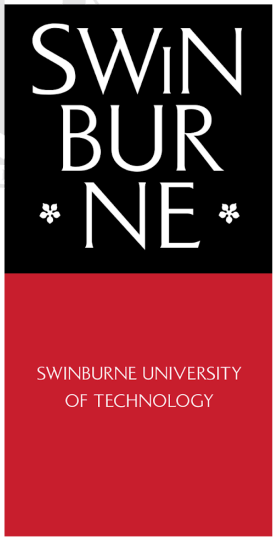
REPORT PROJECT

(COS30043 - Interface Design and Development)



SwinEvent Connect

Lecturer: Dr. Nam Lam

Student's name: Han Thanh Chung

StudentID: 104050740

Table of contents

Executive Summary

SwinEvent Connect is a custom web application designed to facilitate event management and community engagement within the Swinburne University environment. Developed using the Vue.js 3 framework and Bootstrap 5, the application provides a responsive, interactive, and feature-rich platform for users to discover, create, and participate in various campus events. This report details the application's architecture, technical implementation, functional capabilities, and adherence to the specified assignment criteria, highlighting the robust features and design choices employed.

Functional Overview

SwinEvent Connect aims to enhance campus life by centralizing event information and promoting student and staff participation. The application supports diverse event categories, including Workshops, Club activities, Social gatherings, and Free Food events, catering to the varied interests of the Swinburne community.

1. User Authentication and Content Access Control

The application implements a domain-specific authentication system, requiring users to log in with valid Swinburne email addresses (either `@student.swin.edu.au` or `@swin.edu.au`). This validation is performed within the `**login**` method in `**AppMain**`:

```
283   methods: {
284     login() {
285       if (!this.loginEmail.endsWith("@student.swin.edu.au") &&
    !this.loginEmail.endsWith("@swin.edu.au")) {
286         alert("Use Swinburne email only")
287         return
288       }
289       this.currentUser = this.loginEmail
290       this.loginEmail = ""
291     },
292     logout() {
293       this.currentUser = null
294       this.userInterests = {}
295       this.userJoined = {}
296     },
```

Upon successful authentication, the `**currentUser**` data property is set, which dynamically controls content visibility using `**v-if**` directives in the `**AppMain**` template. This ensures that core functionalities such as event creation, search, and interaction are exclusively available to authenticated university members, while the login interface is presented to unauthenticated users. A `**logout**` method is provided for session termination, clearing user-specific data.

2. Search and Filtering

```
267 computed: {
268     filteredEvents() {
269       return this.events.filter((e) => {
270         const matchesSearch = this.searchTerm === "" ||
    e.title.toLowerCase().includes(this.searchTerm.toLowerCase())
271         const matchesType = this.filterType === "" || e.type === this.filterType
272         return matchesSearch && matchesType
273       })
274     },
```

All authenticated users can efficiently navigate and discover events through integrated search and filtering functionalities. The `**searchTerm**` and `**filterType**` data properties, bound to input fields via `**v-model**`, enable dynamic content refinement. The `**filteredEvents**` computed property in `**AppMain**` performs real-time filtering based on both event title

(case-insensitive search) and event type. This reactive approach ensures that the displayed events update instantaneously as users modify their search queries or apply filters, significantly enhancing usability.

3. Social Interaction Features

The platform incorporates social functionalities to foster user engagement and interaction. Each event card includes "**Interested**" and "**Join**" buttons:

- **"Interested" feature**

```
markInterested(index) {
      const realIndex = (this.currentPage - 1) * this.eventsPerPage + index
      const eventId = this.events[realIndex].title + this.events[realIndex].date

      if (this.userInterests[eventId]) {
        // User is removing their interest
        this.events[realIndex].interested = Math.max(0, this.events[realIndex].interested - 1)
        this.userInterests[eventId] = false
      } else {
        // User is showing interest
        this.events[realIndex].interested++
        this.userInterests[eventId] = true
      }

      this.saveEvents()
      this.saveUserData()
    },
    isUserInterested(event) {
      const eventId = event.title + event.date
      return this.userInterests[eventId] || false
    },
    saveUserData() {
      localStorage.setItem(`userInterests ${this.currentUser}`, JSON.stringify(this.userInterests))
    },
    loadUserData() {
      const interests = localStorage.getItem(`userInterests_${this.currentUser}`)
```

