

Quiz & Quest feature

What is the Quiz & Quest Feature?

The **Quiz & Quest** module is a dual-engagement system designed to:

- **Educate users through Quizzes** (objective question-based challenges).
 - **Encourage real-world or skill-based actions through Quests** (task submissions that require approval).
-

Who Uses It?

1. Admin Users (HR, Managers, Admins)

Admins can:

- Create Quizzes and Quests.
- Link them to specific organizations and departments.
- Set start and end times, point values, rules, and attempts.
- Define how and when alerts (via Email, SMS, WhatsApp, Push Notification) are sent.
- Review Quest submissions and approve or reject them.

2. Normal Users (Employees, Learners, Participants)

Users can:

- View quizzes and quests relevant to their department and organization.
- **Attempt quizzes** within the given time and try to pass based on a minimum score.
- **Submit quests** by uploading content (text, files, etc.) based on the instructions.
- Earn points upon successful **quiz completion** or **quest approval**.
- Track their progress and number of attempts.

Feature Summary

Feature	Quiz	Quest
Purpose	Test knowledge with MCQs	Complete real-world tasks or challenges
Attempt Logic	Has maximum attempts, time-bound	Submit once, await review/approval
Evaluation	Auto-evaluated	Manually reviewed by admin
Points	Awarded only if passed	Awarded only on admin approval
Notifications	Sent based on alert mode configuration	Sent during creation & after approval

Example Scenarios

Quiz

“How much do you know about company policies?”

You attempt 10 MCQs. If you score above the minimum (say 7/10), you earn points.

Quest

“Submit a video showcasing your teamwork during the last project.”

You upload a file and write a description. An admin reviews your submission. If they approve it, you earn points.

DB Models:

1. AlertMode

Used to define how and when alerts (notifications) should be sent for a Quiz or Quest.

- name: Mode of alert delivery (email, sms, push, etc.), from predefined ALERT_MODE choices.
- template: FK to common_app.Template. Used to define the content format for alerts.
- alert_schedule_type: When to send the alert. Choices from ALERT_SCHEDULE_TYPE like same, scheduled.
- alert_schedule_time: Specific time to send the alert, applicable if type is scheduled.

Relations:

- Used in Quiz and Quest via a ManyToMany relationship (alert_modes).
-

2. Quiz

Represents a quiz created by an admin with questions and scoring.

Inherits from BaseContent, which includes:

- title, description, organizations, departments, start_datetime, end_datetime, points, rewards, alert_modes, lock, created_by, created, updated

Additional fields:

- minimum_score_required: Minimum percentage to pass the quiz.
- maximum_attempts: 0 for unlimited attempts.
- question_order: Whether questions appear in a fixed or random order.
- choices_order: Whether options appear in a fixed or random order.
- number_of_questions_to_pick: If > 0, picks a subset from the pool.

Relations:

- ManyToMany with Question through QuizQuestion.
 - Has multiple QuizAttempts by users.
-

3. Quest

Represents a task or challenge that users can submit responses to.

Also inherits from BaseContent (same fields as above).

Additional fields:

- approval_start_datetime: When the quest submissions can start being reviewed.
- approval_end_datetime: When review period ends.
- allow_edit: Whether users can edit their submission.
- allowed_submissions: Number of submissions allowed. 0 = unlimited.
- approvers: Users allowed to approve/reject quest submissions.

Relations:

- ManyToMany with CustomUser for approvers.
 - Has multiple UserQuestSubmission instances from users.
-

4. Question

Represents a quiz question.

- text: Question content.
- question_type: Type of question (e.g., multiple choice).
- allow_multiple_correct: If multiple options can be correct.
- authors: M2M with users who authored the question.
- created, updated

Relations:

- Has Many Options.
 - Linked to multiple quizzes via QuizQuestion.
 - Used in Answer.
-

5. QuizAttempt

Tracks a user's attempt on a quiz.

- user: FK to CustomUser who attempted.
- quiz: FK to Quiz (nullable).
- score: Actual score obtained.
- attempt_number: Which attempt number (1, 2, ...).
- passed: Whether user passed based on minimum_score_required.
- status: in_progress, completed, etc.
- started_at, ended_at, created, updated
- transaction: FK to Transaction, optional

Relations:

- Has many Answers.
 - Belongs to one Quiz.
-

6. Answer

Captures a user's response to a question in a quiz attempt.

- question: FK to Question.
- quiz_attempt: FK to QuizAttempt.

