Module Code: SFDAD501

Module Name:Develop Game with Vue

Date: …../02/2023

NESA NATIONAL EXAMINATION AND SCHOOL INSPECTION AUTHORITY

END OF TERM II EXAMINATIONS MARKING GUIDE

District:Gasabo

RTQF LEVEL: III

TRADE:Software Development

MARKS: 100

DURATION: 3 HOURS

INSTRUCTIONS TO CANDIDATES:

This Exam paper is composed of Two Sections (A, B) Follow the instructions given below, and answer the indicated questions for a total of 100 marks

Section A: Eleven (12) questions, all **Compulsory 60 marks**

Section B: Attempt the four (4) questions, **40 marks**

SECTION A: Compulsory questions (60 marks)

A1.Vue.js is a progressive framework of JavaScript used to create Dynamic User Interfaces and single-page applications.

A2. Vue.js called a progressive framework because it is being changed and developed continually.

A3.Vue.js is also used to build User Interfaces and single-page applications like Angular. Nowadays, it is evolving very fast, and with time, new libraries and extensions are coming into existence.

A4. The VUE-resource is a plug-in for Vue.js. This plug-in is used with Vue.js to make web requests and handle responses, in which XHMLHttpRequests or JSONP is used.

You can use the following yarn or npm command to install VUE-resource:

```
1.          $ yarn add vue-resource
2.          $ npm install vue-resource
```

A5.You can create a new Vue instance by using the Vue function:

```
1.          var vm = new Vue({
2.            // options
3.          })
```

You have to create a new Vue instance when you want to start a Vue application.

A6. Advantages of using Vue

Very Small In Size

One of Vue.js' biggest advantages is that it is very small in size. This exciting JavaScript plug-in is only 18-21KB, so you can download and use it very easily in no time.

Easy to Understand and Use

The framework of Vue.js is very easy to understand, and it id one of the reasons for the popularity of this framework. The users can easily add Vue.js to their web project because of its simple structure and develop applications.

Flexible in nature

The flexible nature of Vue.js also makes it easy to understand for the developers of React.js, Angular.js, and any other new JavaScript framework. It provides a lot of flexibility to use virtual nodes to write HTML files, JavaScript files, and pure JavaScript files.

Components

You can create reusable custom elements in Vue.js applications.

Easy & comprehensive documentation

The documentation of Vue.js is very easy and comprehensive so that developers can develop applications or web pages, only having little knowledge about HTML.

Virtual DOM

Vue.js uses virtual DOM similar to other existing frameworks such as ReactJS, Ember, etc. Virtual DOM is a light-weight in-memory tree representation of the original HTML DOM and updated without affecting the original DOM.

Two-Way Communication

Vue.js provides two-way communications with its MVVM architecture that makes it very easy to handle HTML blocks.

A7.

Step1. Use Slots to Make Your Components Easier to Understand and More Powerful

Step 2. Organize Your Vuex Store Properly

Step 3. Use Actions to Make API Calls and Commit the Data

Step 4.Simplify Your Code Base with mapState, mapGetters, mapMutations and mapActions

Step 5. Use API Factories

A8. **How can you optimize the routing system of a Vue project**

1. Using links that lead to trustworthy resources.
1. Fast loading.
2. Mobile optimization (being mobile app accessible)
3. Data protection setup.
4. Using metadata.
5. Architecture for search bots.

A9. Each component you create in your Vue app should be registered so Vue knows where to locate its implementation when it is encountered in a template.

To register your component in another component, you should add it to the `components` property in the Vue object.

```
<template>

        <AnotherComponent /></template>

<script>import AnotherComponent from './AnotherComponent.vue'

export default {

  components: {

    AnotherComponent

  }}</script>
```

A10.Validating user inputs with vee-validate While writing custom validation logic is possible, there is already a great plugin that helps validate inputs easily and displays the corresponding errors. This plugin is vee-validate.

A11. **How can you implement robust form data handling in Vue that accounts for different user scenarios…..**

1.Input validation: Validate user input on the client-side before submitting it to the server to prevent invalid or malicious data from being processed. You can use Vue's built-in form validation library or a third-party library like Vuelidate to define validation rules and display error messages based on user input.

