Add icon Add cover

# 9.13 - Internal Comments and User Insights Mechanism

**Description:**

**As a** registered user of CORDAP

**I want** to be able to post comments and be part of conversations related to resources and datasets

**So that** I can engage with peers around research topics and contribute to crowd-sourced insights for the research team

**Acceptance Criteria**

1. Mechanism to capture user comments or insights directly on the dataset and resource pages
2. Comments should be limited to logged in users, to ensure there is accountability for comments
3. No comments are possible on public datasets
4. Include upvote button, with counter of total upvotes
5. Sort option to reorder comments based on most recent, most engagement (upvotes or replies)
6. Ability to reply to existing comments (as thread under initial comment)
7. Ability to delete comments as dataset owner
8. Simple text comments, with user, date and time stamp on each comment
9. Ability to @tag cordap users within the text of the comment
10. Notification, shown in notifications panel for a user
    a. if someone @tags them
    b. if their dataset is commented
    c. if their comment is replied to
    d. if someone upvotes their comment
11. **optional** - clickable URL links in comments

# Technical Design Document: CKAN Comments Extension

## ckanext-comments

## 1. Overview

### 1.1 Purpose

This document describes the technical design for `ckanext-comments`, a CKAN extension that provides a commenting and discussion system for datasets and resources. The extension enables authenticated users to engage in discussions, mention other users, upvote comments, and receive notifications.

### 1.2 Scope

The extension will be implemented as a reusable CKAN plugin that can be enabled for:

- Dataset detail pages

- Resource detail pages
- Potentially other entity types in future (groups, organizations)

### 1.3 Goals

- Provide a unified commenting system across CKAN entities
- Enable threaded discussions with reply functionality
- Implement @mention system with user notifications
- Support upvoting/reactions to surface relevant comments
- Allow content owners to moderate discussions
- Ensure accountability by restricting comments to authenticated users

## 2. Requirements Summary

### 2.1 Functional Requirements

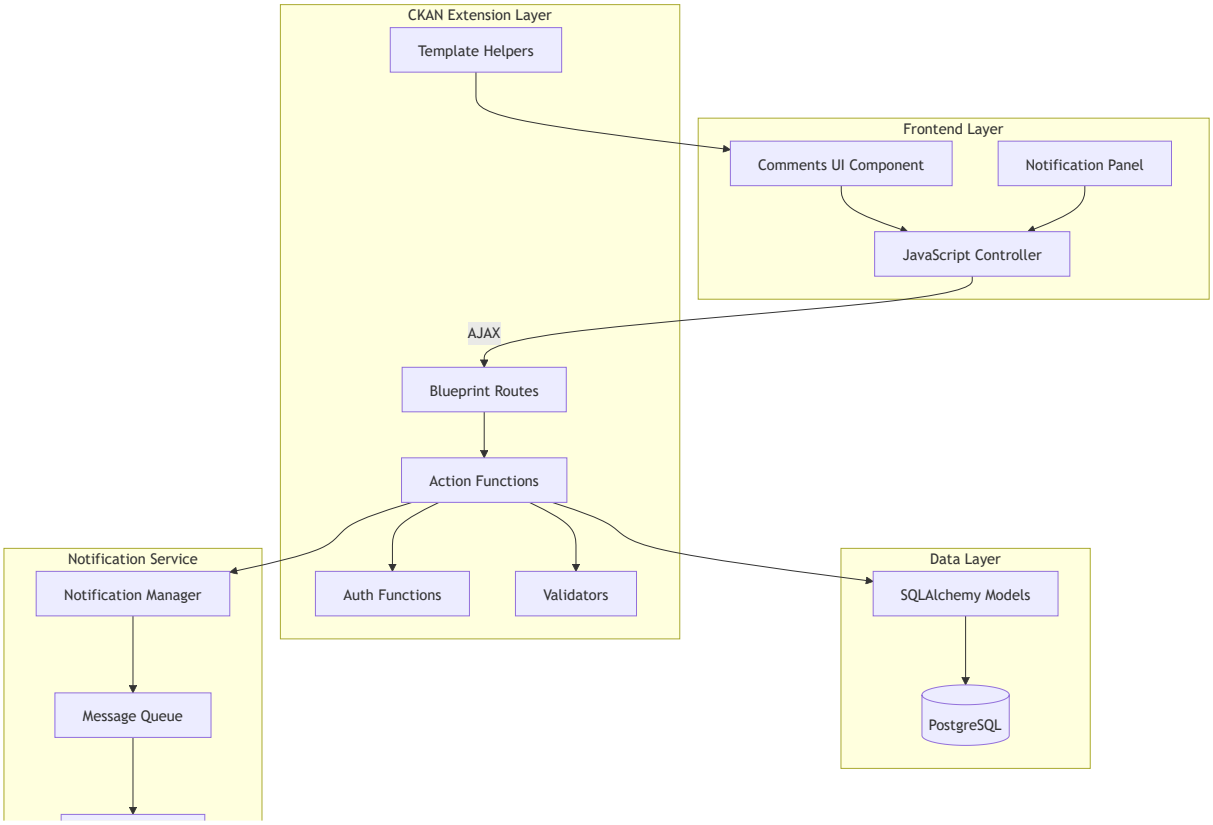
| | ID | Requirement | Priority |
|---|---|---|---|
| 1 | FR-01 | Comments restricted to logged-in users | High |
| 2 | FR-02 | Upvote/reaction system for comments | High |
| 3 | FR-03 | Sort comments by most upvoted/newest/oldest | High |
| 4 | FR-04 | Comments on dataset pages | High |
| 5 | FR-05 | Comments on resource pages | High |
| 6 | FR-06 | Simple text comments | High |
| 7 | FR-07 | User, date, and timestamp on comments | High |
| 8 | FR-08 | @tagging of users with notification | High |
| 9 | FR-09 | Reply to existing comments (threaded) | High |
| 10 | FR-10 | Delete comments as dataset/resource owner | High |
| 11 | FR-11 | Notification panel in UI | Medium |
| 12 | FR-12 | Notification types (mention, reply, upvote, new comment) | Medium |