- selected_options: M2M to Option (for multiple choice answers).
 - text_answer: Text response (if applicable).
 - created, updated
-

7. Option

Represents one possible answer for a Question.

- question: FK to Question.
 - text: Option text.
 - is_correct: Whether this is a correct option.
 - order: Integer to maintain display order.
 - created, updated
-

8. QuizQuestion

Defines the connection between a Quiz and a Question.

- quiz: FK to Quiz.
- question: FK to Question.
- order: Display order in the quiz.
- points: Points for correct answer.

Relations:

- Connects Quiz and Question as a through table.
 - Unique constraint on (quiz, question).
-

9. UserQuestSubmission

Tracks what a user submitted for a quest.

- quest: FK to Quest.
- user: FK to CustomUser.
- created, updated
- content: User's submission text.
- status: in_review, approved, rejected, etc.
- feedback: Approver's comments.
- edit_count: Number of edits done.
- attachments: M2M to common_app.Attachment.
- approver: FK to CustomUser who approved (nullable).
- approved_on: Timestamp of approval.
- transaction: FK to Transaction, optional.

Questions — Overview and Functionality

Purpose:

Questions are core components of a quiz. Admin users can define questions, specify their type (e.g., single or multiple choice), and link them to quizzes.

Creating, Updating, and Deleting Questions

Who can create/edit/delete?

Only **admin users** have permissions to manage questions.

1. Create Question

Endpoint: POST /quiz/api/questions/

Request Body:

None

```
{
  "text": "What is the capital of France?",
  "question_type": "multiple_choice",
  "allow_multiple_correct": false,
  "authors": [USER_ID]
}
```

Important Fields:

- text: The question text.
- question_type: Type of question — e.g., multiple_choice.
- allow_multiple_correct: If true, more than one correct answer can be selected.
- authors: Admin users who authored the question.

2. Update Question

Endpoint: PATCH /quiz/api/questions/{id}/

Allows partial updates like changing text or toggling multiple correct options.

3. Retrieve Question

Endpoint: GET /quiz/api/questions/{id}/

Fetch full detail of a question including its options and link status with quizzes.

4. Delete Question

While there's no explicit delete endpoint shown in Postman, typically Django REST allows DELETE /questions/{id}/ if enabled.

Linking Questions to Quizzes

You can **link** or **unlink** a question to/from a quiz using two options:

Link Quiz to Question

Endpoint: POST /quiz/api/questions/{id}/link_quiz/

Payload:

```
None
{
  "quiz_id": QUIZ_ID,
  "order": 1,
  "points": 2.0
}
```


- order: Position of the question in the quiz.
- points: How many points this question carries in the quiz context.

Unlink Quiz from Question

Endpoint: POST /quiz/api/questions/{id}/unlink_quiz/

Payload:

```
None
{
  "quiz_id": QUIZ_ID
}
```

 These APIs are typically used when managing a question from its detail page.

List Questions

Endpoint: GET /quiz/api/questions/

Can be filtered or paginated. Returns all questions (usually for admin).

QUIZ — Overview

A **Quiz** is a time-bound set of questions configured by admins to test user knowledge. Quizzes are linked to organizations and departments and reward users upon successful completion based on scoring rules.

Admin Workflow — Creating & Managing a Quiz

Create Quiz

Endpoint: POST /quiz/api/quizzes/

Fields & Their Purpose:

Field	Description
title	Name of the quiz.
description	Optional explanation or context for the quiz.
start_datetime	When the quiz becomes visible and accessible.
end_datetime	When the quiz expires.
minimum_score_required	Minimum percentage needed to pass.
maximum_attempts	0 = unlimited. Otherwise, limits user attempts.
points	Points awarded for passing.
organizations	Users of these orgs can access this quiz.

departments (Optional) Narrowed to departments under selected orgs.

rewards Associated rewards given after passing.

alert_modes Notifications to be sent post-creation.

lock If true, hides quiz before start for non-admins.

question_order fixed or random — determines question display order.

choices_order fixed or random — option display order.

number_of_questions_to_pick Allows random subset of questions (0 = all questions).

⚠ Quizzes are linked with questions via a separate linking API (not during creation).

Update Quiz

Endpoint: PATCH /quiz/api/quizzes/{id}/

Allows updating the above fields.

Link or Delink Questions

To manage which questions are shown in the quiz.

- **Link:**

POST /quiz/api/quizzes/{id}/link_question/

Payload:

None

```
{
  "question_id": 1,
  "order": 1,
  "points": 2.0
}
```

-

- **Delink:**

POST /quiz/api/quizzes/{id}/delink_question/

Payload:

None

```
{
  "question_id": 1
}
```

Admin Quiz Attempts

GET /quiz/api/quizzes/{id}/admin_attempts/

Lists all attempts of all users for the given quiz — useful for evaluation and tracking.

