**20L0943**

**Saadan Khokhar**

**SQE Assignment 3**

**Architectural and Component level modularity requirements**

**Micro Services**

The micro service architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.

Micro services architecture is a concept of building an application based on breaking it into multiple modules. Each module has its own specific responsibilities but communicates with others to form a unified system. This approach provides freedom of development and choice of tools (technology language, framework) within each application.

The right way to deploy applications is in [containers](https://opensource.com/resources/what-are-linux-containers). Briefly, a container is a miniature virtual server packed with the software required to run an application. The container pack is small, smart, and easy to deploy and maintain. And deploying your application in a container is clever. You can deploy as many instances as you need and scale up or down as needed to meet the current load.

https://i.imgur.com/ELcmMjy.png)

**Micro Services Example in Vue StoreFront Code**

<https://i.imgur.com/gEN73AD.png>

Vue StoreFront has micro services requirements as they divided the services in many micro ones. To manage the increase and decrease of components and distribute the requests effectively

### Modular Folder Structure

This structure is interesting for projects with a larger scale. The main idea is that each module we define has all the code related to it and only code from the module itself is imported. When we have several modules that need the same piece of code, we can write it into a shared folder and import it into the different modules. The fundamental rule to be followed is not to import code between modules. However, the great drawback of this structure is that it can be complex to define what a module is and this can have a great impact on the success of our organization**.**

Every project has its own requirements and each team has its own way of working, so this is not THE guide on how to structure your Vue project; it’s more of a source of ideas and good practices that you can follow to improve your code maintainability. Model based is not a file based system

When starting a new Vue project, the easiest way to go is vuecli project structure, which somehow forces you to group your files by type: components, services, helpers with a few odd exceptions, such as the router and the store. This structure is not wrong — for small applications, it’s very easy to maintain, but for midsize/large applications there are a few downsides:

The source code navigation is not simple: imagine the amount of files inside the components folder, and knowing the purpose of each one — same for any other file type

Debugging can be very time-consuming: you’d need to jump filetofile and draw a mental map of how a simple process works to get to the root cause of a bug.

A better alternative is to structure your project using old-fashioned modules and to think of each module as a small VUECLI project, decreasing the downsides explained above, so you get something like this:

<https://i.imgur.com/tln2Yys.png>)

As described on the [Introduction to Vue Storefront](https://docs.vuestorefront.io/v2/getting-started/introduction.html) page, Vue Storefront is just a Nuxt.js project under the hood. For this reason, our project structure inherits from Nuxt.js but has some additional files.

* [**.nuxt**](https://nuxtjs.org/docs/2.x/directory-structure/nuxt)is a dynamically generated build directory. You should **not** manually modify it, nor synchronize it using version control like GIT.
* **components** contains Vue.js components used on different pages or parts of your application. You can import these components from pages, layouts, and other components.
* [**lang**](https://docs.vuestorefront.io/v2/getting-started/internationalization.html)contains translations for your application. Available locales are configured in the nuxt.config.js file.

**Class Components**

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML via a render () function. Components come in two types, Class components and Function components. Class component is instantiated and a different life cycle method is kept alive and being run and invoked depending on the phase of the class component. Moreover, React lifecycle methods can be used inside class components (for example, componentDidMount). It requires different syntax inside a class component to implement hooks. Constructors are used as it needs to store state.

Example: constructor(props) { super(props); this.state = {name: ‘ ‘}}

Vue StoreFront is doing each functionality in a separate module and whenever they need that module to be there they just simply "import" that module.

Example:

<https://i.imgur.com/HnZcfKJ.png>)

**Algorithmic Optimization**

To achieve better time and space complexities we should choose the best possible algorithms. Some optimal algorithms for Sorting are Bubble Sort, Insertion Sort, Merge Sort and Quick Sort. For Searching some optimal algorithms are Linear Search,  Binary Search and Hash table Lookup. These algorithms may not be the best in all scenarios but when combined with certain data structures these can produce exceptional results.

### Reusability

Components are usually designed to be reused in different situations in different applications. However, some components may be designed for a specifi–c task. Reusability is the use of existing assets in some form within the software product development process; these assets are products and byproducts of the software development life cycle and include code, software components, test suites, designs and documentation.