### 2.2 Non-Functional Requirements

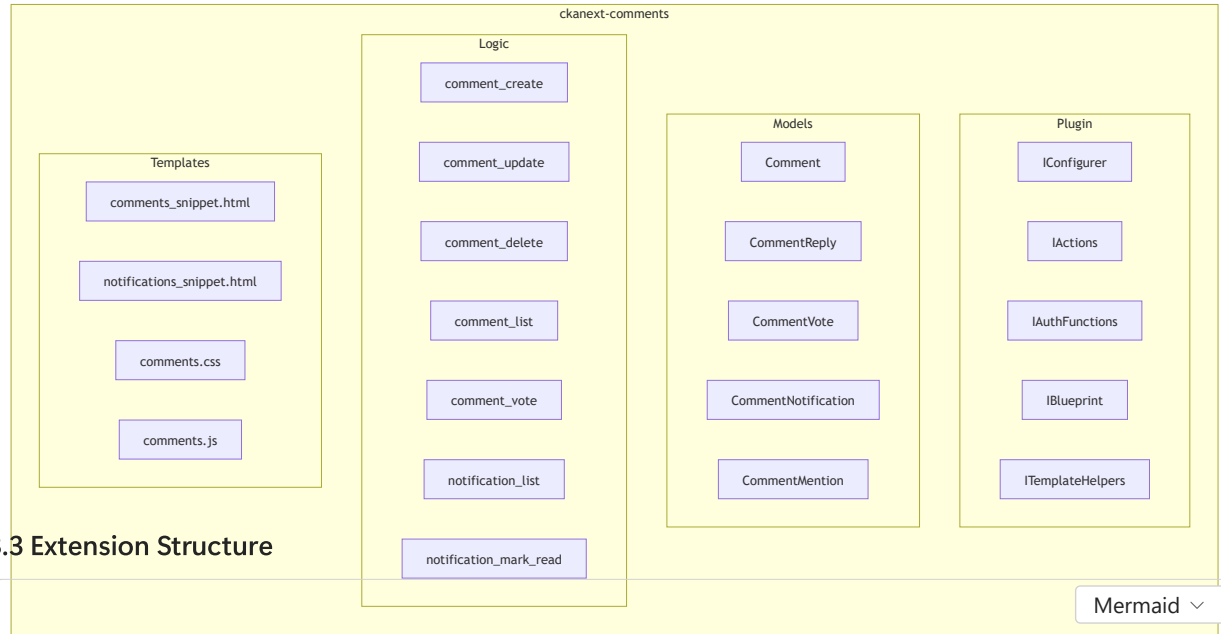
| | ID | Requirement | Priority |
|---|---|---|---|
| 1 | NFR-01 | Response time < 500ms for comment operations | High |
| 2 | NFR-02 | Support 1000+ comments per entity | Medium |
| 3 | NFR-03 | Pagination for large comment threads | Medium |
| 4 | NFR-04 | XSS protection for comment content | High |
| 5 | NFR-05 | Rate limiting to prevent spam | Medium |