Normal User Workflow — Attempting a Quiz

List Quizzes

GET /quiz/api/quizzes/

Shows quizzes accessible to the user.

Sorting Logic:

1. In-progress attempts first

2. Available quizzes sorted by end_datetime
3. Expired quizzes (sorted descending)

 Filters:

- submission_status=approved,pending
- organization, department, start, end (date filters)

Quiz Detail

GET /quiz/api/quizzes/{id}/

Returns full quiz detail (title, questions, etc.)

Options and question order may be randomized based on config.

Start Attempt

POST /quiz/api/quizzes/{id}/start_attempt/

Starts an attempt. Returns attempt_id. If attempt already in progress, returns that.

Save Answer

POST /quiz/api/quizzes/{id}/save_answer/

Payload:

```
None
{
  "attempt_id": 123,
  "question_id": 55,
  "selected_option_ids": [201],
  "text_answer": ""
}
```

Submit Attempt

POST /quiz/api/quizzes/{id}/submit_attempt/

Payload:

```
None
{
  "attempt_id": 123
}
```

- Evaluation happens server-side.
 - Based on minimum_score_required, marks attempt as passed or failed.
-

Latest Attempt

GET /quiz/api/quizzes/{id}/latest_attempt/

Returns latest attempt by the current user — helpful for resuming or reviewing.

Access Control Logic

- **Organizations** (M2M): Quiz visible only to users from these orgs.
 - **Departments** (M2M): Optional — narrows down to specific departments.
 - lock=True: hides quiz before start_datetime from non-admins.
 - start_datetime/end_datetime: controls visibility and availability.
-

QUEST — Overview

A **Quest** is a user-submitted task or challenge that requires **manual review** before rewards and points are granted. It supports content submission, attachments, and approval workflows — ideal for assignments, uploads, or real-world activities.

Admin Workflow — Creating & Reviewing a Quest

Create Quest

Endpoint: POST /quiz/api/quests/

Fields & Their Purpose:

Field	Description
title	Name of the quest.
description	Task details/instructions.
start_datetime	When the quest becomes active.
end_datetime	When submissions are closed.
approval_start_datetime	When review of submissions can begin.
approval_end_datetime	Deadline for reviewing user submissions.
points	Points awarded for approved submission.
rewards	Associated rewards granted on approval.
organizations	Target audience by organization.

departments	Targeted departments (optional).
alert_modes	Notification modes to be triggered.
lock	If true, hides quest until start for non-admins.
allow_edit	If true, users can edit their submissions.
allowed_submissions	Max submissions allowed per user (0 = unlimited).
approvers	Admin users responsible for reviewing submissions.

⚠ Similar to Quiz, organization and department filters control visibility.

Update Quest

Endpoint: PATCH /quiz/api/quests/{id}/

Allows updating any of the fields mentioned above.

Get Quest List / Detail

- GET /quiz/api/quests/ – list all quests accessible to current user
- GET /quiz/api/quests/{id}/ – get details of a specific quest

Sorting Logic:

1. User has a pending submission → Top
2. Quests ending soon
3. Expired quests (by most recent end)

 Filters:

- submission_status=approved,pending,in_review
 - organization, department, start, end (date filters)
-

✓ Review User Submission

Endpoint: POST /quiz/api/quests/review_submission/

Used by approvers/admins to **approve or reject** a user's submission.

Payload:

```
None
{
  "submission_id": 123,
  "approved": true,
  "feedback": "Great job!"
}
```

Upon approval, points and rewards are credited. Status changes to approved, and reviewer info is recorded.

👤 Normal User Workflow — Submitting a Quest

📝 Create Submission

Endpoint: POST /quiz/api/quests/{id}/submit/

Payload:

```
None
{
  "content": "Here's my answer or submission.",
  "attachment_ids": [1, 2]
}
```

- Each submission is linked to the user and the quest.
- Status is initially set to in_review.
- Limited by allowed_submissions and availability window.

Update Submission

PATCH /quiz/api/quests/{id}/update_submission/

Allowed **only** if allow_edit=True and submission hasn't been approved.

