Entity History Mechanism: Intro & How To Guide

What Is the Entity History Mechanism?

Think of it as a digital journal for any "thing" (entity) in our system—like a booking, quote, or chat. It tracks the full story: creation, updates, comments, status changes, and attachments. It's standalone but plugs into any entity, linking to notifications for alerts. Built on existing features (keyword/keyvalue for statuses/actions, Spatie media for files, graph for hierarchies if needed).

Key benefits:

- Audit trail for compliance.
- Facebook-like comments/replies.
- Status tracking (e.g., submitted → approved).
- Notifications to stakeholders.
- Attachments (images, audio, video, docs).

Components Explained

CommMaster (The Journal Cover)

- **Purpose**: Main record for an entity's history.
- **Fields** (layman's view):
    - Entity Type/ID: What (e.g., Booking) and which one (ID 123).
    - Title: Short name, e.g., "Booking #123 by John".
    - Description: Extra details.
    - Status: Current state (from keyvalues: submitted, approved, rejected—use KeywordValueService for IDs).
    - Actor: Who last acted (user ID).
    - Timestamps/Soft Deletes: When created/updated/deleted.
- **How it links**: Morphs to any entity (e.g., Booking::commMaster).

CommThread (The Journal Pages)

- **Purpose**: Entries for actions/comments/replies.
- **Fields**:
    - Comm Master ID: Ties to journal.
    - Parent ID: For replies (nested like Facebook threads, using Nestedset).
    - Actor ID: Who wrote it.
    - Action ID: What happened (from keyvalues: created, updated, remarked, status_changed).

    - Title: Optional label.
    - Message Text: The comment.
    - Extra Data: JSON for custom info.
    - Attachments: Files via Spatie media (images, audio, video, PDFs—collection 'attachments').
    - Timestamps/Soft Deletes.
- **Nesting**: Replies form trees (get children recursively).

How It Works

1. **Create History**: When an entity is created/updated, make CommMaster if none exists.
2. **Add Entry**: Log action as CommThread (top-level or reply).
3. **Status Update**: Change CommMaster status, add thread for details.
4. **Notifications**: On new thread, notify stakeholders (owner, approvers via graph/RBAC).
5. **Attachments**: Upload files to thread's media.
6. **Retrieval**: Get master, then threads (root + children).

Error Handling: Uses BaseController (successResponse/errorResponse/handleException). E.g., not found → 404 uniform JSON.

Use Cases & Examples

Use Case 1: Basic Entity Creation (e.g., Booking)

- **Scenario**: User creates booking.
- **Steps**:
    1. After Booking::create, call $historyService->createMaster($booking, 'Booking #123', 'New vehicle booking');.
    2. Add thread: $historyService->addThread($master, 'created', 'Booking Started', 'Details here');.
- **Status History**: Master status starts 'submitted'.
- **Communication**: No initial comments.

Use Case 2: Status Change (e.g., Approve Quote)

- **Scenario**: Approver changes quote status to approved.
- **Steps**:
    1. Get master: $quote->commMaster.
    2. Update master status: $master->update(['status_id' => KeywordValueService::getValueId('entity_statuses', 'approved')]);.
    3. Add thread: $historyService->addThread($master, 'status_changed', 'Approved', 'Approved with note');.
- **Notification**: Alerts quote owner/others.

Use Case 3: Facebook-Like Comments (e.g., Enquiry Discussion)

- **Scenario**: Team discusses enquiry.
- **Steps**:
  1. Add top comment: $historyService->addThread($master, 'remarked', 'Question', 'Need more info?');.
  2. Reply: $historyService->addThread($master, 'remarked', 'Reply', 'Here it is', [], $parentThread);.
- **With Attachments**: Pass ['path' => $file->store('attachments')] array.
- **Use Case Variant**: Audio note on claim—upload MP3, notify stakeholders.

Use Case 4: Pluggable to Chat/Communication

- **Scenario**: Attach to Message entity for threaded replies.
- **Steps**: Same as above; morph handles any model.

Use Case 5: Error Cases

- Invalid Action: Throws validation exception (uniform JSON).
- No Master: Auto-creates on the first thread.
- Delete Thread: Soft delete, audit logged.

