
Feed & Post Management Feature Breakdown

AutoNateAI Learning Hub

Last Updated: February 13, 2026

Version 1.0 | Status: In Development

Document Purpose

This document provides a comprehensive technical and product breakdown of the Feed & Post Management features being built for the AutoNateAI Learning Hub. It covers the architecture, user experience decisions, file-level changes, and the rationale for how each piece is tailored specifically for a coding education platform rather than a generic social feed.

Table of Contents

| | |
|--|--|
| 1. Executive Summary | |
| 2. Why a Social Feed for AutoNateAI? | |
| 3. Feature Overview & Status | |
| 4. Post Card 3-Dot Menu - Architecture | |
| 5. Feed & Post Settings - Architecture | |
| 6. File Change Map | |
| 7. Data Model & Firestore Schema | |
| 8. UI/UX Design Decisions | |
| 9. Security Considerations | |
| 10. Rollout Checklist | |

1. Executive Summary

The Feed & Post Management update introduces two major capabilities to the AutoNateAI Learning Hub social feed:

- > **Post Actions Menu:** A traditional 3-dot dropdown on every post card giving users contextual actions - Edit and Delete for their own posts, plus Copy Link, Report, and Mute Author for community moderation.
- > **Feed & Post Settings:** A new section in the Settings page allowing users to configure their feed experience - display preferences, post defaults, and feed privacy controls.

Both features follow the platform's existing glass-card morphism design language and are built with the same Firebase + vanilla JS service architecture used across the dashboard. Settings auto-save to Firestore with debounced writes, and the post menu uses event delegation for reliable behavior across dynamically rendered content.

2. Why a Social Feed for AutoNateAI?

AutoNateAI is not a generic social platform - it is a coding education hub. The feed and post management features are purpose-built to serve learners, not casual social users. Here is why each piece matters for this specific context:

Learning Through Sharing

When students share code snippets, project showcases, and milestone achievements on the feed, they reinforce their own learning through articulation. The ability to edit posts lets them refine their explanations - a core learning behavior. Unlike Twitter or Instagram where posts are throwaway, learning content benefits from iteration.

Safe Learning Environment

The Report and Mute features are critical for maintaining a safe educational space. Coding communities can attract spam, off-topic content, or discouraging comments. Giving learners control over what they see keeps the feed focused on growth. The Mute Author feature is especially important - it lets a student quietly filter out content that is not relevant to their learning journey without confrontation.

Ownership and Accountability

Delete Post gives students agency over their content. A beginner might post code they later realize has issues - the ability to remove it reduces anxiety around sharing early work. This lowers the barrier to participation, which is essential for a learning community where many users are posting code for the first time.

Personalized Feed Experience

The Feed Settings in the Settings page let users tailor the feed to their learning style. Some learners prefer seeing trending community content for inspiration; others want to follow specific peers working on similar courses. The Show Code Snippets toggle is AutoNateAI-specific - it lets users who are on mobile or prefer prose-only content collapse inline code blocks for a cleaner reading experience.

Privacy for Learners

Many AutoNateAI users are career changers or beginners who may not want their learning activity publicly visible. The Feed Privacy settings (Public Profile, Show in Leaderboard, Allow Mentions) give users control over their visibility. This is especially important for users from partner organizations like Endless Opportunities Foundation, where participants may prefer to learn privately.

3. Feature Overview & Status

Post Card 3-Dot Dropdown Menu

- HTML Template & Conditional Rendering [COMPLETE]
- CSS Styling (Dark + Light Theme) [COMPLETE]
- Event Delegation Click Handling [COMPLETE]
- Edit Post Modal (create, populate, save) [COMPLETE]
- Delete Post (confirm, remove, Firestore sync) [COMPLETE]
- Copy Link to Clipboard [COMPLETE]
- Report Post (toast feedback) [COMPLETE]
- Mute Author (toast feedback) [COMPLETE]
- Firestore-backed mute/report persistence [PLANNED]

Feed & Post Settings (Settings Page)

- Feed Preferences UI (3 controls) [COMPLETE]
- Post Defaults UI (3 controls) [COMPLETE]
- Feed Privacy UI (3 controls) [COMPLETE]
- Load settings from Firestore [COMPLETE]
- Auto-save with debounce to Firestore [COMPLETE]
- Feed page reads settings on load [PLANNED]

4. Post Card 3-Dot Menu - Architecture

Template Structure

The dropdown menu is rendered inside each post card as part of the `renderPostCard()` template literal in `feed.html`. It uses conditional rendering based on post ownership:

```
<div class="post-card__menu-wrapper">
  <button class="post-card__menu-btn">...</button>
  <div class="post-card__dropdown" id="dropdown-{postId}">
    // Owner-only: Edit Post, Delete Post, divider
    // All users: Copy Link, Report Post
    // Non-owner only: Mute Author
  </div>
</div>
```

The conditional logic compares `post.authorId` against `this.currentUser?.uid`. When the logged-in user owns the post, they see Edit and Delete at the top with a divider. Copy Link and Report always appear. Mute Author only shows on other users' posts since muting yourself is not meaningful.

