# COS30043 Custom Web Application Project Report

Dan Pan

May 26, 2025

## Introduction

This report outlines the design, features and technical challenges faced in the development of **TechVision**, a fully response single-page application (SPA) built in Vue 3, Bootstrap 5, Vite and a custom PHP backend with MySQL all deployed on the mercury server.

## Main Functionality

- **User Authentication:** Secure login and registration pages, all passwords hashed and role-based access control.
- **Home Page:** landing page showing few highlights and some images.
- **About Page:** Dynamic about page, with a personalized welcome message. There is even a theme selection with live image updates as well as a simple paragraph about Techvision's mission.
- **News:** News is loaded from JSON. users are free to filter the news based on different categories like name, date, tag.
- **Posts:** Admins can create, read, update or delete posts while regular users may only view them. Both user roles can like/vote on the content. Posts uses the same pagination system as the news. Posts are stored on the MySQL backend and fetched using the php api.
- **Kanban Board:** A kanban board with different columns, like todo, in progress, done. All stored on MySQL backend, with card tagging. Only logged in users may access this.
- **Validated Application Form:** Live error messages with RegEx checking against the inputs.
- **Accessibility and Responsiveness:** All modules are fully accessible and adjust to three screen sizes.

## Technical Components and Tools

- **Frontend:**
  - Vue 3 (composition API, SFCs)
  - Bootstrap 5 grid & icons
  - Vue Router (SPA navigation)
  - Vite (build/dev)
  - vuedraggable (drag-and-drop Kanban)

- **Backend:**
  - PHP API (`api.php`) for CRUD actions (auth, posts, kanban)
  - Data persisted with MYSQL.
- **Other:**
  - News data in JSON

Key project configuration (`vite.config.js`):

```
import { fileURLToPath, URL } from "node:url";
import { defineConfig } from "vite";
import vue from "@vitejs/plugin-vue";
import vueDevTools from "vite-plugin-vue-devtools";
export default defineConfig(({ mode }) => {
  return {
    base: mode === "production" ? "/cos30043/s103866373/project" : "/",
    plugins: [vue(), vueDevTools()],
    resolve: {
      alias: {
        "@": fileURLToPath(new URL("./src", import.meta.url)),
      },
    },
  };
});
```

## User Authentication: Login, Register, and API Integration

Setting up the user auth was challenging. It required field validation, role selection and secure posting to the API. A lot of the complexity was the login and registration system.

### Registration: Validated, Role-based, and API-integrated

The registration form validates all the input fields (including the role selected) and submits this as json to the PHP api. Error handling is both network and on the server-side.

```
<!-- src/components/Register.vue (template, abbreviated for focus) -->
<form @submit.prevent="handleSubmit">
  <input class="form-control" v-model="form.username" required>
  ...
  <input class="form-control" type="email" v-model="form.email" required>
  ...
  <select class="form-select" v-model="form.role" required>
    <option value="user">User</option>
    <option value="admin">Admin</option>
  </select>
  <button type="submit" class="btn btn-primary w-100">Register</button>
</form>
<div v-if="message" :class="['alert', messageType, 'mt-3']">
  {{ message }}
</div>
```

The registration logic had a lot of debugging info in it. Because there were evidently a lot of bugs while making it.

```
// src/components/Register.vue (script setup, excerpt)
const handleSubmit = async () => {
  try {
    const response = await fetch(API_URL, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        action: 'register',
        ...form
      })
    });
    const rawText = await response.text();
    let data;
    try {
      data = JSON.parse(rawText);
    } catch (parseError) {
      throw new Error(`Invalid JSON response: ${rawText}`);
    }
    if (response.ok) {
      message.value = data.message || 'Registration successful!';
      messageType.value = 'alert-success';
      // Clear form
      form.username = '';
      form.email = '';
      form.password = '';
      form.role = 'user';
    } else {
      message.value = data.error || 'Registration failed';
      messageType.value = 'alert-danger';
    }
  } catch (error) {
    message.value = `Error: ${error.message || 'An error occurred. Please try again.'}`;
    messageType.value = 'alert-danger';
  }
};
```