## 3. Architecture Design

### 3.1 High-Level Architecture




### 3.2 Component Architecture




## 3.3 Extension Structure

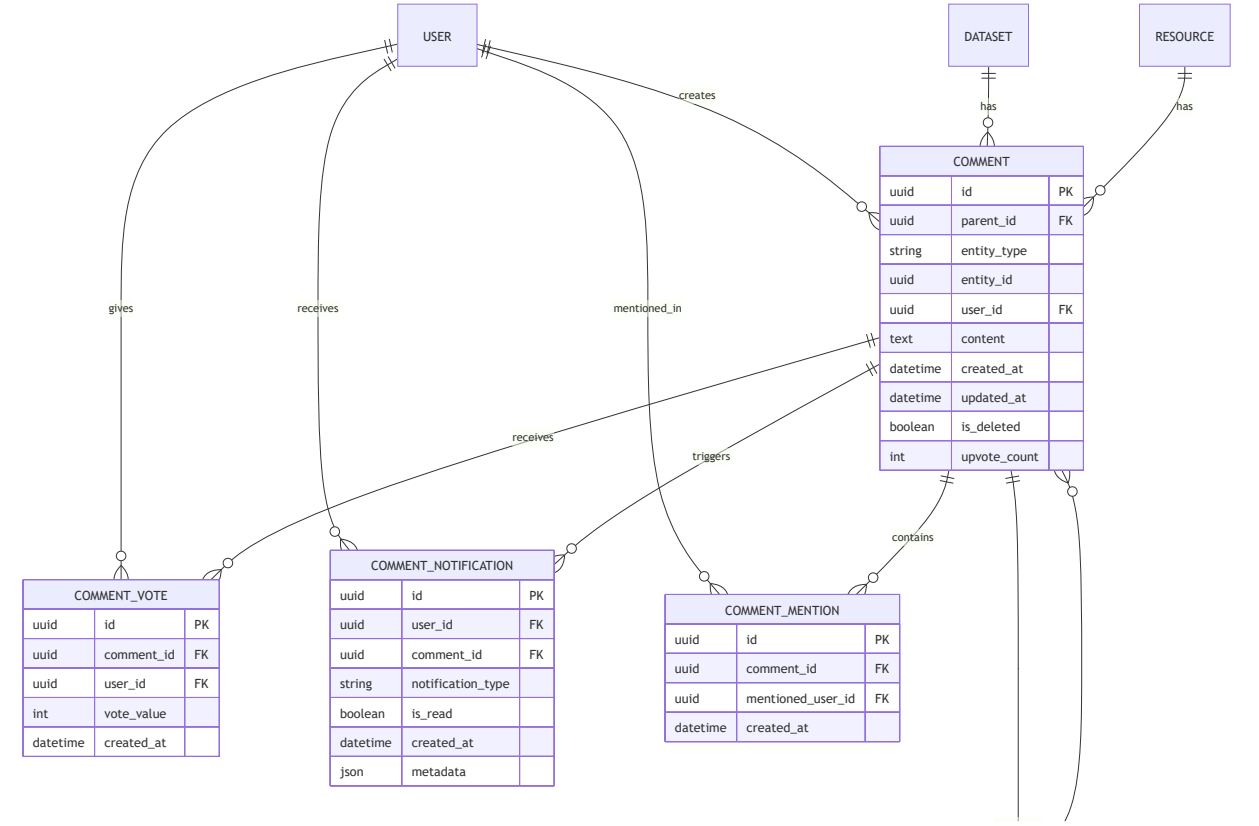
Mermaid

```
ckanext-comments/
├── ckanext/
│   └── comments/
│       ├── __init__.py
│       ├── plugin.py
│       ├── model/
│       │   ├── __init__.py
│       │   ├── comment.py
│       │   ├── vote.py
│       │   ├── notification.py
│       │   └── mention.py
│       ├── logic/
│       │   ├── __init__.py
│       │   ├── action/
│       │   │   ├── __init__.py
│       │   │   ├── create.py
│       │   │   ├── update.py
│       │   │   ├── delete.py
│       │   │   └── get.py
│       │   ├── auth/
│       │   │   ├── __init__.py
│       │   │   └── auth.py
│       │   └── validators.py
│       ├── views/
│       │   ├── __init__.py
│       │   └── comments.py
│       ├── helpers.py
│       ├── cli.py
│       ├── templates/
│       │   └── comments/
│       │       ├── snippets/
│       │       │   ├── comments_section.html
│       │       │   ├── comment_card.html
│       │       │   ├── reply_card.html
│       │       │   ├── comment_form.html
│       │       │   └── notification_panel.html
│       │       └── base/
│       │           └── notification_bell.html
│       ├── public/
│       │   ├── css/
│       │   │   └── comments.css
│       │   └── js/
│       │       └── comments.js
│       ├── migration/
│       │   └── comments/
│       │       └── versions/
│       │           └── 001_create_comments_tables.py
```

```
│       └── tests/
│           ├── __init__.py
│           ├── test_actions.py
│           ├── test_auth.py
│           └── test_views.py
├── setup.py
├── setup.cfg
├── requirements.txt
├── README.md
└── MANIFEST.in

```

# 4. Database Schema

## 4.1 Entity Relationship Diagram




## 4.2 Table Definitions

### 4.2.1 comment

SQL

```sql
CREATE TABLE comment (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    parent_id UUID REFERENCES comment(id) ON DELETE CASCADE,
    entity_type VARCHAR(50) NOT NULL,  -- 'dataset' or 'resource'
    entity_id UUID NOT NULL,
    user_id UUID NOT NULL REFERENCES "user"(id),
    content TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_deleted BOOLEAN DEFAULT FALSE,
    upvote_count INTEGER DEFAULT 0,

