

Vue + REST Polling - Notification Implementation Guide

Perfect for most Vue applications. Checks for new notifications every 15 seconds.

Prerequisites

Backend Requirements:

- Notification service running at <https://iccare.desmarttrader.com/notification>
- Authentication token (JWT) available in your app

Frontend Dependencies:

```
{  
  "axios": "^1.6.0",  
  "vue": "^3.3.0"  
}
```

Implementation Steps

Step 1: Create Composable

File: [src/composables/useNotifications.js](#)

```
import { ref, onMounted, onUnmounted } from 'vue';  
import axios from 'axios';  
  
const API_BASE_URL = 'https://iccare.desmarttrader.com/notification';  
const POLLING_INTERVAL = 15000; // 15 seconds  
  
export function useNotifications(token) {  
  const notifications = ref([]);  
  const unreadCount = ref(0);  
  const loading = ref(false);  
  const error = ref(null);  
  let intervalId = null;  
  
  // Fetch notifications from backend  
  const fetchNotifications = async () => {  
    if (!token.value) return;  
  
    try {  
      loading.value = true;  
      const response = await axios.get(` ${API_BASE_URL}/feed` , {  
        headers: { Authorization: `Bearer ${token.value}` },  
        params: {  
          limit: 10,  
          offset: 0  
        }  
      });  
      notifications.value = response.data;  
      unreadCount.value = notifications.value.filter(notification => notification.read === false).length;  
    } catch (error) {  
      error.value = error.message;  
    } finally {  
      loading.value = false;  
    }  
  };  
  
  onMounted(fetchNotifications);  
  onUnmounted(() => {  
    if (intervalId) clearInterval(intervalId);  
  });  
  
  const poll = () => {  
    if (unreadCount.value > 0) {  
      fetchNotifications();  
    }  
  };  
  
  const startPolling = () => {  
    if (!intervalId) {  
      intervalId = setInterval(poll, POLLING_INTERVAL);  
    }  
  };  
  
  const stopPolling = () => {  
    if (intervalId) {  
      clearInterval(intervalId);  
      intervalId = null;  
    }  
  };  
  
  return {  
    notifications,  
    unreadCount,  
    loading,  
    error,  
    startPolling,  
    stopPolling  
  };  
}
```

```
        limit: 50,
        unread_only: false
    });
});

notifications.value = response.data;
unreadCount.value = response.data.filter(n => !n.read).length;
error.value = null;
} catch (err) {
    console.error('Failed to fetch notifications:', err);
    error.value = err.response?.data?.detail || err.message;
} finally {
    loading.value = false;
}
};

// Mark notification as read
const markAsRead = async (notificationId) => {
    if (!token.value) return;

    try {
        await axios.post(
            `${API_BASE_URL}/feed/${notificationId}/mark-read`,
            {},
            { headers: { Authorization: `Bearer ${token.value}` } }
        );

        // Update local state optimistically
        notifications.value = notifications.value.map(n =>
            n.notification_id === notificationId
                ? { ...n, read: true }
                : n
        );
        unreadCount.value = Math.max(0, unreadCount.value - 1);
    } catch (err) {
        console.error('Failed to mark as read:', err);
    }
};

// Mark all as read
const markAllAsRead = async () => {
    const unreadIds = notifications.value
        .filter(n => !n.read)
        .map(n => n.notification_id);

    for (const id of unreadIds) {
        await markAsRead(id);
    }
};

// Set up polling on mount
onMounted(() => {
    // Initial fetch
    fetchNotifications();
});
```

```
// Set up polling interval
intervalId = setInterval(fetchNotifications, POLLING_INTERVAL);
};

// Clean up on unmount
onUnmounted(() => {
  if (intervalId) {
    clearInterval(intervalId);
  }
});

return {
  notifications,
  unreadCount,
  loading,
  error,
  markAsRead,
  markAllAsRead,
  refresh: fetchNotifications
};
}
```