**Extensibility**

Most projects need to extend the base of the framework to meet their needs, be it with a ready to use or a custom plugin. Well-thought-out extensions or plugins system enables flexibility to meet the demands of the most, even very diverse projects. A component can be extended from existing components to provide new behavior. Extensibility is a measure of the ability to extend a system and the level of effort required to implement the extension. Extensions can be through the addition of new functionality or through modification of existing functionality. First, you should consider which part of the application you need to extend frontend, backend. Depending on your needs, you can need to extend:

* Vue,js
* Nuxt,js
* Middleware

To extend it we use plugins. Plugins allow adding global level functionalities to Vue.js, such as components, methods, helpers, or directives. These mainly extend the frontend portion of the application and can be divided into two categories:

* UI plugins
* NonUI plugins

**Vue.js UI plugins**

UI plugins extend how the application looks or behaves on user interactions. They include plugins that add support for:

* Event handling.
* Responsive design, resizing, scrolling, and animations.
* Handling forms and validation.
* Routing, lazy loading, lazy hydration, meta tags.

**Vue.js nonUI plugins**

NonUI plugins extend how the application works under the hood or handles state and storage. Non UI plugins are all the extensions that are not for the user interface or the interaction with the User such as :-

* Making HTTP requests.
* Internationalization (i18n).
* Custom events.
* Storage
* State management.

### Linters for better code quality

A code linter is a tool that automatically checks your code to see if it meets your project’s or company’s coding convention. If it doesn’t match the convention, linters will show these errors as warnings. During software development, these warnings tend to get overlooked when developers prioritize speed over quality. But these small errors can accumulate fast in a short amount of time, leading to a significant workload for your team. To avoid these hefty workloads, linters identify these errors quickly so that your team can resolve them quickly, too.

## **Performance Optimization**

Performance optimization, also known as “performance tuning”, is usually an iterative approach to making and then monitoring modifications to an application and its database. It could involve adjusting the configuration of the database and server, or making changes to the applications and the [SQL](https://www.sciencedirect.com/topics/computer-science/structured-query-language) that maintain and query the database. Number of users

* Complexity of the application and the SQL
* Volatility of the data
* DBMS and [server platform](https://www.sciencedirect.com/topics/computer-science/server-platform).

Declare preload links in your HTML to instruct the browser to download key resources as soon as possible. This is achieved by

* Reducing third party usage
* Eliminate Render blocking resources
* Minimize the JS and CSS so that load time is less
* // nuxt.config.js
* export default {
* buildModules: [
* ['@vue-storefront/nuxt', {
* performance: {
* purgeCSS: {
* enabled: true,
* paths: [
* '\*\*/\*.vue'
* ]
* }
* }
* }]
* ]
* };
* Image handling
* <img src="cat-large.jpg" srcset="cat-small.jpg 480w, cat-large.jpg 1080w" sizes="50vw">
* Reduce Exec time

### Usage of Cache

Cache is memory placed in between the processor and main memory. Cache is responsible for holding copies of main memory data for faster retrieval by the processor. Cache memory is important because it improves the efficiency of data retrieval.

The most frequently visited pages are embedded with Cache so that they don’t have to reload every time someone visits it, The pages that are cached are

* Home page with vhome tag
* Category page with Vcategory tag
* Tags for products P$(prod.id) and C$(cat.id)
* Product page with the Vproduct-${route.value.params.id} tag and tags for the main
* P${prod.\_id} as well as related products P${prod.\_id} and categories C${cat.id}
* Tags are invalidated also using the invalidation endpoints .inv
* key — string matching the VSF\_REDIS\_CACHE\_INVALIDATE\_KEY key in the .env file.
* tags — a comma-separated list of tags for invalidation.

### Usage of Content Delivery Network

To minimize the distance between the visitors and your website's server, a CDN stores a cached version of its content in multiple geographical locations (a.k.a., points of presence, or PoPs).

### 

### Usage of Lazy Loading in web and asset workflows in games

Lazy loading is a technique for waiting to load certain parts of a webpage — especially images — until they are needed. Instead of loading everything all at once, known as "eager" loading, the browser does not request certain resources until the user interacts in such a way that the resources are needed. The advantages of lazy loading are as follows:

Lazy loading is a technique used to prevent or delay the loading of non-critical resources until they are needed. You can use this mechanism for different types of resources, but in the case of images, our goal is to lazily load everything that is not visible to the user within the initial viewport. All other images can be loaded when the user scrolls down the page.

Use the loading="lazy" attribute to load an image lazily. It also works for the <nuxt-img> component.

