George Mason University

Department of Computer Science

Campus Survey Application: Vue.js with Spring Boot

SWE 642 - Software Engineering for the World Wide Web

Project 4 Report

Prepared by
Sri Bhuvan Maddipudi (G01488473)
Praneeth Naidu (G01477360)
Ankit Raut (G01476996)
Shreyas Patil (G01508612)

April 17, 2025

Contents

1 Introduction

This report documents the development of a Campus Survey Application for Project 4, using Vue.js for the frontend with a Spring Boot RESTful backend. This project builds on the knowledge gained from previous assignments and applies it to create a full-stack web application that leverages modern JavaScript frameworks and Java-based backend technologies.

1.1 Project Objectives

The main objectives of this project were to:

- Create a responsive and interactive user interface using Vue.js
- Develop a robust Spring Boot backend with RESTful Web Services
- Implement JPA/Hibernate for database operations
- Build a complete Student Survey management system
- Apply software engineering best practices throughout development
- Demonstrate proficiency in both frontend and backend technologies

1.2 Technologies Used

- **Frontend**: Vue.js 3, Vuex, Vue Router, Axios, Bootstrap 5, Font Awesome
- **Backend**: Spring Boot 2.7, JPA/Hibernate, MySQL, Java 17
- **Development Tools**: VS Code, Git, npm, Maven

2 System Architecture

The application follows a client-server architecture with clear separation of concerns:

2.1 Frontend Architecture

The Vue.js frontend is built with a component-based architecture following the Single Page Application (SPA) pattern. Key architectural elements include:

- **Vue Components**: Reusable UI elements that encapsulate HTML, CSS, and JavaScript
- **Vuex Store**: Centralized state management for application data
- **Vue Router**: Client-side routing for navigation between views
- **Axios**: HTTP client for API communication

2.2 Backend Architecture

The Spring Boot backend follows a layered architecture pattern:

- **Controller Layer**: REST endpoints for handling HTTP requests
- **Service Layer**: Business logic and data processing
- **Repository Layer**: Data access through JPA/Hibernate
- **Model Layer**: Entity classes representing the domain model

2.3 System Interaction

The frontend and backend communicate through a RESTful API:

- The Vue.js application makes HTTP requests to the Spring Boot API
- Data is exchanged in JSON format
- The backend processes requests and performs database operations
- Responses are sent back to the frontend for display

3 Implementation Details

3.1 Backend Implementation

The backend implementation was retained from Assignment 3 with minor adjustments:

3.1.1 Entity Model

The Survey entity class defines the data structure with validation annotations:

```
@Entity
@Table(name = "surveys")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Survey {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "First name is required")
    @Column(name = "first_name", nullable = false)
    private String firstName;

    // Other fields with validation annotations
```

```
16
17      @ElementCollection
18      @CollectionTable(name = "survey_likes", joinColumns =
    @JoinColumn(name = "survey_id"))
19      @Column(name = "liked_item")
20      private List<String> likes;
21
22      // Additional fields
23  }
```

Listing 1: Survey Entity Class

3.1.2 RESTful API

The REST controller provides endpoints for CRUD operations:

```
1  @RestController
2  @RequestMapping("/api/surveys")
3  @CrossOrigin(origins = "http://localhost:8080")
4  public class SurveyController {
5      private final SurveyService surveyService;
6
7      @Autowired
8      public SurveyController(SurveyService surveyService) {
9          this.surveyService = surveyService;
10     }
11
12     @GetMapping
13     public ResponseEntity<List<Survey>> getAllSurveys() {
14         List<Survey> surveys = surveyService.getAllSurveys();
15         return new ResponseEntity<>(surveys, HttpStatus.OK);
16     }
17
18     // Other CRUD endpoints
19 }
```

Listing 2: Survey Controller

3.1.3 CORS Configuration

CORS was configured to allow requests from the Vue.js frontend:

```
1  @Bean
2  public WebMvcConfigurer corsConfigurer() {
3      return new WebMvcConfigurer() {
4          @Override
5          public void addCorsMappings(CorsRegistry registry) {
6              registry.addMapping("/api/**")
7                      .allowedOrigins("http://localhost:8080")
```

```
                        .allowedMethods("GET", "POST", "PUT", "DELETE")
                        .allowCredentials(true);
            }
        };
}
```

Listing 3: CORS Configuration

3.2 Frontend Implementation

The Vue.js frontend was implemented with a focus on component reusability and state management:

3.2.1 Project Structure

The frontend follows a modular structure:

```
survey-app-vue/
          public/
          src/
              assets/
                    css/
                        styles.css
              components/
                    AlertMessage.vue
                    ConfirmDialog.vue
                    LoadingSpinner.vue
                    SurveyDetail.vue
              plugins/
                    confirmDialog.js
              router/
                    index.js
              store/
                    index.js
              views/
                    HomeView.vue
                    SurveyForm.vue
                    SurveyList.vue
              App.vue
              main.js
```