Edge Cases

- Nested Replies (10 levels deep): Nestedset handles.
- Large Attachments: Spatie limits by config.
- Notifications Fail: Log error, don't block.
- Deleted Entity: History remains (soft delete).
- Multiple Stakeholders: Use RBAC/Graph for dynamic lists.

Usage Examples

Web (Backpack CRUD)

1. In BookingCrudController setup:

```
protected function setupCreateOperation() {
$this->crud->addField([...]);
$this->crud->setValidation(BookingRequest::class);
}

public function store(StoreRequest $request) {
    $item = $this->crud->create($request->validated());
$historyService->createMaster($item, 'Booking #' . $item->id);
$historyService->addThread($item->commMaster, 'created', 'Booking
Created');
    return \Redirect::to($this->crud->route);
}
```

View History: Add button to list, load threads via AJAX.

API

1. Get History: GET /api/v1/history/Booking/123 → Returns threads JSON.
2. Add Comment: POST /api/v1/history/Booking/123/thread Body: {"action_key": "remarked", "message": "Hi", "attachments": [files]}.
3. Reply: Include "parent_id": 456.

Integration Tips

- **Notifications**: Auto-triggered on threads.
- **Media**: Use Spatie for uploads.
- **Keywords**: Use for actions/statuses (e.g., KeywordMaster 'entity_actions').
- **Graph**: For stakeholder hierarchies.
- **Errors**: Uniform via BaseController.
- **Testing**: Seed entities, assert threads/notifications.

This covers full setup/use. Expand as needed.

Use case Examples
example of following use cases :

1. I have created a new Booking # 2345 now I want to make a status creation of it
2. After few hours I want to add a question from userid 10 on this Booking
3. Userid 15 has replied on the question
4. UserId 10 has asked 3 separate questions
5. userid 12 has changed the status to "On-hold"
6. userid 12 added a comment
7. Userid 18 restored the booking to Processed status
8. Userid 11 replied to all three questions of userid 10 separately
9. userid 20 wants to see the status only history of booking #2345
10. userid 25 wants to see the communication only history
11. userid 30 wants to see the full history

Understanding the Entity History Mechanism

Imagine managing items like bookings or quotes—each has a lifecycle with creations, updates, questions, and resolutions. The Entity History Mechanism acts as a shared diary, logging every event. It supports status tracking (e.g., "submitted" to "approved") and Facebook-style threaded comments/replies with attachments (images, audio, video, docs). It's standalone but attaches to any entity via morph relations. Integrates with notifications to alert stakeholders (owners, approvers). Uses keyvalues for statuses/actions, Spatie media for files, Nestedset for threading.

Components: CommMaster and CommThread

- **CommMaster**: The diary's cover. One per entity.

    - Fields: Entity type/ID (links to Booking/Quote), title (e.g., "Booking #2345"), description, status ID (from keyvalues like 'submitted'), actor ID (last user), timestamps/soft deletes.
    - Example: For Booking #2345, title="New Vehicle Booking", status='submitted'.
- **CommThread**: The diary's pages. Entries for actions/comments, nested for replies.
    - Fields: Master ID (ties to cover), parent ID (for replies), actor ID (who wrote), action ID (from keyvalues like 'created', 'remarked'), title, message text, extra data (JSON), attachments (via Spatie), timestamps/soft deletes.
    - Nesting: Replies under parents form trees (query rootThreads() for top-level).

How It Works

1. **Attach to Entity**: On entity create (e.g., Booking::create), auto-make CommMaster if none.
2. **Log Status/Action**: Update master status; add thread for details.
3. **Add Comment/Reply**: Create thread (top-level or under parent); upload attachments.
4. **Notifications**: On new thread, auto-notify stakeholders (entity owner + approvers from graph/RBAC).
5. **Retrieval**: Get master, then threads (with children/media).
6. **Errors**: Uniform JSON via BaseController (e.g., 404 for missing history).
7. **Pluggable**: Morphs to any model; use in services/controllers.

Use Cases & Examples

Assume each action in new session—no vars persist. Locate via entity type/ID. Use EntityHistoryService ($historyService). Web: Backpack CRUD. API: /api/v1/history.

