStoreState, serverStates, operations \rangle$

Strong Typing

$UserIdVal \triangleq USERIDS \cup \{\text{“UNSET”}\}$
$MetadataVal \triangleq METADATAS \cup \{\text{“UNSET”}\}$
$ImageVal \triangleq IMAGES \cup \{\text{“UNSET”}\}$
$UUIDVal \triangleq UUIDS \cup \{\text{“UNSET”}\}$ added *UUID* type

Describes a database record. We need this now that it has to keep track of which image *UUID* it is associated with.

$DatabaseRecord \triangleq [$
$metadata : MetadataVal,$
$imageId : UUIDVal$
$]$

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

$ServerStateVal \triangleq$
[
state : {
current:
“waiting”, next: *StartWrite* or *StartRead*
after: *StartWrite*
“started_write”, next: *WriteBlob* or *FailWrite*
after: *WriteBlob*
“wrote_blob”, next: *WriteMetadataAndReturn* or *FailWrite*
after: *StartRead*

```
            "started_read",    next: ReadMetadata
             after: ReadMetadata, ReadMetadataAndReturnEmpty
            "read_metadata"  next: ReadBlobAndReturn
        },
        userId : UserIdVal,
        metadata :  MetadataVal,
        imageId : UUIDVal,   Need to track imageId to perform a lookup
        image : ImageVal
    ]

 This is an observability value, and we are still measuring the same thing
 No changes are needed
OperationValue ≜ [type : {"READ", "WRITE"},
                  userId : UserIdVal,
                  metadata : MetadataVal,
                  image : ImageVal]


TypeOk ≜
    Database state modified to hold database records
    ∧ databaseState ∈ [USERIDS → DatabaseRecord]
    Blob store uses UUIDs as keys rather than userIds
    ∧ blobStoreState ∈ [UUIDS  → ImageVal]
    ∧ serverStates ∈ [SERVERS → ServerStateVal]
    ∧ operations ∈ Seq(OperationValue)


Init ≜
    Database record needs to be initialized differently
    ∧ databaseState =
          [u ∈ USERIDS ↦ [metadata ↦ "UNSET", imageId ↦ "UNSET"]]
    Blob store is initialized with UUIDS
    ∧ blobStoreState = [u ∈ UUIDS  ↦  "UNSET"]
    ∧ serverStates = [s ∈ SERVERS ↦  [state ↦ "waiting",
                                      userId ↦ "UNSET",
                                      metadata ↦ "UNSET",
                                      imageId ↦ "UNSET",
                                      image ↦ "UNSET"
                                     ]]
    ∧ operations = ⟨⟩

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

Writes

StartWrite(s) ≜
```

```
    ∧ serverStates[s].state = "waiting"
    ∧ ∃ u ∈ USERIDS, m ∈ METADATAS, i ∈ IMAGES :
        ∧ serverStates' = [serverStates EXCEPT
                                ![s].state = "started_write",
                                Set values for the upcoming write
                                ![s].userId = u,
                                ![s].metadata = m,
                                ![s].image = i]
          Record the write for observability
        ∧ operations' = Append(operations,
                                    [
                                        type ↦ "WRITE",
                                        userId ↦ u,
                                        metadata ↦ m,
                                        image ↦ i
                                    ])
    ∧ UNCHANGED ⟨databaseState, blobStoreState⟩

WriteBlob(s) ≜
    LET currentState ≜ serverStates[s]
    IN
    ∧ currentState.state = "started_write"
    ∧ ∃ id ∈ UUIDS :
        Guarantees a unique Id to simulate UUID. Note: If we run out of unset UUIDs, our
        system will just stop writing. We need to look out for this and ensure the set of UUIDs
        is large enough.
        ∧ blobStoreState[id] = "UNSET"
        ∧ blobStoreState' = [blobStoreState EXCEPT
                                ![id] = currentState.image]
          Track Id to write to database
        ∧ serverStates' = [serverStates EXCEPT
                                ![s].state = "wrote_blob",
                                ![s].imageId = id]
    ∧ UNCHANGED ⟨databaseState, operations⟩

 Writing the database is now the last part of a write operation
WriteMetadataAndReturn(s) ≜
    LET currentState ≜ serverStates[s]
    IN
        ∧ currentState.state = "wrote_blob"
        ∧ databaseState' = [databaseState EXCEPT
                                ![currentState.userId] = [
                                    metadata ↦ currentState.metadata,
                                     Store imageId in database for read
                                    imageId ↦ currentState.imageId]]
```

```
        ∧ serverStates' = [serverStates EXCEPT
                                    ![s].state = "waiting",
                                    ![s].userId = "UNSET",
                                    ![s].metadata = "UNSET",
                                    ![s].image = "UNSET",
                                    ![s].imageId = "UNSET"]
        ∧ UNCHANGED ⟨blobStoreState, operations⟩

FailWrite(s) ≜
    ∧ serverStates[s].state ∈ {"started_write", "wrote_blob"}
    ∧ serverStates' = [serverStates EXCEPT
                                ![s].state = "waiting",
                                ![s].userId = "UNSET",
                                ![s].metadata = "UNSET",
                                ![s].image = "UNSET",
                                ![s].imageId = "UNSET"]
    ∧ UNCHANGED ⟨databaseState, blobStoreState, operations⟩
```

Reads

```
StartRead(s) ≜
     Reading only starts when a server is waiting
    ∧ serverStates[s].state = "waiting"
    ∧ ∃ u ∈ USERIDS :
            serverStates' = [serverStates EXCEPT
                                ![s].state = "started_read",
                                ![s].userId = u]

    ∧ UNCHANGED ⟨databaseState, blobStoreState⟩
    ∧ UNCHANGED operations

 If database record is present
ReadMetadata(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

 If database record is not present
ReadMetadataAndReturnEmpty(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⟩

ReadBlobAndReturn(s) ≜
    LET currentState ≜ serverStates[s]
    IN
     ∧ currentState.state = "read_metadata"
     ∧ operations' = Append(operations,
                                [
                                    type ↦ "READ",
                                    userId ↦ currentState.userId,
                                    metadata ↦ currentState.metadata,
                                     Looks up image by imageId
                                    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⟩
```

Specification / *Next*

$Next \triangleq$

For every step, pick a server and have it advance one state

$\exists\, s \in SERVERS :$
$\lor\ StartWrite(s)$
$\lor\ WriteBlob(s)$ New step
$\lor\ WriteMetadataAndReturn(s)$ New step
$\lor\ FailWrite(s)$
$\lor\ StartRead(s)$
$\lor\ ReadMetadata(s)$ New step
$\lor\ ReadMetadataAndReturnEmpty(s)$ New step
$\lor\ ReadBlobAndReturn(s)$

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

Invariants

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

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

This is used for model checker configuration so the simulation doesn't go on forever.

$StopAfter3Operations \triangleq$
$Len(operations) \leq 3$

$StopAfter5Operations \triangleq$
$Len(operations) \leq 5$