Listing 4: Frontend Directory Structure

3.2.2 Vuex Store

The Vuex store provides centralized state management:

```
import { createStore } from 'vuex'
import axios from 'axios'
```

```
3
4 const API_URL = 'http://localhost:8080/api/surveys'
5
6 export default createStore({
7   state: {
8     surveys: [],
9     loading: false,
10    error: null
11  },
12  getters: {
13    getAllSurveys: state => state.surveys,
14    // Other getters
15  },
16  mutations: {
17    SET_SURVEYS(state, surveys) {
18      state.surveys = surveys
19    },
20    // Other mutations
21  },
22  actions: {
23    async fetchSurveys({ commit }) {
24      commit('SET_LOADING', true)
25      try {
26        const response = await axios.get(API_URL)
27        commit('SET_SURVEYS', response.data)
28      } catch (error) {
29        commit('SET_ERROR', 'Failed to fetch surveys')
30      } finally {
31        commit('SET_LOADING', false)
32      }
33    },
34    // Other actions
35  }
36 })
```

Listing 5: Vuex Store Implementation

3.2.3 Survey Form Component

The SurveyForm component handles both creation and editing of surveys:

```
1 export default {
2   name: 'SurveyForm',
3   data() {
4     return {
5       survey: {
6         firstName: '',
7         lastName: '',
```

```
        // Other form fields
      },
      errors: {},
      isSubmitting: false
    }
  },
  computed: {
    isEditMode() {
      return this.$route.params.id !== undefined
    }
  },
  methods: {
    validateForm() {
      // Form validation logic
      return isValid
    },

    async submitForm() {
      if (!this.validateForm()) {
        return
      }

      this.isSubmitting = true

      try {
        if (this.isEditMode) {
          await this.updateSurvey({
            id: this.surveyId,
            survey: this.survey
          })
        } else {
          await this.createSurvey(this.survey)
        }
        this.$router.push('/surveys')
      } catch (error) {
        console.error('Error submitting survey:', error)
      } finally {
        this.isSubmitting = false
      }
    }
  }
}
```

Listing 6: Survey Form Component

3.2.4 Survey List Component

The SurveyList component displays all surveys with search and detail view:

```
export default {
  name: 'SurveyList',
  data() {
    return {
      searchQuery: '',
      selectedSurvey: null
    }
  },
  computed: {
    filteredSurveys() {
      if (!this.searchQuery.trim()) {
        return this.surveys
      }

      const query = this.searchQuery.toLowerCase()
      return this.surveys.filter(survey => {
        return survey.firstName.toLowerCase().includes(query) ||
               survey.lastName.toLowerCase().includes(query) ||
               survey.email.toLowerCase().includes(query)
      })
    }
  },
  methods: {
    async promptDelete(id) {
      const confirmed = await this.$confirm({
        title: 'Delete Survey',
        message: 'Are you sure you want to delete this survey?',
        confirmType: 'danger'
      })

      if (confirmed) {
        await this.deleteSurvey(id)
        if (this.selectedSurvey?.id === id) {
          this.selectedSurvey = null
        }
      }
    }
  }
}
```

Listing 7: Survey List Component

3.3 Reusable Components

Several reusable components were created to enhance the user experience:

3.3.1 Confirmation Dialog

A reusable dialog for confirming user actions:

```
export default {
  name: 'ConfirmDialog',
  props: {
    show: Boolean,
    title: String,
    message: String,
    confirmText: {
      type: String,
      default: 'Confirm'
    },
    confirmType: {
      type: String,
      default: 'primary'
    }
  },
  methods: {
    confirm() {
      this.$emit('confirm')
    },
    cancel() {
      this.$emit('cancel')
    }
  }
}
```

Listing 8: Confirm Dialog Component

3.3.2 Alert Message

A component for displaying informational and error messages:

```
export default {
  name: 'AlertMessage',
  props: {
    type: {
      type: String,
      default: 'info',
      validator: (value) => ['success', 'info', 'warning', 'danger
   '].includes(value)
    },
    dismissible: {
      type: Boolean,
      default: false
    }
  }
```

```
}
```

Listing 9: Alert Message Component

4 Features Implementation

4.1 Form Validation

Client-side validation was implemented to ensure data integrity:

- Required field validation
- Email format validation
- Phone number format (XXX-XXX-XXXX)
- ZIP code format (5 digits)
- Minimum selection requirements

4.2 Responsive Design

The application is fully responsive using Bootstrap 5:

- Mobile-first approach
- Responsive grid system
- Collapsible navigation
- Appropriate font sizes and spacing

4.3 User Experience Enhancements

Several features were added to improve user experience:

- Loading indicators for asynchronous operations
- Confirmation dialogs for destructive actions
- Informative error messages
- Search functionality for survey list
- Side panel for viewing survey details

5 Challenges and Solutions

5.1 CORS Configuration

Challenge: The frontend could not communicate with the backend due to CORS restrictions.

Solution: Properly configured CORS in the Spring Boot application to allow requests from the Vue.js application.

5.2 Form Validation

Challenge: Implementing comprehensive form validation without a dedicated form library.

Solution: Created a custom validation system using Vue's reactivity system and computed properties.

5.3 State Management

Challenge: Managing application state across multiple components.

Solution: Implemented Vuex store with actions, mutations, and getters to centralize state management.