Using the [<nuxt-img> (opens new window)](https://image.nuxtjs.org/components/nuxt-img)component from the [@nuxt/image (opens new window)](https://image.nuxtjs.org/)package is likely the single best thing you can do to stop worrying about images. It offers features for most of the things mentioned in the following sections. Using most of them only requires you to pass a single attribute.

It offers image resizing, converting formats, preloading, and integrations with the most popular image transformation services.

<nuxt-img

src="/hero-image.png"

format="webp"

quality="80"

width="200"

height="100"

loading="lazy"

/>

If you have just a few static images on your website, you can manually compress them using a website like [Squoosh.app (opens new window)](https://squoosh.app/" \t "_blank).

**Faster page load:** All else being equal, webpages with smaller file sizes load faster. With lazy loading, a web page starts off smaller than its full size and thus loads faster. [Speedy web performance](https://www.cloudflare.com/learning/performance/why-site-speed-matters/) has numerous benefits, including [better SEO](https://www.cloudflare.com/learning/performance/how-website-speed-boosts-seo/), [higher conversion rates](https://www.cloudflare.com/learning/performance/more/website-performance-conversion-rates/), and an improved user experience.

**No unnecessary content:** Suppose a page loads multiple below-the-fold images but the user exits the page before scrolling down. In such a case, the bandwidth used to deliver the images and the browser's time spent requesting and rendering the images were essentially wasted. In contrast, lazy loading ensures that these images only load when necessary. This saves time and processing power, and it may save money for the website owner because less [bandwidth](https://www.cloudflare.com/learning/cdn/how-cdns-reduce-bandwidth-cost/) is used.

### Usage of PWA

Progressive Web Application (PWA) is a web standard that uses design patterns and advanced technologies to deliver a fast and reliable experience to users on both mobile and desktop devices. It can work on any platform using standard-compliant browsers with the aims to: Increase customer engagement.

With PWAs you can build one website that functions as an iOS and Android application. This saves the cost and complexity of building a separate experience of iOS and Android, while giving users the best functionality regardless of how they are accessing the content. Because a progressive web app is lightweight and doesn't put a lot of demand on a device's resources, it loads pages fast. Therefore, people will spend more time on your webpage.

PWA is

* Reliable
* Fast Loading
* Engaging
* Cost effective
* Better SEO results
* Lower Bounce Rates
* PWA leverages the power of headless commerce
* Boost engagement

### Service workers and local storage

## Service worker may refer to: Social service worker, a person engaged in social work. Pink-collar worker, a person in the service industry whose labor is related to customer interaction, entertainment, sales or other service-oriented work.

**import** { ActionTree } **from** 'vuex'

**import** \* **as** types **from** './mutation-types'

**import** RootState **from** '@vue-storefront/core/types/RootState'

**import** WishlistState **from** '../types/WishlistState'

**import** { StorageManager } **from** '@vue-storefront/core/lib/storage-manager'

**const** actions: ActionTree<WishlistState, RootState> = {

clear (context) {

context.commit(types.WISH\_DEL\_ALL\_ITEMS, [])

},

**async** load ({ commit, getters, dispatch }, force: boolean = false) {

**if** (!force && getters.isWishlistLoaded) **return**

commit(types.SET\_WISHLIST\_LOADED)

**const** storedItems = **await** dispatch('loadFromCache')

commit(types.WISH\_LOAD\_WISH, storedItems)

},

loadFromCache () {

**const** wishlistStorage = StorageManager.get('wishlist')

**return** wishlistStorage.getItem('current-wishlist')

},

Create your own wishlist or storage. Create your own module src/module

LocalStorage or IndexedDB (depending on browser support, abstracted with LocalForage library) are mainly used for storing user-based data like cart, login token, made orders, personal data, other choices (like favorite products).

The Service Workers cache works like we're consistently executing the backend requests to invalidate the local cache. It's the NetworkFirst mode. That means there is virtually no risk of stale / non-invalidated data served from the local cache

Whenever it's possible, the service worker cache is in use. For easier service worker cache management, Google provided Workbox is in use.  
Service worker cache is used mainly for catalog (category, product, and attribute list and detail) and CMS (pages, blocks, banners).

**Security Requirements**:

**Built In security:**

We guarantee 99,8% SLA for the infrastructure and Vue Storefront-based app and provide you with complete geofencing. DDoS protection by giving you full control over which country your store is available in.