```

```
    CONSTRAINT valid_entity_type CHECK (entity_type IN ('dataset', 'resource'))
);

CREATE INDEX idx_comment_entity ON comment(entity_type, entity_id);
CREATE INDEX idx_comment_parent ON comment(parent_id);
CREATE INDEX idx_comment_user ON comment(user_id);
CREATE INDEX idx_comment_created ON comment(created_at DESC);

```

#### 4.2.2 comment_vote

SQL

```
CREATE TABLE comment_vote (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    comment_id UUID NOT NULL REFERENCES comment(id) ON DELETE CASCADE,
    user_id UUID NOT NULL REFERENCES "user"(id),
    vote_value INTEGER NOT NULL DEFAULT 1,  -- 1 for upvote
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT unique_vote UNIQUE (comment_id, user_id)
);

CREATE INDEX idx_vote_comment ON comment_vote(comment_id);
CREATE INDEX idx_vote_user ON comment_vote(user_id);

```

#### 4.2.3 comment_notification

SQL

```
CREATE TABLE comment_notification (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES "user"(id),
    comment_id UUID NOT NULL REFERENCES comment(id) ON DELETE CASCADE,
    notification_type VARCHAR(50) NOT NULL,
    is_read BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    metadata JSONB,

    CONSTRAINT valid_notification_type CHECK (
        notification_type IN ('mention', 'reply', 'upvote', 'new_comment')
    )
);

CREATE INDEX idx_notification_user ON comment_notification(user_id, is_read);
CREATE INDEX idx_notification_created ON comment_notification(created_at DESC);

```

#### 4.2.4 comment_mention

SQL

```
CREATE TABLE comment_mention (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    comment_id UUID NOT NULL REFERENCES comment(id) ON DELETE CASCADE,
    mentioned_user_id UUID NOT NULL REFERENCES "user"(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT unique_mention UNIQUE (comment_id, mentioned_user_id)
);

CREATE INDEX idx_mention_user ON comment_mention(mentioned_user_id);

```

# 5. API Design

## 5.1 Action Functions

### 5.1.1 Comment Actions

| | Action | Description | Auth Required |
|---|---|---|---|
| 1 | `comment_create` | Create a new comment or reply | Yes |
| 2 | `comment_update` | Update comment content | Yes (author only) |
| 3 | `comment_delete` | Soft delete a comment | Yes (author or entity owner) |
| 4 | `comment_list` | List comments for an entity | No |
| 5 | `comment_show` | Get single comment details | No |
| 6 | `comment_vote` | Upvote/remove upvote | Yes |

### 5.1.2 Notification Actions

| | Action | Description | Auth Required |
|---|---|---|---|
| 1 | `notification_list` | Get user notifications | Yes |
| 2 | `notification_mark_read` | Mark notification(s) as read | Yes |
| 3 | `notification_count` | Get unread count | Yes |

## 5.2 API Specifications

**comment_create**

Python

```python
def comment_create(context, data_dict):
    """
    Create a new comment or reply.

    :param entity_type: 'dataset' or 'resource'
    :param entity_id: ID of the dataset or resource
    :param content: Comment text content
    :param parent_id: (optional) ID of parent comment for replies

    :returns: Created comment dict
    """

```

**Request:**

JSON

```
{
    "entity_type": "dataset",
    "entity_id": "abc-123-def",
    "content": "Great dataset! @JohnSmith have you seen this?",
    "parent_id": null
}

```

Response:

JSON

```
{
    "success": true,
    "result": {
        "id": "comment-uuid",
        "entity_type": "dataset",
        "entity_id": "abc-123-def",
        "user": {
            "id": "user-uuid",
            "name": "jane_doe",
            "display_name": "Dr. Jane Doe"
        },
        "content": "Great dataset! @JohnSmith have you seen this?",
        "created_at": "2026-04-09T10:30:00Z",
        "upvote_count": 0,
        "is_upvoted": false,
        "replies": [],
        "mentions": ["john_smith"]
    }
}

```

### comment_list

Python

```
def comment_list(context, data_dict):
    """
    List comments for an entity.

    :param entity_type: 'dataset' or 'resource'
    :param entity_id: ID of the entity
    :param sort: 'top' (upvotes), 'newest', 'oldest'
    :param limit: Number of comments (default 20)
    :param offset: Pagination offset

    :returns: List of comments with nested replies
    """