1. Created Booking #2345, Make Status Creation

- **Logic**: Create master; add 'created' thread.
- **Code (Service/Controller)**:

```
$booking = Booking::find(2345);  // Locate entity

$master = $booking->commMaster ?? $historyService->createMaster($booking, 'Booking #2345', 'New booking', 'created');

$historyService->addThread($master, 'created', 'Status: Created', 'Booking initiated');
```

- **Web (Backpack)**: In store(): Above code after create.
- **API**: POST /history/Booking/2345/thread { "action_key": "created", "title": "Status: Created" }.
- **Result**: Master status 'created'; thread logged.

2. Add Question from UserID 10 on Booking

- **Logic**: Locate master; add 'remarked' thread.
- **Code**:

```
$booking = Booking::find(2345);

$master = $booking->commMaster;
```

$historyService->addThread($master, 'remarked', 'Question', 'Need more details?', [], null, User::find(10));

- **Web**: Form submission in CRUD view.
- **API**: POST /history/Booking/2345/thread { "action_key": "remarked", "title": "Question", "message": "Need more details?" } (auth as user 10).
- **Notification**: Alerts stakeholders.

3. UserID 15 Replies to Question

- **Logic**: Locate thread; add reply.
- **Code**:

$booking = Booking::find(2345);

$master = $booking->commMaster;

$questionThread = CommThread::where('comm_master_id', $master->id)->where('title', 'Question')->first(); // Locate thread

$historyService->addThread($master, 'remarked', 'Reply', 'Here are details', [], $questionThread, User::find(15));

- **Web**: Reply form under comment.
- **API**: POST /history/Booking/2345/thread { "action_key": "remarked", "title": "Reply", "message": "Here are details", "parent_id": $questionThread->id }.
- **Result**: Nested under question.

4. UserID 10 Asks 3 Separate Questions

- **Logic**: Add 3 top-level threads.
- **Code**:

$booking = Booking::find(2345);

$master = $booking->commMaster;

for ($i=1; $i<=3; $i++) {

    $historyService->addThread($master, 'remarked', "Question $i", "Q$i details?", [], null, User::find(10));

}

- **Web**: Loop in form submission.
- **API**: 3 separate POSTs with different titles/messages.
- **Result**: 3 independent threads.

5. UserID 12 Changes Status to "On-hold"

- **Logic**: Update master status; add thread.
- **Code**:

$booking = Booking::find(2345);

```
$master = $booking->commMaster;

$onHoldId = KeywordValueService::getValueId('entity_statuses', 'on-hold');

$master->update(['status_id' => $onHoldId]);

$historyService->addThread($master, 'status_changed', 'On Hold', 'Hold reason', [], null, User::find(12));
```

- **Web**: Status dropdown in CRUD.
- **API**: PATCH /bookings/2345 { "status": "on-hold" } (integrate in controller).
- **Notification**: Alerts on status change.

6. UserID 12 Adds Comment

- **Logic**: Add 'remarked' thread.
- **Code**:

```
$booking = Booking::find(2345);

$master = $booking->commMaster;

$historyService->addThread($master, 'remarked', 'Comment', 'Additional note', [], null, User::find(12));
```

- **Web**: Comment box.
- **API**: POST /history/Booking/2345/thread { "action_key": "remarked", "message": "Additional note" }.

7. UserID 18 Restores to Processed Status

- **Logic**: Update status; add thread.
- **Code**:

```
$booking = Booking::find(2345);

$master = $booking->commMaster;

$processedId = KeywordValueService::getValueId('entity_statuses', 'processed');

$master->update(['status_id' => $processedId]);

$historyService->addThread($master, 'status_changed', 'Processed', 'Restored', [], null, User::find(18));
```

- **Web/API**: Similar to case 5.

8. UserID 11 Replies to All Three Questions of UserID 10 Separately

- **Logic**: Locate each question thread; add replies.
- **Code**:

```
$booking = Booking::find(2345);

$master = $booking->commMaster;
```

```
$questions = CommThread::where('comm_master_id', $master->id)

    ->where('actor_id', 10)

    ->where('title', 'like', 'Question%')

    ->get();  // Locate threads

foreach ($questions as $q) {

    $historyService->addThread($master, 'remarked', 'Reply', "Answer to {$q->title}", [], $q,
User::find(11));

}
```

- **Web**: Reply forms under each.
- **API**: 3 POSTs with parent_id for each question.