Step 2: Create Notification Bell Component

File: [src/components/NotificationBell.vue](#)

```
<template>
<div class="notification-bell-container" ref="dropdownRef">
  <button
    class="notification-bell-button"
    @click="toggleDropdown"
    aria-label="Notifications"
    title="Notifications"
  >
    <img alt="Notification bell icon" />
    <span v-if="unreadCount > 0" class="notification-badge">
      {{ unreadCount > 99 ? '99+' : unreadCount }}
    </span>
  </button>

  <div v-if="isOpen" class="notification-dropdown">
    <div class="notification-header">
      <h3>Notifications</h3>
      <button
        v-if="unreadCount > 0"
        @click="markAllAsRead"
        class="mark-all-read"
      >
        Mark all read
      </button>
    </div>
    <div class="notification-list">
      <ul>
        <li>...
```

```
</div>

<div class="notification-list">
    <!-- Loading State -->
    <div v-if="loading && notifications.length === 0" class="notification-loading">
        Loading...
    </div>

    <!-- Error State -->
    <div v-if="error" class="notification-error">
        Failed to load notifications: {{ error }}
    </div>

    <!-- Empty State -->
    <div v-if="!loading && notifications.length === 0" class="no-notifications">
        <span class="no-notifications-icon">∅</span>
        <p>No notifications</p>
    </div>

    <!-- Notification Items -->
    <div
        v-for="notification in notifications"
        :key="notification.notification_id"
        :class="[
            'notification-item',
            { 'unread': !notification.read },
            `priority-${notification.priority}`
        ]"
        @click="handleNotificationClick(notification)"
    >
        <div class="notification-icon">
            {{ getPriorityIcon(notification.priority) }}
        </div>
        <div class="notification-content">
            <div class="notification-title">{{ notification.title }}</div>
            <div class="notification-message">{{ notification.message }}</div>
            <div class="notification-time">
                {{ formatTimestamp(notification.timestamp) }}
            </div>
        </div>
        <div v-if="!notification.read" class="unread-dot"></div>
    </div>
    </div>
</div>

<script setup>
import { ref, onMounted, onUnmounted, computed } from 'vue';
import { useNotifications } from '../composables/useNotifications';

const props = defineProps({
```

```
token: {
  type: String,
  required: true
});
});

const isOpen = ref(false);
const dropdownRef = ref(null);

const tokenRef = computed(() => props.token);
const {
  notifications,
  unreadCount,
  loading,
  error,
  markAsRead,
  markAllAsRead
} = useNotifications(tokenRef);

const toggleDropdown = () => {
  isOpen.value = !isOpen.value;
};

const handleNotificationClick = (notification) => {
  markAsRead(notification.notification_id);

  // Navigate to relevant page if action_url exists
  if (notification.action_url) {
    window.location.href = notification.action_url;
  }

  isOpen.value = false;
};

const getPriorityIcon = (priority) => {
  switch (priority) {
    case 'urgent': return '⚠';
    case 'high': return '⚠';
    case 'normal': return '🟡';
    default: return '🔗';
  }
};

const formatTimestamp = (timestamp) => {
  const date = new Date(timestamp);
  const now = new Date();
  const diffMs = now - date;
  const diffMins = Math.floor(diffMs / 60000);

  if (diffMins < 1) return 'Just now';
  if (diffMins < 60) return `${diffMins}m ago`;
  if (diffMins < 1440) return `${Math.floor(diffMins / 60)}h ago`;
  return date.toLocaleDateString();
};
```

```
// Close dropdown when clicking outside
const handleClickOutside = (event) => {
  if (dropdownRef.value && !dropdownRef.value.contains(event.target)) {
    isOpen.value = false;
  }
};

onMounted(() => {
  document.addEventListener('mousedown', handleClickOutside);
});

onUnmounted(() => {
  document.removeEventListener('mousedown', handleClickOutside);
});
</script>

<style scoped>
/* Container */
.notification-bell-container {
  position: relative;
  display: inline-block;
}

/* Bell Button */
.notification-bell-button {
  position: relative;
  background: none;
  border: none;
