CoreUI Pro Vue

CoreUI Pro Vue + NetBeans IDE

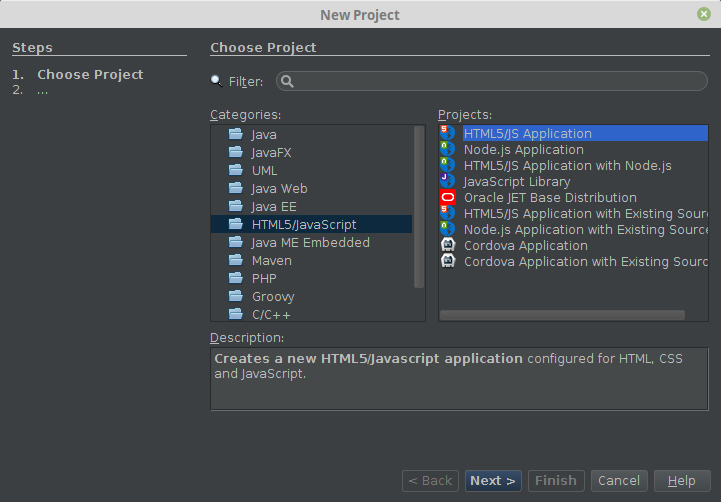
Let’s start by creating a new project in NetBeans IDE in order to edit CoreUI Pro Vue source code. It is not necessary to use NetBeans IDE as this is only used to make editing files easier. You can use any other IDE (Eclipse, IntelliJ, etc.) or whatever is best for you.

# System Settings

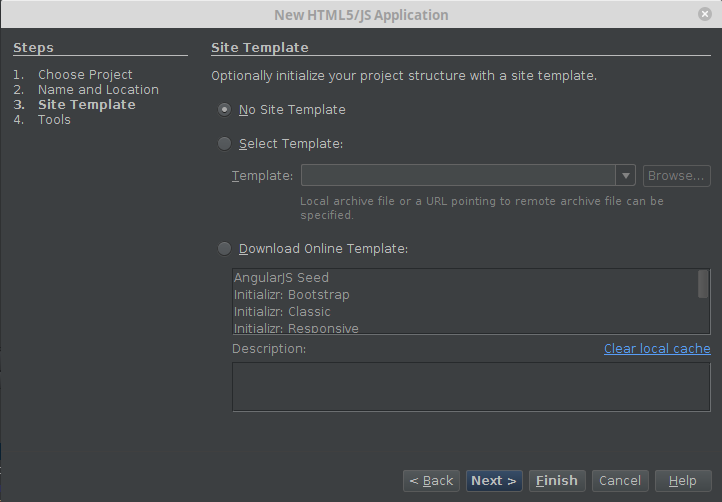
1. CoreUI Pro Vue template 2.0.3
2. NetBeans IDE 8.2
3. NPM 3.5.2

# Integration Guide

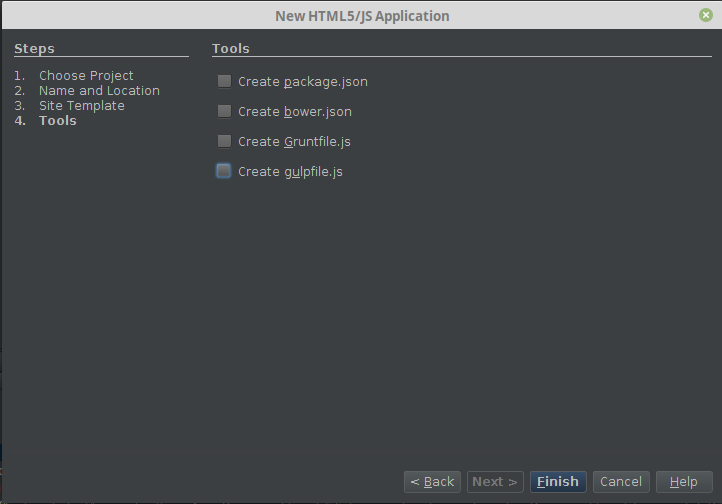
1. Create a new project *HTML5/JavaScript* > *HTML5/JS Application* using your NetBeans IDE.



1. Select *No Site Template* in Site Template step.



1. Tools > Deselect options *Create package.json*, *Create bower.json*, *Create Gruntfile.js* and *Create gulpfile.js* in Tools step.