9. UserID 20 Sees Status Only History of Booking #2345

- **Logic**: Query master updates (audit log via owen-it).
- **Code**:

```
$booking = Booking::find(2345);

$master = $booking->commMaster;

$statusHistory = $master->audits()->where('event',
'updated')->where('new_values->status_id', '!=', null)->get(['old_values->status_id',
'new_values->status_id', 'created_at']);
```

- **Web**: Table in view.
- **API**: GET /history/Booking/2345/status → Return $statusHistory.

10. UserID 25 Sees Communication Only History

- **Logic**: Filter threads by 'remarked' action.
- **Code**:

```
$booking = Booking::find(2345);

$master = $booking->commMaster;

$remarkId = KeywordValueService::getValueId('entity_actions', 'remarked');

$commHistory = $master->rootThreads()->where('action_id',
$remarkId)->with('children')->get();
```

- **Web**: Filtered list.
- **API**: GET /history/Booking/2345/communication → Return $commHistory.

11. UserID 30 Sees Full History

- **Logic**: Get master + all threads.
- **Code**:

```
$booking = Booking::find(2345);

$master = $booking->commMaster;

$fullHistory = $master->load('threads.children', 'threads.media', 'threads.actor',
'threads.action');
```

- **Web**: Full timeline view.
- **API**: GET /history/Booking/2345 → Return $fullHistory.

Usage (Web & API)

- **Web (Backpack)**: In CRUD controllers, hook into store/update for auto-logging. Views: Use relations to display threads.
- **API**: Use EntityHistoryController for get/add. Auth via Sanctum.
- **Errors**: Handle via BaseController (uniform JSON).
- **Tips**: Always locate via entity type/ID. Use transactions for atomicity.

Chat Usage Examples with Entity History Mechanism

Below are detailed examples for all key use cases of using the mechanism for group or one-to-one chats (no entity required). Each action is stateless—assume new sessions, so locate chat/master/thread by ID each time. Use EntityHistoryService ($historyService). Code is full/executable; integrate in controllers/services as needed. Web: Backpack CRUD. API: Via EntityHistoryController (POST /chats for create, POST /history/Chat/{chatId}/thread for messages).

1. Create New Group Chat (Users 10, 15, 20)

- **Logic**: Create Chat model; attach master.
- **Code (Service/Controller)**:

```
// Locate users (stateless)

$participants = [10, 15, 20];

$master = $historyService->createChatMaster('group', $participants, 'Team Discussion',
'Group chat for project');
```

Web (Backpack): In ChatCrudController store(): Above code.

- **API**: POST /chats { "type": "group", "participants": [10,15,20], "title": "Team Discussion" }.
- **Result**: Chat ID created; master for history.

2. Create New One-to-One Chat (Users 10, 15)

- **Logic**: Similar, but type 'one_to_one'.
- **Code**:

```
$participants = [10, 15];

$master = $historyService->createChatMaster('one_to_one', $participants, 'Private Chat',
'1-1 discussion');
```

- **Web**: Form in CRUD.
- **API**: POST /chats { "type": "one_to_one", "participants": [10,15], "title": "Private Chat" }.
- **Result**: Chat for direct messaging.

3. Add Message to Chat (UserID 10 on Chat ID 1)

- **Logic**: Locate master; add 'messaged' thread (custom action keyvalue).
- **Code**:

```
$chat = Chat::find(1);  // Locate chat

$master = $chat->commMaster;  // Or CommMaster::where('entityable_type',
'App\Models\Core\Chat')->where('entityable_id', 1)->first();

$historyService->addThread($master, 'messaged', 'Message', 'Hello team!', [], null,
User::find(10));
```

- **Web**: Message form.
- **API**: POST /history/Chat/1/thread { "action_key": "messaged", "message": "Hello team!" } (auth as user 10).
- **Notification**: Alerts participants except sender.