### Login: Robust and User-Friendly

Login is also handled with validation and errors are displayed. Upon success user is returned to homepage and the navbar will contain new items that only logged in users can access

```
<!-- src/components/Login.vue (template excerpt) -->
<form @submit.prevent="login">
  <input type="text" class="form-control" v-model="form.username" required>
  <input type="password" class="form-control" v-model="form.password" required>
  <div v-if="error" class="alert alert-danger">
    {{ error }}
  </div>
  <button type="submit" class="btn btn-primary" :disabled="loading">
    <span v-if="loading" class="spinner-border spinner-border-sm me-2"></span>
    Login
  </button>
</form>
```

```vue
// src/components/Login.vue (script setup)
const login = async () => {
  error.value = ''
  loading.value = true
  try {
    const response = await fetch(getUrl('api.php'), {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        action: 'login',
        ...form.value
      })
    })
    const data = await response.json()
    if (data.success) {
      router.push('/')
    } else {
      error.value = data.error || 'Login failed'
    }
  } catch (err) {
    error.value = 'An error occurred. Please try again.'
  } finally {
    loading.value = false
  }
}
```

### Authentication Status and Navbar Updates

App needs to check constantly the auth status so the UI (navbar and routing) can update in realtime. For example:

```vue
// src/components/Navbar.vue
const checkAuthStatus = async () => {
  try {
    const response = await fetch(`${getUrl('api.php')}?action=current_user`, {
      credentials: 'include'
    })
    const data = await response.json()
    isLoggedIn.value = !!data.user
    username.value = data.user?.username || ''
    isAdmin.value = data.user?.role === 'admin'
  } catch (error) {
    isLoggedIn.value = false
    username.value = ''
    isAdmin.value = false
  }
}
```

### API Layer (PHP): Small But Essential

Backend was written in PHP. all the states are stored in this api.

```php
// public/api.php (excerpt)
header("Content-Type: application/json");
$data = json_decode(file_get_contents("php://input"), true);
```

```
// DB connection and logic here
echo json_encode(["success" => true]);
```

By dividing front end and backend, maintainability and security are respected and this is more realistic to real-world workflows and future scaling.

## Innovative Features and Unique Approaches

### 1. Role-based, Conditional Navigation and Routing

Navbar and routes update live based on authentication and role:

```
<!-- src/components/Navbar.vue -->
<ul class="navbar-nav me-auto">
  ...
  <li class="nav-item" v-if="isLoggedIn">
    <router-link class="nav-link" to="/kanban">Kanban</router-link>
  </li>
</ul>
<script setup>
const isLoggedIn = ref(false);
const isAdmin = ref(false);
const checkAuthStatus = async () => {
  const response = await fetch(`${getUrl('api.php')}?action=current_user`, {
    credentials: 'include'
  })
  const data = await response.json()
  isLoggedIn.value = !!data.user
  isAdmin.value = data.user?.role === 'admin'
}
</script>
```

### 2. Drag-and-Drop Kanban Board with Real-time Updates

Cards can be created, moved, deleted and dragged around! All while syncing to the backend.

```
<!-- src/components/Kanban.vue -->
<draggable
  v-model="todoCards"
  group="cards"
  @change="onChange"
  item-key="id"
  class="min-height"
>
  <template #item="{ element }">
    <div class="card mb-2">
      <div class="card-body">
        <h5 class="card-title">{{ element.title }}</h5>
        <span v-if="element.tag" :class="getTagClass(element.tag)" class="badge mb-2">
          {{ element.tag }}
        </span>
      </div>
    </div>
  </template>
</draggable>
```

### 3. Fully Computed Pagination and Filtering

News and post component modules are computed properties for fast filtering and paging.

```
const paginatedNews = computed(() => {
  const start = (currentPage.value - 1) * itemsPerPage
  return filteredNews.value.slice(start, start + itemsPerPage)
})
```

### 4. Live, Accessible Validation

Every form provides immediate feedback and supports keyboard navigation for screen readers.

```
function validateForm() {
  errors.firstName = /^[A-Za-z]+$/.test(form.firstName) ? '' : 'First Name must contain only letters
  ...
  errors.dob = form.dob && (new Date().getFullYear() - new Date(form.dob).getFullYear()) >= 16 ? ''
  errors.category = form.category ? '' : 'Please select a category'
  return Object.values(errors).every(e => !e)
}
```

### 5. Dynamic Personalization

The About page responds live to name input and theme selection.

```
<!-- src/components/About.vue -->
<div v-if="firstName || lastName" class="alert alert-primary mt-4">
  <h4 class="alert-heading mb-0">
    Welcome{{ firstName ? ', ' + firstName : '' }}{{ lastName ? ' ' + lastName : '' }}!
  </h4>
</div>
...
<div class="mt-4">
  <label class="form-label">Choose Your Theme</label>
  <div class="d-flex gap-3">
    <input class="form-check-input" type="radio" id="mountain" value="mountain" v-model="selectedIma
    ...
    <input class="form-check-input" type="radio" id="ocean" value="ocean" v-model="selectedImage">
  </div>
</div>
```

## Development Challenges and Solutions

### Challenge 1: Conditional Rendering and Navigation (Auth and Roles)

**Problem:** The conditional rendering was challenging, ensuring that the correct links and routes could be accessed based on if your role was an 'admin' or 'user', or maybe if the user wasn't logged in at all.