View Submission

- **List Submissions**

GET /quiz/api/quests/my_submissions/

- **Submission Detail**

GET /quiz/api/quests/my_submissions/{id}/

Access Control Logic

Condition	Effect
organizations	Only users of listed orgs can view quest
departments	Further filters quest visibility
start_datetime & end_datetime	Defines availability window

approval_start_datetime	Submission can't be reviewed before this
-------------------------	--

allow_edit	Controls user edit ability
------------	----------------------------

allowed_submissions	Max number of tries per user
---------------------	------------------------------

lock	Hides quests before start time
------	--------------------------------

Additional Notes

- Admin users can see and review all submissions.
 - Points and rewards are granted only after manual review (unless you automate it in future).
 - Every submission tracks approver, feedback, edit_count, and attachments.
-

Notification System in Quiz & Quest

Notifications are sent to users when a **quiz or quest is created** or becomes **available**, based on configured **alert modes**.

Each **Quiz** and **Quest** has a ManyToMany relation to AlertMode, which defines:

- **How** notifications are sent (email, SMS, WhatsApp, push)
 - **When** to send (immediately or scheduled)
 - **Which template** to use for each channel
-

AlertMode Model

Field	Description
name	The mode of delivery — values from ALERT_MODE() enum (e.g., email, sms, push, whatsapp)
template	FK to a Template object that contains the message body
alert_schedule_type	Enum from ALERT_SCHEDULE_TYPE() – e.g., same, scheduled
alert_schedule_time	When to send if using scheduled

These modes can be added to each quiz or quest. The system checks them while sending notifications asynchronously.

Notification Trigger Logic

When a quiz or quest is created:

1. **send_alerts_for_content task is called** (Celery task)
2. It fetches all alert modes (email, sms, push, etc.)
3. For each alert mode:
 - Fetches the target users (based on organizations & departments)
 - Renders the template by filling in **context variables**
 - Sends the message using the correct channel (email gateway, SMS provider, etc.)





Example trigger point (in signal):

None

```
send_alerts_for_content(content=instance, model_name='quiz')
```

Supported Alert Modes (Channels)

These are defined in `ALERT_MODE()` and may include:

-  email
-  sms
-  whatsapp
-  push_notification

Template System

Templates are HTML/text snippets stored in the `common_app.Template` model and are rendered with Django-style context variables.

Supported Context Variables for Templates

These variables are passed when rendering the alert message:

Variable	Meaning
user	The target user receiving the message
quiz or quest	The object (quiz or quest)
content	Alias for either quiz or quest
start_datetime	When the content becomes available

end_datetime	When the content ends
title	Title of the content
description	Description field
points	Points awarded on completion
organization	User's organization
departments	User's department list
link	A CTA link to access the content (custom-built using frontend URL + object ID)

Example placeholders in template:

```
None
Hi {{ user.first_name }},

A new quiz titled "{{ title }}" is now live!

Starts: {{ start_datetime }}
Ends: {{ end_datetime }}

You can access it here: {{ link }}
```

Alert Scheduling

Using alert_schedule_type:

Type	Effect
------	--------

same	Send immediately after creation
------	---------------------------------

scheduled	Send at a custom alert_schedule_time (datetime field)
-----------	---

Scheduled tasks are handled via Celery — queued and executed accordingly.

Summary of Notification Flow

1. Admin creates a quiz/quest and links alert_modes
2. The system triggers send_alerts_for_content
3. Target users are fetched using organization/department filters
4. Templates are rendered using context
5. Alerts are sent using appropriate backends
6. If scheduled, they are delayed till the alert_schedule_time

Filters & Sorting Logic + Scoring & Points System

FILTERS

Filters are supported in both **Quiz** and **Quest** list APIs to help users (especially admins) narrow down results based on different criteria.

Quiz Filters (/quiz/api/quizzes/)

Filter Param	Type	Description
organizations	comma-separated IDs	Show quizzes linked to selected orgs
departments	comma-separated IDs	Show quizzes assigned to specific depts
start	date or datetime	Quizzes starting on/after this time
end	date or datetime	Quizzes ending on/before this time
submission_status	in_progress, passed, failed	Based on the user's quiz attempts
search	string	Searches by quiz title or description

✓ Quest Filters (/quest/api/quests/)

Filter Param	Type	Description
organizations	comma-separated IDs	Show quests for selected orgs
departments	comma-separated IDs	Filter quests by department
start	date or datetime	Start datetime filter
end	date or datetime	End datetime filter

submission_status in_review, approved, rejected Based on user submissions

search string Filter by title or description

↕ SORTING LOGIC

Sorting ensures users see **most relevant or actionable** content first.