1. Delete the default *index.html* file under *Site Root* folder.
2. Copy all CoreUI Pro Vue files inside the *Site Root* folder.
3. In the command line, change your working directory to your project’s public\_html folder:

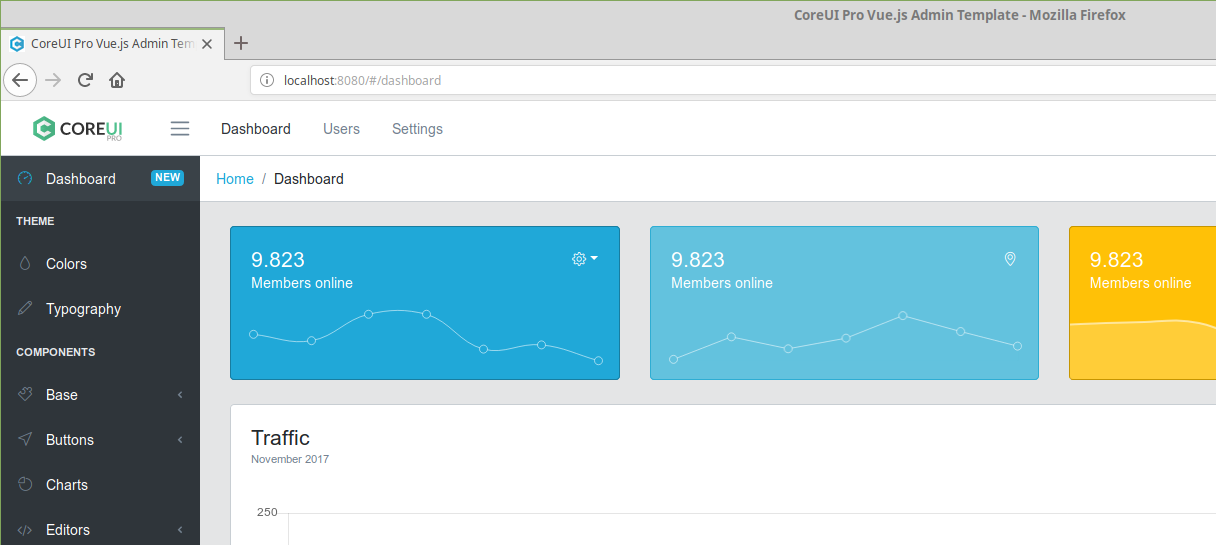
> cd /*<project\_folder>*/public\_html

1. Install all required dependencies running the following command:

> npm install

1. Run your template running the following command. You should be able to see your application running in [*http://localhost:8080*](http://localhost:8080/)

> npm run serve



NUXT

Now that you have your CoreUI Pro Vue template working and opened in NetBeans IDE, you can start Nuxt integration. First you still need a new Nuxt application to get the necessary resources. Let’s create a Nuxt application by using the *create-nuxt-app* basic template provided by Nuxt.