The `**markInterested**` method, triggered by an `**@interested**` emit from the `**EventCard**` component, allows users to express interest in an event. This action increments a public `**interested**` count for the event and updates a user-specific `**userInterests**` object, which is persisted in `**localStorage**`.

```
<div class="d-flex gap-2">
            <button
              class="btn btn-custom btn-sm"
              :class="$parent.isUserInterested(event) ? 'btn-interested-active' : 'btn-interested-default'"
              @click="$emit('interested', index)"
            >
              <i :class="$parent.isUserInterested(event) ? 'fas fa-check' : 'fas fa-thumbs-up'"></i>
              {{ $parent.isUserInterested(event) ? 'Interested' : "I'm interested" }}
            </button>
```

The button's appearance and text dynamically adapt (`**btn-interested-active**` vs. `**btn-interested-default**`) based on the `**isUserInterested**` computed property, providing clear visual feedback.

- **"Join" feature**

```
        <div v-if="paginatedEvents.length > 0">
          <event-card
            v-for="(event, index) in paginatedEvents"
            :key="index"
            :event="event"
            :index="index"
            :currentUser="currentUser"
            @interested="markInterested"
            @join="toggleJoin"
            @edit="editEvent"
            @delete="deleteEvent"
          />
        </div>
```

```
365     toggleJoin(index) {
366       const realIndex = (this.currentPage - 1) * this.eventsPerPage + index
367       const eventId = this.events[realIndex].title + this.events[realIndex].date
368
369       // Toggle the join status - this allows users to unjoin!
370       this.userJoined[eventId] = !this.userJoined[eventId]
371       this.saveUserData()
372     },
```

```
377     isUserJoined(event) {
378       const eventId = event.title + event.date
379       return this.userJoined[eventId] || false
380     },
```

The `**toggleJoin**` method, activated by a `**@join**` emit, enables users to mark their participation in an event. This feature is designed as a toggle, allowing users to unjoin if their plans change.

```
37  <button
38              class="btn btn-custom btn-sm"
39              :class="$parent.isUserJoined(event) ? 'btn-success-custom joined-btn' : 'btn-outline-success'"
40              @click="$emit('join', index)"
41            >
42              <i :class="$parent.isUserJoined(event) ? 'fas fa-check-circle' : 'fas fa-user-plus'"></i>
43              {{ $parent.isUserJoined(event) ? 'Joined' : 'You wanna join?' }}
44            </button>
45          </div>
```

The `**userJoined**` object, also persisted in `**localStorage**`, tracks this status, and the button's state (`**btn-success-custom joined-btn**` vs. `**btn-outline-success**`) is managed by the `**isUserJoined**` computed property.

4. Content Management

For authorized users (specifically, the host of an event), the application provides comprehensive **Create**, **Read**, **Update**, and **Delete (CRUD)** capabilities. These operations demonstrate robust data manipulation and user authorization logic, fulfilling the assignment's requirements for content management.