```

## 5.3 REST Blueprint Routes

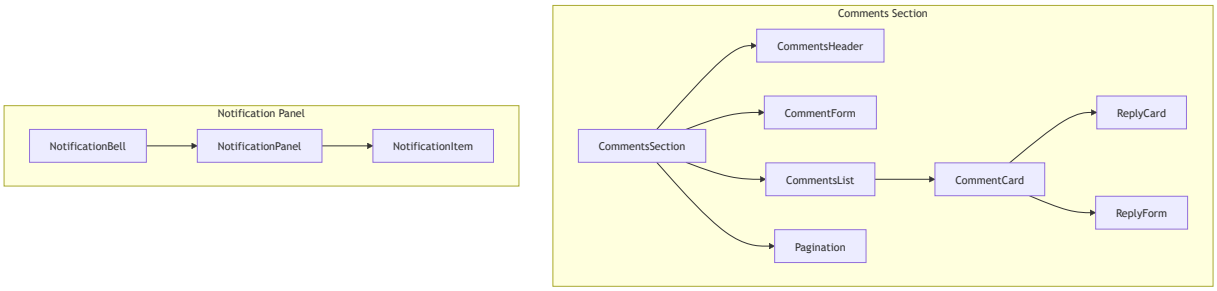
Python

```
# ckanext/comments/views/comments.py

comments_blueprint = Blueprint('comments', __name__)

# Comment endpoints
@comments_blueprint.route('/api/comments/<entity_type>/<entity_id>', methods=['GET'])
def list_comments(entity_type, entity_id):
    """GET /api/comments/dataset/{id} or /api/comments/resource/{id}"""

@comments_blueprint.route('/api/comments', methods=['POST'])
def create_comment():
```

```
    """POST /api/comments"""

@comments_blueprint.route('/api/comments/<comment_id>', methods=['PUT'])
def update_comment(comment_id):
    """PUT /api/comments/{id}"""

@comments_blueprint.route('/api/comments/<comment_id>', methods=['DELETE'])
def delete_comment(comment_id):
    """DELETE /api/comments/{id}"""

@comments_blueprint.route('/api/comments/<comment_id>/vote', methods=['POST'])
def vote_comment(comment_id):
    """POST /api/comments/{id}/vote"""

# Notification endpoints
@comments_blueprint.route('/api/notifications', methods=['GET'])
def list_notifications():
    """GET /api/notifications"""

@comments_blueprint.route('/api/notifications/count', methods=['GET'])
def notification_count():
    """GET /api/notifications/count"""

@comments_blueprint.route('/api/notifications/mark-read', methods=['POST'])
def mark_notifications_read():
    """POST /api/notifications/mark-read"""

```

## 6. Frontend Components

### 6.1 Component Hierarchy




### 6.2 Template Snippets

#### comments_section.html

Jinja

```
{% snippet "comments/snippets/comments_section.html",
   entity_type=entity_type,
   entity_id=entity_id,
   can_delete=h.comments_can_delete(entity_type, entity_id)
%}