**Secrets Managements**

Secrets management refers to the tools and methods for managing digital authentication credentials (secrets), including passwords, keys, APIs, and tokens for use in applications, services, privileged accounts and other sensitive parts of the IT ecosystem.

To improve the security Vue uses Helmet using the nuxt js and the server middlware

Helmet is disabled by default. You can enable it using the helmet property in the @vue-storefront/middleware/nuxt module configuration.

// nuxt.config.js

export default {

modules: [

['@vue-storefront/middleware/nuxt', {

helmet: true

// or

helmet: {

// ...configuration

}

}]

]

}

If you don’t want to hardcode the URL you can use the environment variables and the middleware. Then add an entry in the .env file or use any other method for passing environmental variables that suits your needs

**Auto Scaling**

AWS Auto Scaling lets you build scaling plans that automate how groups of different resources respond to changes in demand

* the benefits include
* high performance
* faster changes
* smart scaling decisions
* In the context of Vue applications, a "composable" is a function that leverages Vue's Composition API to encapsulate and reuse **stateful logic**.
* When building frontend applications, we often need to reuse logic for common tasks. For example, we may need to format dates in many places, so we extract a reusable function for that. This formatter function encapsulates **stateless logic**: it takes some input and immediately returns expected output.

**Load Balancing**

It is used to navigate and distribute high traffic to multiple targets. They use Load Balancing to avoid the System to crash. Multiple paths are provided to ensure that the system doesnot reach an endpoint and alternate routes are there to ensure reliability

**Cloud Based**

Vue Storefront uses the Cloud Services to ensure faster deployment and cost efficient services. Daily backups are guaranteed and the data is extracted from the cloud. The data is stored and retrieved from the cloud

**Server less Deployment**

Since Microservices are used in Vue Storefront it allows the serverless deployment . Serverless is a cloud-native development model that allows developers to build and run applications without having to manage servers. Using the AWS and VUE it has a serverless deployment and the cloud based is also used so you don’t have to manage the servers

Serverless is the fastest growing cloud method. It allows the reusability of independent components and build scalable apps

**Versioning**

Multiple Versions of Vue Storefront are relased. The latest being the **V**[**1.12.3**](https://github.com/vuestorefront/vue-storefront/releases/tag/v1.12.3)

Vue does not have a fixed release cycle.

* Patch releases are released as needed.
* Minor releases always contain new features, with a typical time frame of 3~6 months in between. Minor releases always go through a beta pre-release phase.
* Major releases will be announced ahead of time, and will go through an early discussion phase and alpha / beta pre-release phases.
* The following changes are made and bugs have been fixed in the most recent release
* Remove redundant user token invalidation code - TerrorSquad ([#4533](https://github.com/vuestorefront/vue-storefront/pull/4533))
* configurableChildrenImages helper function incorrect loading images - @RakowskiPrzem ([#4575](https://github.com/vuestorefront/vue-storefront/issues/4575))
* Fix user/register call by adding storeId - [@haelbichalex](https://github.com/haelbichalex) ([#4697](https://github.com/vuestorefront/vue-storefront/pull/4697))
* refresh categoryMappedFilters after loading attributes in PLP - [@gibkigonzo](https://github.com/gibkigonzo) ([#4695](https://github.com/vuestorefront/vue-storefront/issues/4695))
* Check module registration for wishlist and compare list - gibkigonzo ([#4764](https://github.com/vuestorefront/vue-storefront/issues/4764))
* Fix gallery image generation by checking if image exists - simonmaass (#4908)
* Fix getSelectedOption based on attribute\_code check - simonmaass ([#4851](https://github.com/vuestorefront/vue-storefront/issues/4851))
* add new resetUserInvalidation action to clear invalidation state before login,  
  clear order history and refresh token after logout,  
  add redirection when user is logged out and sees myaccount page - gibkigonzo ([#4882](https://github.com/vuestorefront/vue-storefront/pull/4882))
* Fix config.products.filterUnavailableVariants feature - Ogek ([#4923](https://github.com/vuestorefront/vue-storefront/pull/4923))
* added 'afterUserProfileUpdated' hook which allows us to take control over update address flow (send notification if something failed)

The details of the previous releases and modifications can be read on the following link

https://github.com/vuestorefront/vue-storefront/releases

**Lifecycle management**

**Contribution**