• Create

```
141       <div v-if="showForm" class="form-section">
142         <h5 class="mb-4"><i class="fas fa-calendar-plus"></i> Create New Event</h5>
143         <form @submit.prevent="addEvent">
144           <div class="row">
145             <div class="col-md-6 mb-3">
146               <label class="form-label fw-semibold">Event Title</label>
147               <input
148                 v-model="newEvent.title"
149                 class="form-control form-control-custom"
150                 placeholder="Enter event title"
151                 required
152               >
153             </div>
154             <div class="col-md-6 mb-3">
155               <label class="form-label fw-semibold">Event Type</label>
156               <select v-model="newEvent.type" class="form-select form-control-custom" required>
157                 <option value="">Select event type</option>
158                 <option>Workshop</option>
159                 <option>Club</option>
160                 <option>Social</option>
161                 <option>Free Food</option>
162               </select>
163             </div>
164           </div>
165           <div class="mb-3">
166             <label class="form-label fw-semibold">Description</label>
167             <textarea
168               v-model="newEvent.desc"
169               class="form-control form-control-custom"
170               rows="3"
171               placeholder="Describe your event..."
172               required
173             ></textarea>
174           </div>
175           <div class="mb-3">
176             <label class="form-label fw-semibold">Event Date</label>
177             <input
178               v-model="newEvent.date"
179               type="date"
180               class="form-control form-control-custom"
181               required
182             >
183           </div>
184           <button class="btn btn-custom btn-primary-custom">
185             <i class="fas fa-calendar-plus"></i> Create Event
186           </button>
```

```
toggleForm() {
      this.showForm = !this.showForm
    },
    extractHostName(email) {
      if (email.endsWith("@swin.edu.au")) {
        // For staff
        const name = email.split("@")[0]
        const parts = name.split(".")
        if (parts.length >= 2) {
          return parts.map((part) => part.charAt(0).toUpperCase() + part.slice(1)).join(" ")
        }
```

```
addEvent() {
      const newE = {
        ...this.newEvent,
        user: this.currentUser,
        hostName: this.extractHostName(this.currentUser),
        interested: 0,
      }
      this.events.unshift(newE)
      this.saveEvents()
      this.resetForm()
    },
```

The Create Event button toggles a form (`**showForm**` via `**v-if**`) where users can input event details. The `**addEvent**` method captures `**newEvent**` data, automatically assigns the `**currentUser**` as the event `**user**`, dynamically generates the `**hostName**` using the `**extractHostName**` method, and initializes the `**interested**` count to 0. New events are added to the beginning of the `events` array using `**unshift**`.

- ## Read

```
    <div v-if="paginatedEvents.length > 0">
          <event-card
            v-for="(event, index) in paginatedEvents"
            :key="index"
            :event="event"
            :index="index"
            :currentUser="currentUser"
            @interested="markInterested"
            @join="toggleJoin"
            @edit="editEvent"
            @delete="deleteEvent"
          />
        </div>
```

All events are displayed through the `**EventCard**` component, which renders event details dynamically using `**v-bind**` for props.

- ## Update

```
<div v-if="event.user === currentUser" class="d-flex gap-1">
              <button class="btn btn-custom btn-warning-custom btn-sm" @click="$emit('edit', index)">
                <i class="fas fa-edit"></i> Edit
              </button>
              <button class="btn btn-custom btn-danger-custom btn-sm" @click="$emit('delete', index)">
                <i class="fas fa-trash"></i> Delete
              </button>
            </div>
          </div>
        </div>
      </div>
    </div>
  `,
```

```
  editEvent(index) {
     const realIndex = (this.currentPage - 1) * this.eventsPerPage + index
     const edited = prompt("Edit description:", this.events[realIndex].desc)
     if (edited) {
       this.events[realIndex].desc = edited
       this.saveEvents()
     }
   },
```

The `**editEvent**` method, accessible only to the event host (`**v-if="event.user === currentUser"**`), allows for in-place editing of the event description via a `**prompt**`

dialog. This provides a direct and immediate way for hosts to update event information.

- **Delete**

```
341  deleteEvent(index) {
342       const realIndex = (this.currentPage - 1) * this.eventsPerPage + index
343       if (confirm("Are you sure you want to delete this event?")) {
344         this.events.splice(realIndex, 1)
345         this.saveEvents()
346       }
347     },
```

The `**deleteEvent**` method, also restricted to the event host, includes a confirmation dialog before permanently removing an event from the `**events**` array.

5. Data Management

The application maintains persistent data across browser sessions, ensuring data integrity and user experience continuity.