5.4 Handling Complex Form Data

Challenge: Managing arrays of data like the "likes" checkbox group.

Solution: Created custom components and used v-model bindings to handle array manipulation correctly.

6 Testing

6.1 Frontend Testing

The frontend was tested through manual testing of all features:

- Form submission with valid and invalid data
- Survey listing and filtering
- Edit and delete operations
- Responsive design on various screen sizes
- Cross-browser compatibility

6.2 Backend Testing

The backend was tested using VS Code REST Client:

- Testing all API endpoints
- Validating request and response formats
- Error handling and validation
- Database persistence

6.3 Integration Testing

The full application was tested to ensure proper integration:

- End-to-end workflow testing
- Data consistency between frontend and backend
- Error handling across the stack

7 Comparison with Angular Implementation

7.1 Development Experience

- **Vue.js Advantages:**
  - Simpler learning curve
  - More concise syntax
  - Single-file components
  - Less boilerplate code
- **Angular Advantages:**
  - More opinionated structure
  - Built-in TypeScript support
  - More comprehensive tooling
  - Built-in form validation

7.2 Performance Comparison

- Vue.js typically has a smaller bundle size
- Angular provides more optimization out of the box
- Both frameworks offer good rendering performance
- Vue.js has slightly faster startup time

7.3 Code Maintainability

- Vue.js code tends to be more readable for newcomers
- Angular enforces stronger typing and structure
- Vue.js offers more flexibility in implementation
- Both frameworks support component-based architecture

8 Conclusion

8.1 Summary

The reimplementation of the Campus Survey Application using Vue.js has been successfully completed, meeting all requirements from Assignment 3. The application maintains full functionality while providing an enhanced user experience through Vue.js's reactive components and state management.

8.2 Lessons Learned

- Vue.js offers a more approachable learning curve compared to Angular
- Vuex provides effective state management for complex applications
- Component reusability is key to efficient development
- Proper backend configuration is essential for frontend integration
- Responsive design principles apply across frameworks

8.3 Future Improvements

- Add unit and integration tests with Jest
- Implement TypeScript for better type safety
- Add pagination for larger datasets
- Enhance accessibility features
- Implement more advanced filtering and sorting

9 References

1. Vue.js Documentation, `https://vuejs.org/guide/introduction.html`
2. Vuex Documentation, `https://vuex.vuejs.org/`
3. Vue Router Documentation, `https://router.vuejs.org/`
4. Spring Boot Documentation, `https://docs.spring.io/spring-boot/docs/current/reference/html/`
5. Spring Data JPA Documentation, `https://docs.spring.io/spring-data/jpa/docs/current/reference/html/`
6. Hibernate ORM Documentation, `https://hibernate.org/orm/documentation/`
7. MySQL Reference Manual, `https://dev.mysql.com/doc/refman/8.0/en/`
8. Bootstrap Documentation, `https://getbootstrap.com/docs/5.0/getting-started/introduction/`
9. Font Awesome Documentation, `https://fontawesome.com/docs`
10. Java 17 Documentation, `https://docs.oracle.com/en/java/javase/17/`
11. Axios Documentation, `https://axios-http.com/docs/intro`
12. REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces. By Mark Masse, O'Reilly Media (2011)
13. Learning Vue.js 2. By Olga Filipova, Packt Publishing (2017)
14. Pro Spring Boot 2: An Authoritative Guide to Building Microservices, Web and Enterprise Applications, and Best Practices. By Felipe Gutierrez, Apress (2019)
15. Reactive Web Applications with Spring Boot 2: Connect to Your Database, Send Messages, Add Authentication, and Develop a Complete RESTful API. By Olga Maciaszek-Sharma, Apress (2021)

A API Endpoints Documentation

B Database Schema

Method	Endpoint	Status Code	Description
GET	/api/surveys	200	Retrieve all surveys
GET	/api/surveys/{id}	200, 404	Retrieve a specific survey by ID
POST	/api/surveys	201	Create a new survey
PUT	/api/surveys/{id}	200, 404	Update an existing survey
DELETE	/api/surveys/{id}	204, 404	Delete a survey

Table 1: API Endpoints

Table	Column	Type	Description
surveys	id	BIGINT	Primary key, auto-increment
surveys	first_name	VARCHAR(100)	First name of the student
surveys	last_name	VARCHAR(100)	Last name of the student
surveys	street_address	VARCHAR(255)	Street address
surveys	city	VARCHAR(100)	City
surveys	state	VARCHAR(50)	State
surveys	zip	VARCHAR(5)	ZIP code (5 digits)
surveys	phone	VARCHAR(12)	Phone number (XXX-XXX-XXXX)
surveys	email	VARCHAR(100)	Email address
surveys	survey_date	DATE	Date the survey was taken
surveys	interest_source	VARCHAR(50)	How student became interested
surveys	recommendation_likelihood	VARCHAR(50)	Likelihood of recommendation
surveys	comments	TEXT	Additional comments
survey_likes	survey_id	BIGINT	Foreign key to surveys.id
survey_likes	liked_item	VARCHAR(50)	Item the student liked

Table 2: Database Schema