4. Reply to Message (UserID 15 on Thread ID 5 in Chat ID 1)

- **Logic**: Locate thread; add reply.
- **Code**:

```
$chat = Chat::find(1);

$master = $chat->commMaster;

$parentThread = CommThread::find(5);  // Locate thread

$historyService->addThread($master, 'messaged', 'Reply', 'Hi!', [], $parentThread, User::find(15));
```

- **Web**: Reply button under message.
- **API**: POST /history/Chat/1/thread { "action_key": "messaged", "message": "Hi!", "parent_id": 5 }.
- **Result**: Nested reply.

5. Add Multiple Messages (UserID 10 Adds 3 to Chat ID 1)

- **Logic**: Add 3 top-level threads.
- **Code**:

```
$chat = Chat::find(1);

$master = $chat->commMaster;

for ($i=1; $i<=3; $i++) {

    $historyService->addThread($master, 'messaged', "Message $i", "Msg $i content", [], null, User::find(10));

}
```

- **Web**: Loop submissions.
- **API**: 3 separate POSTs.
- **Result**: 3 messages.

6. Change Chat Status to "Archived" (UserID 12 on Chat ID 1)

- **Logic**: Update master status; add thread.
- **Code**:

```
$chat = Chat::find(1);

$master = $chat->commMaster;

$archivedId = KeywordValueService::getValueId('entity_statuses', 'archived');

$master->update(['status_id' => $archivedId]);
```

```
$historyService->addThread($master, 'status_changed', 'Archived', 'Chat closed', [], null,
User::find(12));
```

- **Web/API**: Similar to status update.

7. UserID 12 Adds Comment/Note to Chat ID 1

- **Logic**: Add 'remarked' thread.
- **Code**:

```
$chat = Chat::find(1);

$master = $chat->commMaster;

$historyService->addThread($master, 'remarked', 'Note', 'Important reminder', [], null,
User::find(12));
```

- **Web/API**: As in case 3.

8. UserID 18 Restores Chat to Active Status (Chat ID 1)

- **Logic**: Update status; add thread.
- **Code**:

```
$chat = Chat::find(1);

$master = $chat->commMaster;

$activeId = KeywordValueService::getValueId('entity_statuses', 'active');

$master->update(['status_id' => $activeId]);

$historyService->addThread($master, 'status_changed', 'Active', 'Reopened', [], null,
User::find(18));
```

- **Web/API**: As in case 6.

9. UserID 11 Replies to All Three Messages of UserID 10 Separately (Chat ID 1)

- **Logic**: Locate messages; add replies.
- **Code**:

```
$chat = Chat::find(1);

$master = $chat->commMaster;

$messagedId = KeywordValueService::getValueId('entity_actions', 'messaged');

$messages = CommThread::where('comm_master_id', $master->id)

   ->where('actor_id', 10)

   ->where('action_id', $messagedId)

   ->get();  // Locate threads
```

```
foreach ($messages as $m) {

    $historyService->addThread($master, 'messaged', 'Reply', "Answer to {$m->title}", [], $m, User::find(11));

}
```

- **Web/API**: As in case 4.

10. UserID 20 Sees Status Only History of Chat ID 1

- **Logic**: Audit log on master.
- **Code**:

```
$chat = Chat::find(1);

$master = $chat->commMaster;

$statusHistory = $master->audits()->where('event', 'updated')->where('new_values->status_id', '!=', null)->get(['old_values->status_id', 'new_values->status_id', 'created_at']);
```

- **Web**: Display table.
- **API**: GET /history/Chat/1/status → Return $statusHistory.

11. UserID 25 Sees Communication Only History of Chat ID 1

- **Logic**: Filter 'messaged' threads.
- **Code**:

```
$chat = Chat::find(1);

$master = $chat->commMaster;

$messagedId = KeywordValueService::getValueId('entity_actions', 'messaged');

$commHistory = $master->rootThreads()->where('action_id', $messagedId)->with('children')->get();
```

- **Web**: Filtered view.
- **API**: GET /history/Chat/1/communication → Return $commHistory.

12. UserID 30 Sees Full History of Chat ID 1

- **Logic**: Master + all threads.
- **Code**:

```
$chat = Chat::find(1);

$master = $chat->commMaster;

$fullHistory = $master->load('threads.children', 'threads.media', 'threads.actor', 'threads.action');
```

- **Web**: Timeline.
- **API**: GET /history/Chat/1 → Return $fullHistory.

These are stateless—always locate first. Errors uniform via BaseController.