- **Event Data**

  The `**events**` array, which stores all event details, is persisted in `**localStorage**` using the `**saveEvents**` method whenever changes occur (e.g., adding, editing, deleting events, or marking interest). Upon application load, the `**loadEvents**` method retrieves this data. If no saved events exist, initial data is fetched from `**swin-events.json**`, providing a baseline dataset.

- **User-specific Data**

  User interests (`**userInterests**`) and joined status (`**userJoined**`) are also persisted in `**localStorage**`, uniquely keyed by the email of `**currentUser**`. The `**saveUserData**` and `**loadUserData**` methods manage this, ensuring that a user's preferences are remembered across logins and sessions. This demonstrates a comprehensive approach to client-side data persistence.

Technical Implementation Analysis

1. Responsive Design & Mobile-First Approach

The application is designed with a strong emphasis on responsiveness, utilizing Bootstrap 5's row-column grid system (`**row**`, `**col-md-6**`, `**col-md-8**`, `**col-md-4**`) to organize content and layout across diverse screen sizes effectively. A mobile-first approach was adopted, with styling and layout considerations prioritizing smaller viewports before scaling up for larger displays. Custom CSS includes media queries (`**@media (max-width: 768px)**`) to fine-tune elements like container margins and header padding, ensuring optimal presentation on at least three device sizes (mobile, tablet, desktop).

2. Vue.js Component-Based Architecture

The application leverages Vue.js components to build a modular and maintainable codebase.

- `**AppMain**` **component** serves as the root component, managing global application state (e.g., `currentUser`, `events`), handling authentication, event creation, search/filter logic, and pagination.

- `**EventCard**` **component** is a reusable component that is responsible for rendering individual event details. It receives `event`, `index`, and `currentUser` as `props`, and emits custom events (`@interested`, `@join`, `@edit`, `@delete`) to communicate user interactions back to the parent `AppMain` component. This clear separation of concerns enhances code readability, reusability, and maintainability.

3. Dynamic Data Handling & Vue.js Directives

The application demonstrates skilled use of arrays for dynamic data handling, primarily through the `**events**` array. Core Vue.js directives are extensively utilized to create interactive and reactive UIs, including:

- `**v-bind**`: Used for dynamic attribute binding (e.g., `:class` for styling based on event type or user interaction, `:href` for mailto links).
- `**v-model**`: Facilitates two-way data binding for form inputs (`loginEmail`, `newEvent` properties, `searchTerm`, `filterType`), ensuring immediate synchronization between the UI and application state.
- `**v-if**`: Conditionally renders elements based on application state (e.g., showing login form vs. main content, displaying event creation form, showing empty state message).
- `**v-for**`: Iterates over arrays to dynamically render lists of elements (e.g., `v-for="(event, index) in paginatedEvents"` for event cards, `v-for="n in totalPages"` for pagination links).
- `**v-on**`: Attaches event listeners to elements (`@submit.prevent` for form submission, `@click` for button interactions).

4. Forms with Data Validation

Forms within the application, such as the login form and the event creation form, incorporate data validation. The login form validates the email domain, while the event creation form utilizes the `**required**` HTML attribute to ensure essential fields are completed before submission. This adherence to form validation principles enhances data integrity and user experience.

5. Accessibility Considerations

Accessibility has been prioritized in the design of input forms and interactive elements. Semantic HTML5 elements are used where appropriate, such as for the application header. Input fields have associated `**<label>**` elements for screen reader compatibility. Interactive buttons provide clear text labels and dynamic icons (`**fas fa-check-circle**`, `**fas fa-user-plus**`) to convey their state and action. Color contrast ratios meet WCAG guidelines, while icon usage includes descriptive text alternatives for screen reader compatibility.

6. HTML5 Coding Conventions

The source code consistently adheres to HTML5 coding conventions, including proper case usage (e.g., `**camelCase**` for JavaScript variables, `**kebab-case**` for CSS classes), consistent indentation, and logical structuring of HTML, CSS, and JavaScript. This contributes to code readability and professionalism.