CSS Visibility Strategy

The dropdown uses `display:none` by default and `display:block` when the `.show` class is added. This was chosen over an opacity-based approach because `opacity:0` with `pointer-events:none` was found to still render child elements visually in some configurations. The `display:none` approach completely removes elements from the render tree.

```
.post-card__dropdown {
  display: none;           /* Hidden by default */
  position: absolute;      /* Relative to wrapper */
  top: 100%; right: 0;
  min-width: 180px;
  z-index: 100;
}
.post-card__dropdown.show {
  display: block;          /* Visible when toggled */
}
```

Event Delegation Pattern

Rather than binding click handlers to each individual menu button and dropdown item (which breaks when posts are dynamically re-rendered), the implementation uses event delegation on the feed container (`#feed-posts`). A single click listener uses `.closest()` to determine which element was clicked:

```
feedContainer.addEventListener("click", (e) => {
  const menuBtn = e.target.closest(".post-card__menu-btn");
  if (menuBtn) { /* toggle dropdown */ return; }

  const item = e.target.closest(".post-card__dropdown-item");
  if (item) { /* dispatch action */ return; }
});
```

This pattern is essential because `FeedPage.renderPosts()` re-renders the entire post list, destroying and recreating DOM elements. Event delegation survives re-renders automatically.

Edit Post Flow

1. User clicks Edit Post in the dropdown.
2. `handleMenuAction` dispatches to `openEditPost(postId)`.
3. A modal overlay is created (or reused if already exists) with a textarea.
4. The textarea is populated with the post's current body text.
5. On save: local `posts` array updated optimistically, `FeedService.updatePost()` writes to Firestore, `renderPosts()` refreshes the UI.
6. The save button is cloned before rebinding to prevent duplicate listeners.

Delete Post Flow

1. User clicks Delete Post in the dropdown.
2. A native `confirm()` dialog asks for confirmation.
3. On confirm: post removed from local array, `renderPosts()` refreshes UI, `FeedService.deletePost()` removes it from Firestore asynchronously.
4. A toast notification confirms the deletion.

5. Feed & Post Settings - Architecture

Settings Page Layout

The Feed & Posts section is added to settings.html as the fourth section, after Account Settings, Notification Preferences, and Appearance. It uses the same glass-card design pattern and is split into three cards within a two-column grid:

- > Feed Preferences (left): Show Code Snippets, Auto-play Milestones, Default Feed View
- > Post Defaults (right): Default Post Type, Allow Comments, Show Activity Status
- > Feed Privacy (full width): Public Profile, Show in Leaderboard, Allow Mentions

Settings Controls

The following 9 controls are available:

```
#feed-show-code (Toggle, default: true)
Show/hide inline code blocks in feed posts

#feed-autoplay-milestones (Toggle, default: true)
Auto-play milestone achievement animations

#feed-default-view (Select, default: trending)
Default tab: Trending / Latest / Following

#post-default-type (Select, default: status)
Pre-selected post type when composing

#post-allow-comments (Toggle, default: true)
Allow comments on your posts by default

#post-show-activity (Toggle, default: true)
Show online/active status to other users

#privacy-public-profile (Toggle, default: true)
Allow anyone to view your profile and posts

#privacy-show-leaderboard (Toggle, default: true)
Include your stats on the leaderboard

#privacy-allow-mentions (Toggle, default: true)
Let others tag you with @mention
```

Load / Save Architecture

Settings are loaded from and saved to the Firestore user document under a settings map. The SettingsPage controller has two methods:

```
loadFeedSettings()
- Reads users/{uid} document from Firestore
- Extracts doc.data().settings.* fields
- Populates each toggle/select with stored value or default

setupFeedSettingsSave()
- Attaches 'change' listener to all 9 controls
- Debounces writes by 500ms
- Writes all 9 values as dot-notation keys
- Shows a toast notification on save
```


6. File Change Map

The following files were modified to implement this feature set:

`courses/dashboard/feed.html`

- > Post card template: Added `.post-card__menu-wrapper` with conditional dropdown
- > `bindPostInteractions()`: Replaced per-element listeners with event delegation
- > Added `handleMenuAction()` dispatcher for edit/delete/copy-link/report/mute
- > Added `openEditPost()` with modal creation, textarea, optimistic save
- > Added `confirmDeletePost()` with confirm dialog, local removal, Firestore delete

`courses/shared/css/feed.css`

- > Added `.post-card__menu-wrapper`, `.post-card__dropdown` (`display:none/block`)
- > Added `.post-card__dropdown-item` with hover states and `--danger` variant
- > Added `.post-edit-overlay` and `.post-edit-modal` styles
- > Added `[data-theme='light']` variants for dropdown and edit modal