2.Error handling: Handle errors gracefully and provide meaningful feedback to users when data validation or submission fails. You can use Vue's error handling capabilities, such as the error boundary component or the global error handler, to catch and handle errors across different components and views.

3.Data submission: Submit form data securely and efficiently using Vue's built-in HTTP client or a third-party library like Axios or Vue Resource. Use best practices for data encoding, authentication, and encryption to protect user data and prevent unauthorized access.

4.Data persistence: Store form data locally or on the server based on the user's preferences and the application requirements. You can use Vue's built-in state management library or a third-party library like Vuex or Redux to manage the application state and ensure data consistency and integrity.

5.Data synchronization: Keep the client-side and server-side data in sync using real-time communication technologies like WebSockets or long polling. Use Vue's reactivity system or a third-party library like Vue Socket.IO to update the user interface and notify users of changes in real-time.

By following these best practices, you can implement robust form data handling in Vue that accounts for different user scenarios and minimizes the risk of data loss or corruption.

A12.Designing and implementing a robust API request system in Vue involves several steps to account for different user scenarios and optimize network performance while minimizing data transfer. Here are some best practices to follow:

1.Request retries: Implement request retries for failed API requests to improve reliability and reduce network congestion. You can use a third-party library like Axios Retry or Vue Retry to automatically retry requests based on predefined rules or user-defined configurations.

2.Error handling: Handle API request errors gracefully and provide meaningful feedback to users when data retrieval or submission fails. You can use Vue's error handling capabilities, such as the error boundary component or the global error handler, to catch and handle errors across different components and views.

3.Data caching: Cache frequently accessed API data on the client-side using browser storage or a third-party caching library like Vue Apollo or Vue Cache. This reduces the number of network requests and improves the application's responsiveness and performance.

4.Authentication: Implement secure authentication and authorization mechanisms for API requests to protect user data and prevent unauthorized access. You can use Vue's built-in authentication features or a third-party library like Auth0 or Firebase Authentication to manage user sessions and secure API endpoints.

5.Network optimization: Optimize API requests for network performance and minimize data transfer using techniques like data compression, HTTP caching, and pagination. You can use a third-party library like Lodash or Vue Paginate to implement these techniques and improve the application's scalability and efficiency.

By following these best practices, you can design and implement a robust API request system in Vue that accounts for different user scenarios and optimizes network performance while minimizing

SECTION B: Attempt any Three (3) questions (40 marks)

Answer 1.

1. Set up the Vue environment by installing the Vue CLI and creating a new project.
2. Create a new component called "GuessingGame" that will contain the logic and interface for the game.
3. Inside the GuessingGame component, define a data property called "targetNumber" that will hold a randomly generated number between 1 and 10.
4. Define another data property called "guessesLeft" that will track the number of remaining attempts the user has to guess the correct number.
5. Create a form with a text input field and a submit button that will allow the user to enter their guess.
6. Add an event listener to the form's submit button that will call a method called "checkGuess" when clicked.
7. Inside the checkGuess method, retrieve the user's guess from the input field, parse it as an integer, and compare it to the targetNumber data property.
8. If the user's guess is correct, display a success message and disable the form. If the user's guess is incorrect but there are still guesses left, display a message telling them whether their guess was too high or too low and decrement the guessesLeft data property. If the user's guess is incorrect and there are no guesses left, display a failure message and disable the form.
9. Add a button to reset the game and generate a new random targetNumber and reset the guessesLeft data property to its original value.
10. Add CSS styles to customize the look and feel of the game, including different colors and fonts for success and failure messages, and animations for when the game is reset or disabled.

That's it! With this simple scenario, you can develop a fun and interactive game using Vue JS.

A2. making guide for integrating third-party services and APIs with a game deployed on a static hosting platform:

1.Identify the third-party services and APIs that you want to integrate with your game, such as social media sharing, analytics tracking, and payment processing. Evaluate the benefits and drawbacks of each service, and ensure that it meets the technical and business requirements of your game.

2.Determine the integration points for each service, such as the API endpoints, data formats, and authentication mechanisms. Ensure that the integration points are well-documented, stable, and reliable, and that they comply with the security and privacy standards of your game.