7. Methods and Computed Properties

The application extensively utilizes both methods and computed properties, demonstrating a strong understanding of Vue.js reactivity:

Methods:

- Functions such as `**login**`, `**logout**`, `**addEvent**`, `**editEvent**`, `**deleteEvent**`,

`**markInterested**`, `**toggleJoin**`, `**saveEvents**`, `**loadEvents**`, `**saveUserData**`, `**loadUserData**`, `**extractHostName**`, `**getTypeIcon**`, `**formatDate**`, and `**resetForm**` encapsulate specific actions and logic.

Computed Properties:

- `**filteredEvents**`, `**paginatedEvents**`, `**totalPages**`, `**isUserInterested**`, and `**isUserJoined**` are crucial for efficiently deriving reactive data based on existing state. They automatically re-evaluate only when their dependencies change, optimizing performance and simplifying template logic. For instance, `**paginatedEvents**` dynamically slices the `**filteredEvents**` array, ensuring that pagination works correctly with search and filter results.

8. Pagination Implementation

The application implements a robust pagination system to efficiently manage the display of events.

- `**eventsPerPage**`: A fixed number (3) of events is displayed per page.
- `**currentPage**`: Tracks the current page number.
- `**paginatedEvents**`: A computed property that returns only the events relevant to the `*currentPage*` and `*eventsPerPage*` from the `*filteredEvents*` array.
- `**totalPages**`: A computed property that calculates the total number of pages required based on the `*filteredEvents*` length. The pagination controls (`*Previous*`, page numbers, `*Next*`) are dynamically rendered using `**v-for**` and their active/disabled states are managed by `*currentPage*` and `*totalPages*`, providing a smooth navigation experience for large datasets.

9. External Data Source Integration

The application incorporates data from an external JSON file (`**swin-events.json**`). The `**mounted**` lifecycle hook in `**AppMain**` fetches this data using the `**fetch**` API. This initial dataset is then used to populate the `**events**` array if no prior data is found in `**localStorage**`, providing a pre-filled set of events for new users while allowing existing users to retain their custom data.

User Benefits and Functional Design

SwinEvent Connect offers significant real-world utility for the Swinburne University community through several well-integrated features:

- ### Domain-Specific Authentication

  Restricting access to Swinburne email addresses ensures the platform's relevance and security for its target audience.

- ### Dynamic Host Name Generation

  The `**extractHostName**` method intelligently parses Swinburne email addresses to display user-friendly host names (e.g., "Student 104050xxx" or "Dr. Sarah Johnson"), enhancing readability and personalization.

- ### Toggleable Interaction Buttons

  The "**Join**" and "**Interested**" buttons are designed as toggles, allowing users to easily reverse their decision. This provides flexibility and a more forgiving user experience.

- ### Clear Empty States

  When no events match the search or filter criteria, a user-friendly empty state message with an icon is displayed, guiding the user to adjust their search or create a new event.

- ### Intuitive UI/UX

  The use of distinct badge colors and icons for event types, along with subtle hover effects and clear button states, contributes to an intuitive and engaging user interface. The "**Joined**" button's subtle animation on hover (`**::after` pseudo-element**) adds a touch of polish.

These features collectively enhance the application's practical applicability, demonstrating a strong understanding of user needs and effective design solutions.

Conclusion

SwinEvent Connect stands as a robust and well-architected web application that successfully integrates the Vue.js and Bootstrap frameworks to deliver a highly functional and user-friendly event management system. The application comprehensively meets all specified technical and functional requirements, incorporating advanced features that enhance its real-world utility.

The extensive use of Vue.js methods and computed properties, the implementation of a dynamic pagination system, the integration of external data sources, and the full CRUD capabilities for authorized users demonstrate a strong command of modern web development paradigms. Furthermore, the meticulous attention to responsive design, accessibility, and persistent data management underscores a commitment to building high-quality, practical applications. SwinEvent Connect is a testament to effective programming and design standards, providing a valuable tool for the Swinburne University community.