```

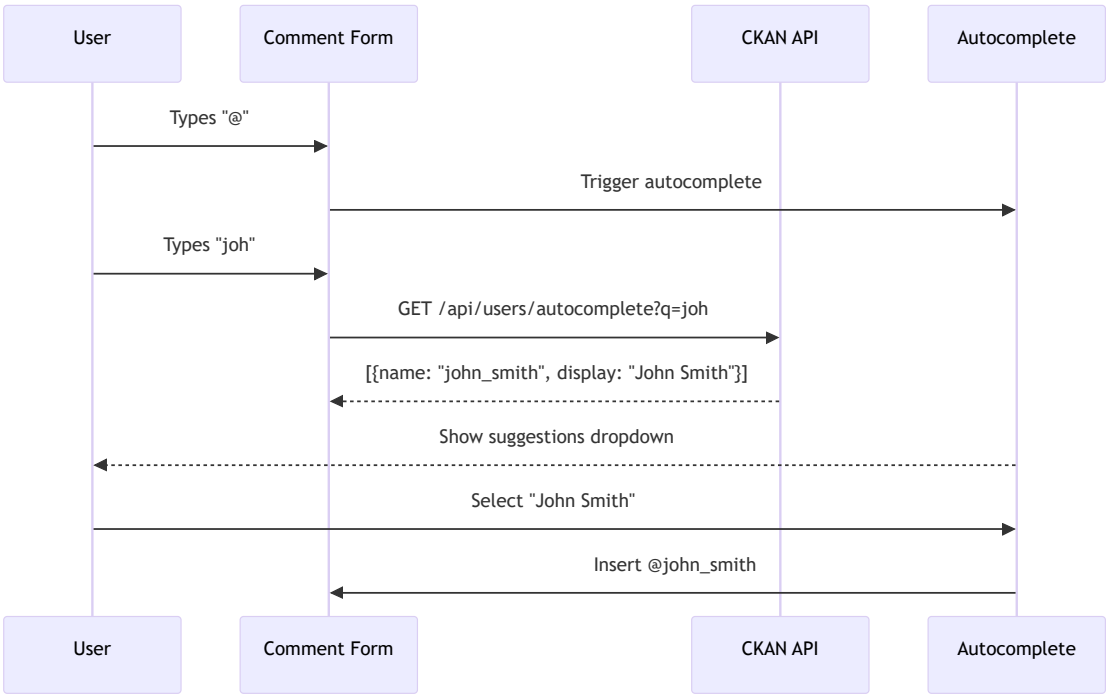
### 6.3 JavaScript Module

JavaScript

```
// ckanext/comments/public/js/comments.js
```

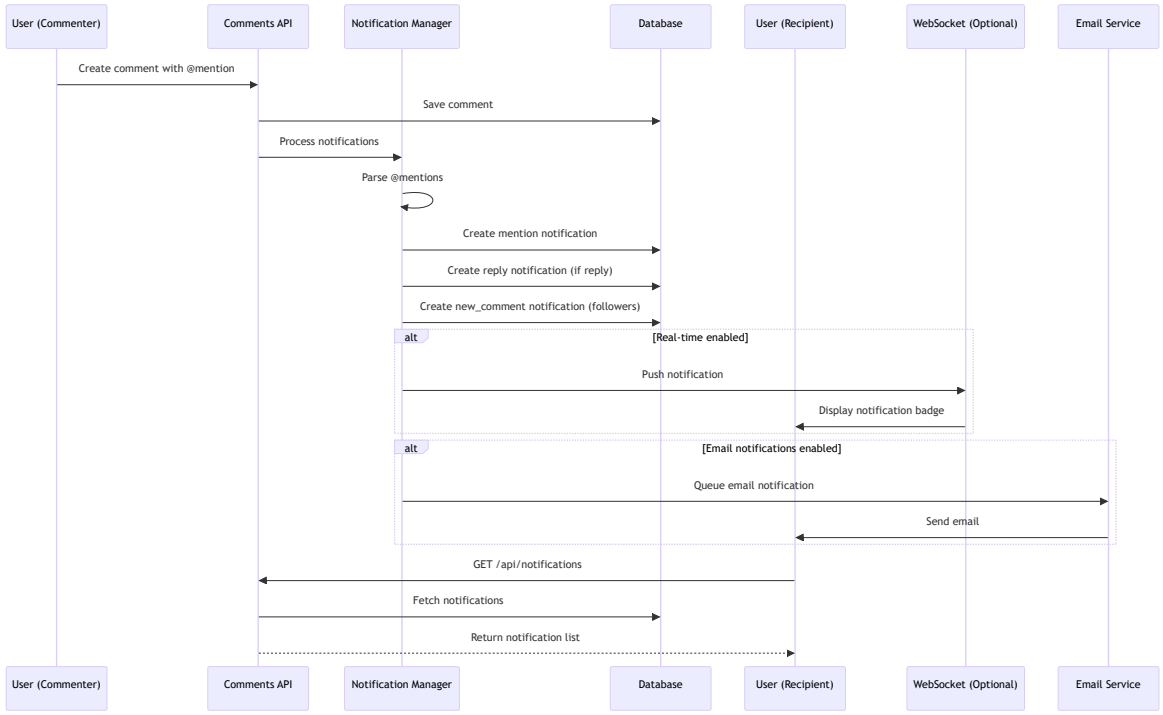
```
const CommentsModule = {
    config: {
        entityType: null,
        entityId: null,
        apiBase: '/api/comments',
        currentSort: 'top'
    },

    init(entityType, entityId) {
        this.config.entityType = entityType;
        this.config.entityId = entityId;
        this.bindEvents();
        this.loadComments();
    },

    // Comment Operations
    async loadComments(sort = 'top') {},
    async createComment(content, parentId = null) {},
    async deleteComment(commentId) {},
    async voteComment(commentId) {},

    // Notification Operations
    async loadNotifications() {},
    async markNotificationRead(notificationId) {},
    async markAllNotificationsRead() {},

    // UI Helpers
    renderComment(comment) {},
    toggleReplyForm(commentId) {},
    highlightMentions(content) {},
    showToast(message, type) {},

    // Event Bindings
    bindEvents() {}
};
```

### 6.4 Mention Autocomplete



# 7. Notification System

## 7.1 Notification Flow




## 7.2 Notification Types

| | Type | Trigger | Recipients |
|---|---|---|---|
| 1 | `mention` | @username in comment | Mentioned user |
| 2 | `reply` | Reply to a comment | Original comment author |
| 3 | `upvote` | Upvote on a comment | Comment author |
| 4 | `new_comment` | New comment on followed entity | Entity followers |