**Solution:** Used navigation guards and Navbar checks to always show/hide the correct routes, even after refresh.

```
// src/router/index.js
router.beforeEach(async (to, from, next) => {
  if (to.meta.requiresAuth) {
    try {
      const response = await fetch(`${getUrl('api.php')}?action=current_user`)
      const data = await response.json()
      if (data.user) {
        next()
      } else {
        next('/login')
      }
    } catch (error) {
      next('/login')
    }
  } else {
    next()
  }
})
```

### Challenge 2: Responsive Design Across Devices

**Problem:** The layout needed to look great and work perfectly across all device sizes.

**Solution:** Used only Bootstrap grid and responsive classes. Minimal custom CSS for minor hover/fade effects. Ensured features stack or rearrange naturally on mobile/tablet.

```
/* src/style.css */
@media (max-width: 768px) {
  .container {
    padding-left: 1rem;
    padding-right: 1rem;
  }
  h1 { font-size: 2rem; }
  h2 { font-size: 1.75rem; }
  h3 { font-size: 1.5rem; }
}
```

### Challenge 3: Efficient Pagination and Filtering

**Problem:** Needed a modular way to display large sets of news/posts and have it be searchable and filterable.

**Solution:** Built a universal Pagination component and used computed properties to filter and slice arrays in-memory.

```
const paginatedNews = computed(() => {
  const start = (currentPage.value - 1) * itemsPerPage
  return filteredNews.value.slice(start, start + itemsPerPage)
})

const filteredNews = computed(() => {
  return news.value.filter(item => {
    const matchesSearch =
      item.title.toLowerCase().includes(search.value.toLowerCase()) ||
      item.content.toLowerCase().includes(search.value.toLowerCase())
```

```
    const matchesCategory = !categoryFilter.value || item.category === categoryFilter.value
    const matchesDate = !dateFilter.value || item.date === dateFilter.value
    return matchesSearch && matchesCategory && matchesDate
  })
})
```

### Challenge 4: Robust Form Validation and Error Feedback

**Problem:** Forms (Register, Apply, Post, Card Edit) needed strong validation, clear errors, and accessible feedback.

**Solution:** By using regular expressions and Bootstrap's feedback classes, the user can clearly see error messages and logic is accurately checked.

```
// src/components/Apply.vue
function validateForm() {
  errors.firstName = /^[A-Za-z]+$/.test(form.firstName) ? '' : 'First Name must contain only letters
  ...
  errors.dob = form.dob && (new Date().getFullYear() - new Date(form.dob).getFullYear()) >= 16 ? ''
  errors.category = form.category ? '' : 'Please select a category'
  return Object.values(errors).every(e => !e)
}
```

### Challenge 5: Kanban Drag-and-Drop State Consistency

**Problem:** Ensuring column and card data is consistent after drag-and-drop (across UI, memory, and backend).

**Solution:** Used the @change event to detect drag/move, then request to update the mysql backend through php api. After backend update, the column is refreshed.

```
const onChange = async (event) => {
  if (!event.added && !event.moved) return

  const card = event.added ? event.added.element : event.moved.element
  const newColumn = event.added ? event.added.to.id : event.moved.to.id

  await fetch(getUrl('api.php'), {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    credentials: 'include',
    body: JSON.stringify({
      action: 'update_card',
      id: card.id,
      title: card.title,
      tag: card.tag,
      column: newColumn
    })
  })
}
```

### Challenge 6: Maintaining Security for Auth Actions

**Problem:** Preventing unauthorized actions (e.g. editing posts, deleting cards) if not logged in or not admin. This is important as the user shouldn't be allowed to access things they aren't authorised to access.

**Solution:** Every route that required authentication was protected with router guards, and every backend API request required user credentials (via `credentials: 'include'`). UI elements were hidden for users without permissions.

```
// src/components/Posts.vue (admin check)
const response = await fetch(`${API_URL}?action=current_user`, {
  credentials: 'include'
})
const data = await response.json()
isAdmin.value = data.user?.role === 'admin'
```

## Conclusion

This project fulfills all the technical and functional requirements. Demonstrating use of Vue 3 SFCs, Vue Router, Bootstrap and some backend knowledge to create a cohesive application.

**Live Demo:** `https://mercury.swin.edu.au/cos30043/s103866373/project/`