3.Implement the integration code using a robust and scalable technology stack, such as Vue.js, Node.js, and Express.js. Use best practices for code organization, modularization, and testing, and ensure that the code is optimized for performance and maintainability.

4.Test and validate the integration code using a variety of test cases and scenarios, such as different user inputs, network conditions, and error handling. Use tools and frameworks like Mocha, Chai, and Sinon to automate the testing process and ensure that the integration code meets the functional and non-functional requirements of your game.

5.Monitor and optimize the integration code using real-time analytics and monitoring tools, such as Google Analytics and New Relic. Use the data to identify performance bottlenecks, security risks, and user behavior patterns, and optimize the integration code accordingly.

6.Deploy and maintain the integration code using a reliable and scalable hosting platform, such as AWS or Heroku. Use best practices for deployment automation, version control, and disaster recovery, and ensure that the integration code remains scalable and maintainable as the game grows in popularity and complexity.

By following these best practices, you can integrate third-party services and APIs with your game deployed on a static hosting platform in a robust, scalable, and maintainable way.

A3.Developing a game interface in Vue that supports different gameplay modes and settings involves several steps, including design, implementation, and testing. Here are the steps to follow:

1.Design the interface: Start by defining the user interface (UI) and user experience (UX) requirements for your game based on the different gameplay modes and settings you want to support. Use wireframes, mockups, and prototypes to visualize the interface design and iterate on it based on user feedback and usability testing.

2.Implement the interface: Use Vue.js and other front-end technologies to implement the interface design and integrate it with the game logic and mechanics. Use components, props, and events to create reusable and modular UI elements, and ensure that the interface is responsive, accessible, and optimized for different screen sizes and resolutions.

3..Test and validate the interface: Use automated and manual testing techniques to validate the interface design and ensure that it meets the needs and expectations of your target audience. Some testing techniques to consider include:

4..Unit testing: Test each component and functionality in isolation to ensure that it works as intended and does not introduce errors or bugs.

5.Integration testing: Test the interface design and functionality in the context of the game mechanics and logic to ensure that they are consistent and compatible.

6.User acceptance testing: Test the interface design and functionality with a representative sample of your target audience to ensure that they are intuitive, usable, and engaging.

7.Performance testing: Test the interface design and functionality under different network conditions, device types, and usage scenarios to ensure that they are fast, responsive, and efficient.

By following these steps, you can develop a game interface in Vue that supports different gameplay modes and settings, and ensure that it meets the needs and expectations of your target audience through rigorous testing and validation.

A4. To leverage the power of Vue to create a dynamic and responsive game that can adapt to different user inputs and scenarios, follow these steps:

Design the game architecture: Start by defining the game mechanics and logic, and breaking them down into modular and reusable components using Vue.js. Use props, events, and slots to create a flexible and composable architecture that can be easily extended and modified as the game evolves.

Implement the game mechanics: Use Vue.js and other front-end technologies to implement the game mechanics, including input handling, game state management, animation, and rendering. Use Vuex or other state management libraries to manage the game state and ensure that it is consistent and scalable.

Optimize game performance and memory usage: Use performance profiling and debugging tools to identify bottlenecks and optimize the game performance and memory usage. Some techniques to consider include:

Code splitting: Split the game code into smaller and more manageable chunks to reduce the initial load time and improve the game performance.

Lazy loading: Load the game assets and components on-demand to reduce the memory usage and improve the game performance.

Caching: Cache the frequently used data and assets to reduce the network requests and improve the game performance.

Debouncing and throttling: Use debounce and throttle functions to limit the number of user inputs and network requests and ensure that the game remains responsive and smooth.

1. Test and validate the game: Use automated and manual testing techniques to validate the game mechanics and performance and ensure that it meets the needs and expectations of your target audience. Some testing techniques to consider include:

Unit testing: Test each game component and functionality in isolation to ensure that it works as intended and does not introduce errors or bugs.

Integration testing: Test the game mechanics and functionality in the context of the game architecture to ensure that they are consistent and compatible.

User acceptance testing: Test the game with a representative sample of your target audience to ensure that it is engaging, fun, and challenging.

END OF ASSESSMENT!