## 7.3 Notification Manager

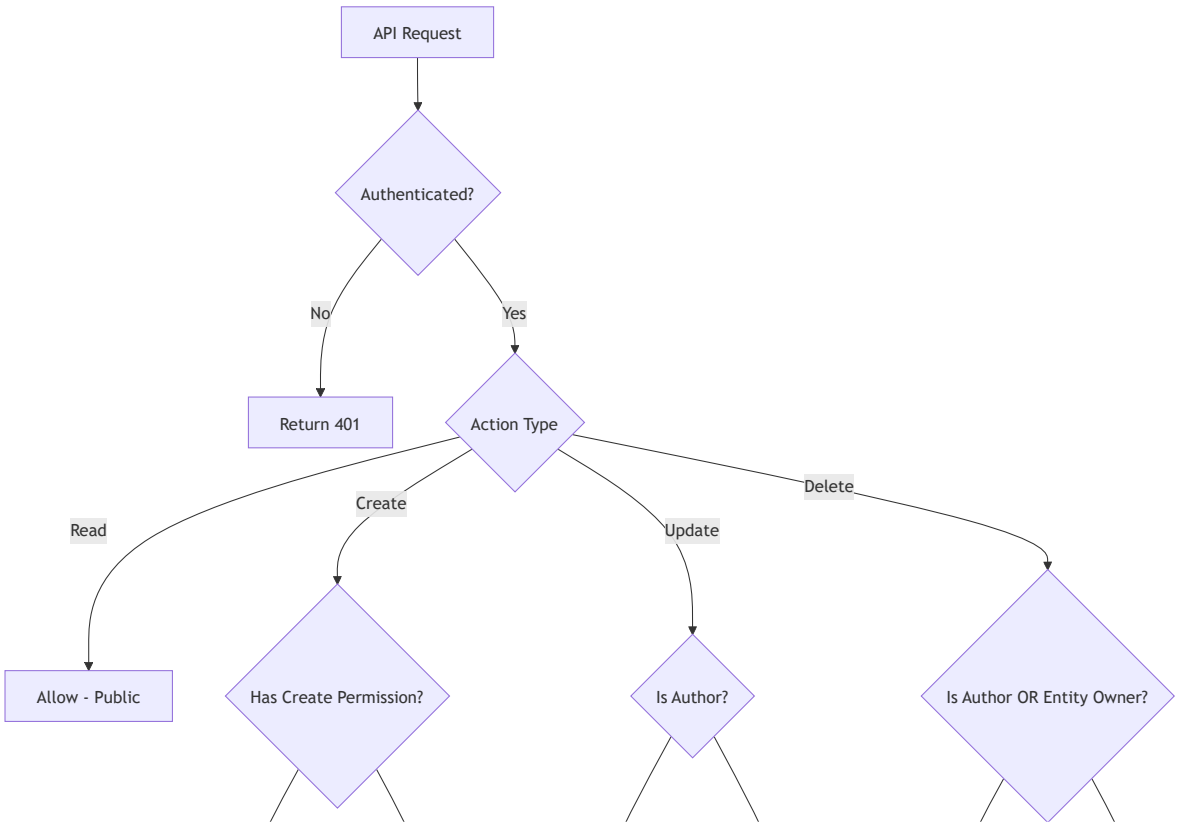
Python

```python
# ckanext/comments/lib/notification_manager.py

class NotificationManager:

    def process_comment(self, comment):
        """Process all notifications for a new comment."""
        self._notify_mentions(comment)
        self._notify_parent_author(comment)
        self._notify_entity_followers(comment)

    def process_upvote(self, vote):
        """Create upvote notification."""
        pass
```

```
14
15      def _parse_mentions(self, content):
16          """Extract @usernames from content."""
17          pattern = r'@(\w+)'
18          return re.findall(pattern, content)
19
20      def _notify_mentions(self, comment):
21          """Create notifications for mentioned users."""
22          pass
23
24      def _notify_parent_author(self, comment):
25          """Notify parent comment author of reply."""
26          pass
27
28      def _notify_entity_followers(self, comment):
29          """Notify users following the dataset/resource."""
30          pass
31
```

## 8. Security Considerations

### 8.1 Authentication & Authorization




### 8.2 Input Validation & Sanitization

Python

```
1  # Content validation
2  VALIDATORS = {
3      'content': [
4          not_empty,
5          unicode_safe,
6          max_length(10000),
7          sanitize_html,
8          rate_limit_check
```

```
    ],
    'entity_type': [
        not_empty,
        one_of(['dataset', 'resource'])
    ],
    'entity_id': [
        not_empty,
        entity_exists
    ]
}

```

## 9. Configuration Options

INI

```ini
# ckan.ini configuration options

# Enable/disable comments extension
ckanext.comments.enabled = true

# Entity types that support comments
ckanext.comments.entity_types = dataset resource

# Moderation settings
ckanext.comments.require_approval = false
ckanext.comments.allow_anonymous = false

# Rate limiting
ckanext.comments.rate_limit.comments = 10
ckanext.comments.rate_limit.window = 60

# Notification settings
ckanext.comments.notifications.email = true
ckanext.comments.notifications.realtime = false

# UI settings
ckanext.comments.default_sort = top
ckanext.comments.page_size = 20
ckanext.comments.max_reply_depth = 3