`courses/dashboard/settings.html`

- > Added Feed & Posts section with 3 glass-cards
- > Added 9 controls: 6 toggles + 3 selects with descriptions
- > Added `loadFeedSettings()` and `setupFeedSettingsSave()` to controller

`courses/dashboard/{8 sidebar pages}`

- > Added Settings sidebar link to: `index.html`, `courses.html`, `challenges.html`, `progress.html`, `achievements.html`, `notes.html`, `leaderboard.html`

7. Data Model & Firestore Schema

User Settings (users/{uid})

All feed and post settings are stored as flat keys under the settings map in the user document. This avoids subcollection overhead and allows atomic updates with dot-notation field paths.

```
users/{uid}: {
  displayName: "...",
  email: "...",
  settings: {
    // Existing
    theme: "dark",
    fontSize: "medium",
    accentColor: "#7986cb",

    // NEW - Feed Preferences
    feedShowCode: true,
    feedAutoplayMilestones: true,
    feedDefaultView: "trending",

    // NEW - Post Defaults
    postDefaultType: "status",
    postAllowComments: true,
    postShowActivity: true,

    // NEW - Feed Privacy
    privacyPublicProfile: true,
    privacyShowLeaderboard: true,
    privacyAllowMentions: true
  }
}
```

Feed Posts (feedPosts/{postId})

Post documents support the edit and delete actions:

```
feedPosts/{postId}: {
  authorId: "uid",           // Ownership check
  authorName: "...",
  bodyJson: {
    format: "plaintext",
    content: "..."         // Updated by Edit Post
  },
  type: "status",
  createdAt: Timestamp,
  updatedAt: Timestamp      // Set on edit
}

// Delete removes the entire document via
// FeedService.deletePost(postId)
```

8. UI/UX Design Decisions

Why `display:none` Over `opacity:0`

The initial implementation used `opacity:0` with `pointer-events:none` to hide the dropdown, with a CSS transition for a smooth fade-in. However, testing revealed that in certain caching scenarios, the child button elements remained visually rendered - appearing as raw unstyled text next to the 3-dot button. Switching to `display:none/display:block` completely removes elements from the render tree. The trade-off is losing the fade animation, but reliability wins.

Why Event Delegation for the Menu

The feed uses `renderPosts()` which replaces `innerHTML` of the posts container. Directly-bound click handlers are destroyed on re-render. Event delegation on the parent container (`#feed-posts`) survives re-renders, which is critical because editing or deleting a post triggers `renderPosts()`.

Why Auto-Save for Settings

Feed settings use the same auto-save pattern as Notification Preferences. When a user toggles a switch or changes a select, a 500ms debounced write sends all 9 values to Firestore. This matches the existing UX, toggle switches imply immediate effect, and it reduces friction for multi-setting adjustments.

Contextual Menu Items

The dropdown adapts based on post ownership:

Your posts: Edit Post | Delete Post | --- | Copy Link | Report Post

Others' posts: Copy Link | Report Post | Mute Author

Edit and Delete only appear on your own posts. Mute Author only appears on others' posts. Copy Link and Report are universal. The divider separates destructive actions from informational ones.

9. Security Considerations

Client-Side Ownership Check

The menu template uses `post.authorId === this.currentUser?.uid` to determine which items to render. This is a UI convenience only - not a security boundary. True enforcement must happen in Firestore Security Rules:

```
match /feedPosts/{postId} {
  allow update, delete:
    if request.auth.uid == resource.data.authorId;
  allow read:
    if request.auth != null;
  allow create:
    if request.auth != null
    && request.resource.data.authorId == request.auth.uid;
}
```

XSS Prevention in Edit Flow

When a user edits a post, the content is placed into a textarea element (which does not render HTML) and saved as plaintext. The `renderPostCard` template uses `escapeHtml()` on user-generated content to prevent script injection.

Privacy Settings Are Advisory

The Feed Privacy toggles are currently client-side preferences stored in Firestore. For them to be enforced, the feed query logic and leaderboard page must read these settings and filter accordingly. This is a planned follow-up. Until enforced server-side, these settings represent user intent but are not technically binding.

10. Rollout Checklist

Items to complete before this feature is considered production-ready:

- [x] Post 3-dot dropdown renders correctly (dark theme)
- [x] Post 3-dot dropdown renders correctly (light theme)
- [x] Edit Post modal opens, populates, and saves
- [x] Delete Post confirms and removes post
- [x] Copy Link copies URL to clipboard
- [x] Report Post shows feedback toast
- [x] Mute Author shows feedback toast
- [x] Dropdown closes on outside click
- [x] Event delegation survives post re-render
- [x] Feed Settings UI renders in Settings page
- [x] Feed Settings load from Firestore on page load
- [x] Feed Settings auto-save to Firestore on change
- [x] Settings sidebar link added to all dashboard pages
- [] Feed page reads user settings on load
- [] Firestore Security Rules enforce post ownership
- [] Report/Mute persist to Firestore collections
- [] Privacy settings enforced in feed queries
- [] End-to-end testing with multiple user accounts