  font-size: 24px;
  cursor: pointer;
  padding: 8px 12px;
  border-radius: 8px;
  transition: background-color 0.2s;
}

.notification-bell-button:hover {
  background-color: rgba(0, 0, 0, 0.05);
}

/* Badge */
.notification-badge {
  position: absolute;
  top: 4px;
  right: 4px;
  background: #ff4444;
  color: white;
  border-radius: 10px;
  padding: 2px 6px;
  font-size: 11px;
  font-weight: bold;
  min-width: 18px;
  text-align: center;
}
```

```
/* Dropdown */
.notification-dropdown {
  position: absolute;
  right: 0;
  top: calc(100% + 8px);
  width: 420px;
  max-height: 600px;
  background: white;
  border: 1px solid #e0e0e0;
  border-radius: 12px;
  box-shadow: 0 8px 24px rgba(0, 0, 0, 0.12);
  z-index: 1000;
  overflow: hidden;
}

/* Header */
.notification-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 16px 20px;
  border-bottom: 1px solid #f0f0f0;
  background: #fafafa;
}

.notification-header h3 {
  margin: 0;
  font-size: 18px;
  font-weight: 600;
  color: #333;
}

.mark-all-read {
  background: none;
  border: none;
  color: #007bff;
  cursor: pointer;
  font-size: 14px;
  font-weight: 500;
  padding: 4px 8px;
  border-radius: 4px;
  transition: background-color 0.2s;
}

.mark-all-read:hover {
  background-color: rgba(0, 123, 255, 0.1);
}

/* Notification List */
.notification-list {
  max-height: 500px;
  overflow-y: auto;
}
```

```
/* Notification Item */
.notification-item {
  display: flex;
  align-items: flex-start;
  gap: 12px;
  padding: 16px 20px;
  border-bottom: 1px solid #f5f5f5;
  cursor: pointer;
  transition: background-color 0.2s;
  position: relative;
}

.notification-item:hover {
  background: #f8f9fa;
}

.notification-item.unread {
  background: #e3f2fd;
}

.notification-item.unread:hover {
  background: #d1e7f9;
}

/* Priority Colors */
.notification-item.priority-urgent {
  border-left: 4px solid #ff0000;
}

.notification-item.priority-high {
  border-left: 4px solid #ff9800;
}

.notification-item.priority-normal {
  border-left: 4px solid #2196f3;
}

/* Notification Icon */
.notification-icon {
  font-size: 24px;
  flex-shrink: 0;
  margin-top: 2px;
}

/* Notification Content */
.notification-content {
  flex: 1;
  min-width: 0;
}

.notification-title {
  font-weight: 600;
  margin-bottom: 4px;
```

```
color: #333;
font-size: 14px;
}

.notification-message {
  font-size: 13px;
  color: #666;
  margin-bottom: 6px;
  line-height: 1.4;
}

.notification-time {
  font-size: 12px;
  color: #999;
}

/* Unread Dot */
.unread-dot {
  width: 8px;
  height: 8px;
  background: #007bff;
  border-radius: 50%;
  flex-shrink: 0;
  margin-top: 8px;
}

/* Empty State */
.no-notifications {
  padding: 60px 20px;
  text-align: center;
  color: #999;
}

.no-notifications-icon {
  font-size: 48px;
  display: block;
  margin-bottom: 12px;
}

/* Loading & Error States */
.notification-loading {
  padding: 40px 20px;
  text-align: center;
  color: #666;
}

.notification-error {
  padding: 20px;
  background: #ffff3cd;
  border: 1px solid #ffc107;
  border-radius: 4px;
  margin: 12px;
  color: #856404;
  font-size: 14px;
}
```

```
}
```

```
</style>
```

Step 3: Use in Your App

File: `src/App.vue`

```
<template>
  <div class="app">
    <header class="app-header">
      <h1>iCCaRE System</h1>
      <nav>
        <!-- Your navigation items -->
        <NotificationBell :token="token" />
      </nav>
    </header>
    <!-- Rest of your app -->
  </div>
</template>

<script setup>
import { ref, onMounted } from 'vue';
import NotificationBell from './components/NotificationBell.vue';

const token = ref('');

onMounted(() => {
  // Get token from localStorage or your auth store
  token.value = localStorage.getItem('access_token') || '';
});
</script>
```

Features

- ⌚ Notification bell icon with unread count badge
- Dropdown with notification list
- Mark as read / Mark all as read functionality
- Automatic polling every 15 seconds
- Priority-based notification styling (urgent, high, normal)
- Responsive design
- Loading, error, and empty states

API Endpoints Used

- **GET** `/notification/feed` - Fetch notifications
- **POST** `/notification/feed/{notification_id}/mark-read` - Mark as read

Vue + REST Polling Complete!