```

## 10. Integration Points

### 10.1 CKAN Core Integration

Python

```python
# plugin.py

class CommentsPlugin(plugins.SingletonPlugin):
    plugins.implements(plugins.IConfigurer)
    plugins.implements(plugins.IActions)
    plugins.implements(plugins.IAuthFunctions)
    plugins.implements(plugins.IBlueprint)
    plugins.implements(plugins.ITemplateHelpers)

    # IConfigurer
    def update_config(self, config):
        toolkit.add_template_directory(config, 'templates')
        toolkit.add_public_directory(config, 'public')
        toolkit.add_resource('public', 'comments')

    # IActions
    def get_actions(self):
        return {
```

```
            'comment_create': action.comment_create,
            'comment_update': action.comment_update,
            'comment_delete': action.comment_delete,
            'comment_list': action.comment_list,
            'comment_vote': action.comment_vote,
            'notification_list': action.notification_list,
            'notification_mark_read': action.notification_mark_read,
        }

    # IAuthFunctions
    def get_auth_functions(self):
        return {
            'comment_create': auth.comment_create,
            'comment_update': auth.comment_update,
            'comment_delete': auth.comment_delete,
            'comment_vote': auth.comment_vote,
        }

    # IBlueprint
    def get_blueprint(self):
        return [comments_blueprint]

    # ITemplateHelpers
    def get_helpers(self):
        return {
            'comments_get_count': helpers.get_comment_count,
            'comments_can_delete': helpers.can_delete_comment,
            'comments_format_mention': helpers.format_mention,
        }

```

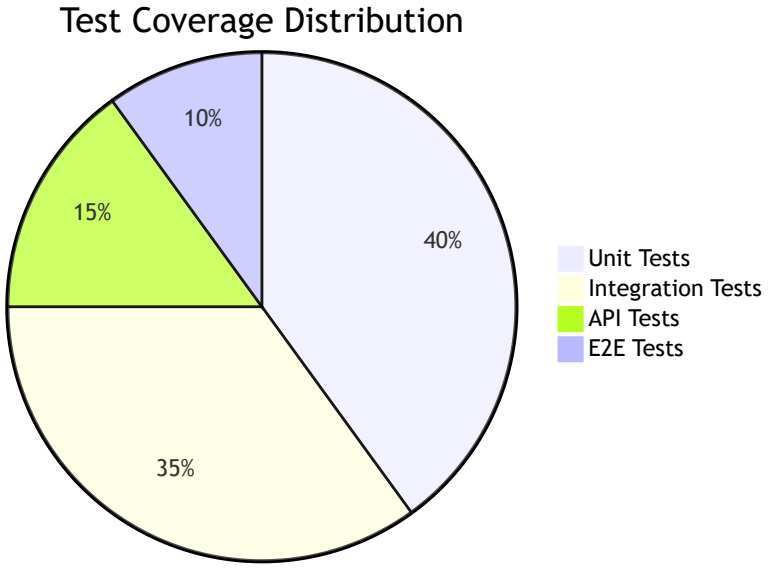
### 10.2 Template Override Points

Jinja

```
{# Override in theme to include comments #}

{# dataset/read.html #}
{% block primary_content_inner %}
    {{ super() }}
    {% snippet "comments/snippets/comments_section.html",
       entity_type='dataset',
       entity_id=pkg.id
    %}
{% endblock %}

{# resource/read.html #}
{% block resource_content %}
    {{ super() }}
    {% snippet "comments/snippets/comments_section.html",
       entity_type='resource',
       entity_id=res.id
    %}
{% endblock %}

{# base.html - notification bell #}
{% block header_account %}
    {% snippet "comments/base/notification_bell.html" %}
    {{ super() }}
{% endblock %}

```

## 11. Testing Strategy

### 11.1 Test Categories




### 11.2 Test Cases

#### Unit Tests

- Model validation tests
- Mention parsing tests
- Notification manager tests
- Permission logic tests

#### Integration Tests

- Comment CRUD operations
- Vote operations
- Notification creation flow
- Database constraint tests

#### API Tests

- Endpoint response validation
- Authentication/Authorization
- Rate limiting
- Error handling

#### E2E Tests (Cypress)

- Comment creation flow
- Reply flow
- Upvote interaction
- Notification panel interaction
- @mention autocomplete

## 12. Deployment

### 12.1 Installation

```shell
# Install extension
pip install -e git+https://github.com/uel/ckanext-comments.git#egg=ckanext-comments

# Or from source
cd ckanext-comments
pip install -e .

# Initialize database tables
ckan -c /etc/ckan/default/ckan.ini comments init-db

```

### 12.2 Database Migration

```shell
# Run migrations
ckan -c /etc/ckan/default/ckan.ini db upgrade -p comments

```

### 12.3 Configuration

```ini
# Add to ckan.ini
ckan.plugins = ... comments

# Configure options
ckanext.comments.enabled = true
ckanext.comments.notifications.email = true

```

## Appendix A: Glossary

| | Term | Definition |
|---|---|---|
| 1 | Entity | A CKAN object (dataset, resource) that can have comments |
| 2 | Thread | A comment and all its nested replies |
| 3 | Mention | @username reference in comment content |
| 4 | Upvote | Positive reaction to a comment |

## Appendix B: References

- CKAN Extension Development Guide: https://docs.ckan.org/en/latest/extensions/
- CKAN Action API: https://docs.ckan.org/en/latest/api/
- SQLAlchemy Documentation: https://docs.sqlalchemy.org/