🧠 Quiz Sorting Logic

Priority:

1. Quizzes where user has an active (in-progress) attempt
2. Quizzes that are currently available (not ended)
3. Quizzes that have already ended

Secondary sorting:

- Quizzes ending **soonest** are shown first within available quizzes.
- Quizzes that have ended are sorted by **most recently expired**.

Backend Implementation:

None

```
.sort_group: 1 → in-progress  
             2 → available (not ended)  
             3 → expired
```

```
Order by: sort_group, end_datetime ASC (available), end_datetime DESC  
(expired)
```

📖 Quest Sorting Logic

Priority:

1. Quests with a user submission in in_review
2. Available quests (not expired)
3. Expired quests

Secondary:

- Available quests: end_datetime ascending (soonest)
- Expired: end_datetime descending (most recently ended)

SCORING & POINTS SYSTEM

Quiz Scoring

1. Each QuizQuestion has a points field (default: 1.0)
2. A quiz may have:
 - All questions
 - Or a subset (number_of_questions_to_pick) selected randomly

Evaluation Logic (in evaluate_answers()):

- For each Answer, compare user-selected options to the correct ones.
- Full points only if all correct options are selected, nothing extra.
- Score is the sum of all correctly answered question points.
- **Percentage** = (earned points / total question points) * 100

When is a Quiz Passed?

- passed = True if user's percentage \geq minimum_score_required (e.g. 60%)
 - Points (quiz.points) and rewards are granted **only if passed**
 - Transaction is created if points are credited
-

Quest Submission Approval & Points

- Quest has:
 - points (float)
 - M2M rewards
 - User submits via UserQuestSubmission → status = in_review
 - If admin approves:
 - status = approved
 - Points are credited
 - Rewards are granted
 - A transaction is created for this reward
-

Edge Cases

- **Quiz:** If no questions or no attempt submitted, score = 0
 - **Quest:** No points granted if rejected
 - **Attempts:** Quiz has maximum_attempts limit (0 = unlimited)
 - **Edit Count:** For quest submission, tracked via edit_count
-

Admin Flow for Quiz & Quest

QUIZ: Admin Flow

Admins can **create quizzes**, **link questions**, and **manage attempts**.


Create Quiz

API: POST /quiz/api/quizzes/

Admin defines:

Field	Description
title, description	Basic info
organizations, departments	Visibility rules (orgs/depts can access)
start_datetime, end_datetime	Quiz availability window
minimum_score_required	Threshold to pass (%)
maximum_attempts	Limit for user retry
points	Rewarded upon passing
rewards	Optional reward objects (e.g., coupons)
alert_modes	Notification channels (WhatsApp, SMS, etc.)
question_order, choices_order	Controls how questions/options are displayed

number_of_questions_to_pick Randomization from question pool (0 = all)

 **Alert scheduling** is handled by the AlertMode model.

Link Questions to Quiz

Admins can **link or delink questions**:

API	Description
POST /quiz/api/link_question/	Add a question to a quiz
POST /quiz/api/delink_question/	Remove a question from a quiz

Admin Quiz Attempts Review

Admins can fetch attempts:

API	Purpose
GET /quiz/api/quizzes/<id>/admin_attempts/	View all users' attempts for a quiz
GET /quiz/api/quizzes/<id>/attempt/	View latest attempt by user

Add or Edit Questions

API	Purpose
-----	---------

POST /quiz/api/questions/ Create a question

PATCH /quiz/api/questions/<id>/ Update

GET /quiz/api/questions/ List questions

GET /quiz/api/questions/<id>/ Retrieve specific

Each question has:

- text
- question_type: (e.g., MCQ)
- allow_multiple_correct
- Linked options: (choices with is_correct=True/False)

QUEST: Admin Flow

Admins can **create quests**, **set submission rules**, and **review submissions**.

Create Quest

API: POST /quest/api/

Field	Description
title, description	General content

start_datetime, end_datetime	Availability for users
approval_start_datetime, approval_end_datetime	Window for admin approval
allow_edit	Can user update their submission later?
allowed_submissions	Submission limit (0 = unlimited)
points	Points awarded upon approval
rewards	Optional associated rewards
departments, organizations	Access control
approvers	Admins assigned to review submissions
alert_modes	Notification channels for alerting users

Review Submissions

API	Description
POST /quest/api/submission/review/	Approve or reject a submission
PATCH /quest/api/submission/<id>/	Update submission

GET /quest/api/submission/

List of submissions for review

GET /quest/api/submission/<id>/

View submission detail

Review options:

- Mark as approved or rejected
- Add feedback
- Attach reviewer
- Trigger reward/point credit