# System Settings

1. NPM 3.5.2
2. NPX 10.2.0

# External References

1. <https://nuxtjs.org/guide/installation/>

# Nuxt Template

1. Create a new Nuxt application using the following command:

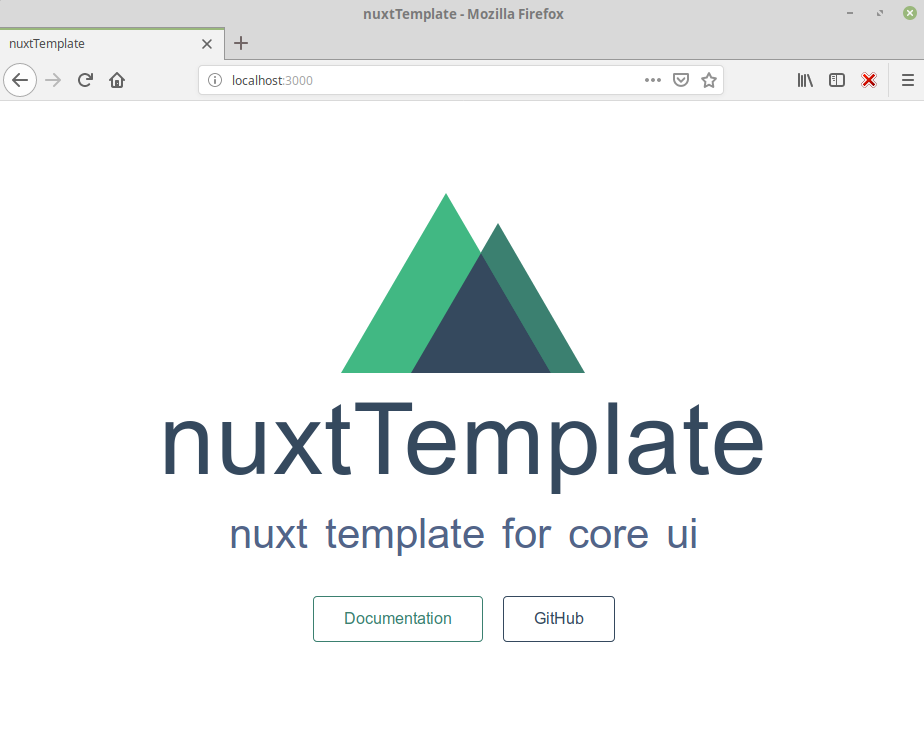
> npx create-nuxt-app *nuxtTemplate*

1. Follow the steps in the installation wizard and configure as follows:
   * Project name: *nuxtTemplate*
   * Project description: *nuxt template for coreui*
   * Use a custom server framework: *none*
   * Use a custom UI framework: *bootstrap*
   * Choose rendering mode: *Single Page App*
   * Use axios module: *yes*
   * Use eslint: *no*
   * Use prettier: *no*
   * Author name: *nuxt template author*
   * Choose a package manager: *npm*
2. The installation wizard has just created a new folder called */nuxtTemplate*. Change your working directory to */nuxtTemplate* folder:

> cd nuxtTemplate

1. Test your application using the following command. You should be able to see your application running in [*http://localhost:3000*](http://localhost:3000/)

> npm run dev



CoreUI Pro Vue + Nuxt

# System Settings

1. CoreUI Pro Vue template 2.0.3
2. NetBeans IDE 8.2
3. NPM 3.5.2

# Integration Guide

1. Copy the following folders from your Nuxt application into the */public\_html* folder of your CoreUI Pro Vue project:
   * */assets*
   * */components*
   * */layouts*
   * */middleware*
   * */pages*
   * */plugins*
   * */static*
   * */store*
2. Copy the Nuxt configuration file from your Nuxt application into the */public\_html*:
   * *nuxt.config.js*
3. Add the following scripts to your *package.json* file. Remove or rename the existing CoreUI Pro Vue scripts.
   * *"dev": "nuxt"*
   * *"build": "nuxt build"*
   * *"start": "nuxt start"*
4. Add the following dependencies to your *package.json* file:
   * *"cross-env": "^5.2.0"*
   * *"nuxt": "^2.0.0"*
   * *"bootstrap-vue": "^2.0.0-beta"*
   * *"bootstrap": "^4.1.3"*
   * *"@nuxtjs/axios": "^5.0.0"*
5. Copy the */public\_html/src/assets/scss* folder to the */public\_html/assets* folder.
6. Create a new folder called */layouts* inside */components* folder.
7. Copy the following files from the */public\_html/src/containers* folder to the */public\_html/components/layouts* folder:
   * *DefaultAside.vue*
   * *DefaultHeaderDropdown.vue*
   * *DefaultHeaderDropdownAccnt.vue*
   * *DefaultHeaderDropdownMssgs.vue*
   * *DefaultHeaderDropdownNotif.vue*
   * *DefaultHeaderDropdownTasks.vue*
8. Create a new folder called */sidebar* inside */public\_html/components* folder.
9. Copy the following files from the */public\_html/node\_modules/@coreui/vue/src/components/Sidebar* folder to the */public\_html/components/sidebar* folder:
   * *SidebarNav.vue*
   * *SidebarNavDivider.vue*
   * *SidebarNavDropdown.vue*
   * *SidebarNavItem.vue*
   * *SidebarNavLabel.vue*
   * *SidebarNavLink.vue*
   * *SidebarNavTitle.vue*
10. Copy the following files from the */public\_html/node\_modules/@coreui/vue/src/mixins* folder to the */public\_html/components/sidebar* folder:
    * *hideMobile.js*
11. Remove the existing *default.vue* file from within */public\_html/layouts* folder.
12. Copy the nav.js file from */public\_html/src* folder to the */public\_html/layouts* folder.
13. Copy the DefaultContainer.vue file from within the /public\_html/src/containers folder to the */public\_html/layouts* folder and rename to *default.vue*
14. Make the following changes to the /public\_html/layouts/default.vue file:
    * Replace the <router-view></router-view> block with <nuxt/>

|  |
| --- |
| ...  <main class="main">  <Breadcrumb :list="list"/>  <div class="container-fluid">  <!--<router-view></router-view>-->  **<nuxt/>**  </div>  </main>  ... |

* + Update the path references in the import section of the script:

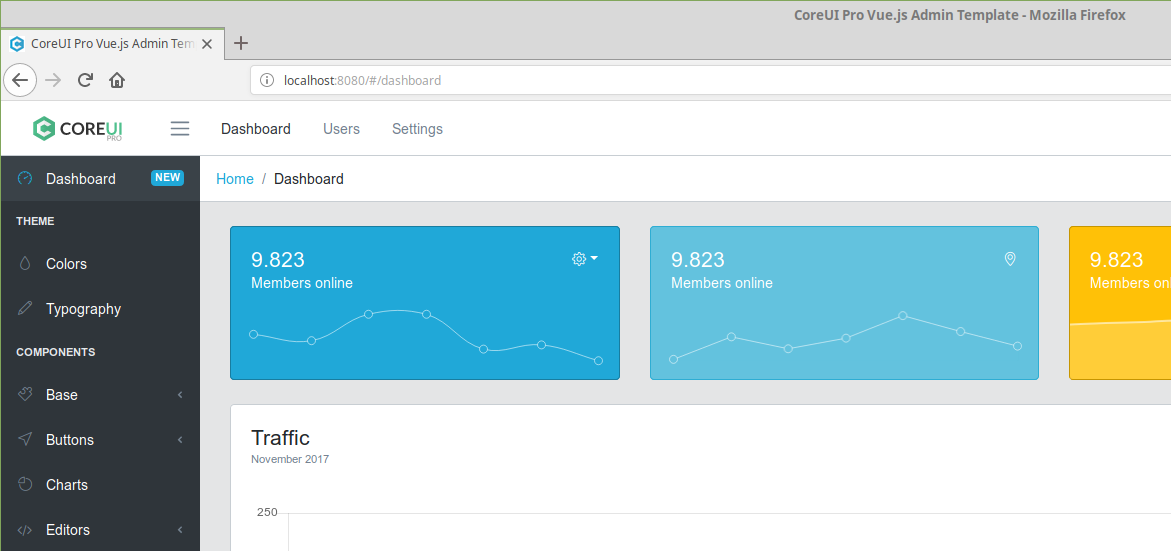
|  |
| --- |
| …  import **Vue** from 'vue';  import **nav** from './\_nav';  import { **Header** as AppHeader, **SidebarToggler**, **Sidebar** as AppSidebar, **SidebarFooter**, **SidebarForm**, **SidebarHeader**, **SidebarMinimizer**, **Aside** as AppAside, **AsideToggler**, **Footer** as TheFooter, **Breadcrumb** } from '@coreui/vue';  import **SidebarNav** from '@/components/sidebar/SidebarNav';  import **DefaultAside** from '@/components/layouts/DefaultAside';  import **DefaultHeaderDropdown** from '@/components/layouts/DefaultHeaderDropdown';  import **DefaultHeaderDropdownNotif** from '@/components/layouts/DefaultHeaderDropdownNotif';  import **DefaultHeaderDropdownAccnt** from '@/components/layouts/DefaultHeaderDropdownAccnt';  import **DefaultHeaderDropdownMssgs** from '@/components/layouts/DefaultHeaderDropdownMssgs';  import **DefaultHeaderDropdownTasks** from '@/components/layouts/DefaultHeaderDropdownTasks';  import { **capitalize** } from '@/middleware/utils';  … |

* + Update the path references in the style section:

|  |
| --- |
| …  // CoreUI Icons Set  @import '@coreui/icons/css/coreui-icons.min.css';  /\* Import Font Awesome Icons Set \*/  $fa-font-path: '~font-awesome/fonts/';  @import '~font-awesome/scss/font-awesome.scss';  /\* Import Simple Line Icons Set \*/  $simple-line-font-path: '~simple-line-icons/fonts/';  @import '~simple-line-icons/scss/simple-line-icons.scss';  /\* Import Flag Icons Set \*/  @import 'flag-icon-css/css/flag-icon.min.css';  /\* Import Bootstrap Vue Styles \*/  @import 'bootstrap-vue/dist/bootstrap-vue.css';  // Import Main styles for this application  @import 'assets/scss/style';  … |

1. Copy the */img* folder and *favicon.ico* file from the */public\_html/public* folder to */public\_html/static* folder.
2. Test your application using the following command. You should be able to see your application running in [*http://localhost:3000*](http://localhost:3000/)

> npm run dev



Security

# Backend Services

## System Settings

1. Java SE Runtime Environment 1.8
2. Maven 3.5.2
3. NetBeans IDE 8.2

## External References

1. <https://medium.com/omarelgabrys-blog/microservices-with-spring-boot-creating-our-microserivces-gateway-part-2-31f8aa6b215b>
2. <https://medium.com/omarelgabrys-blog/microservices-with-spring-boot-authentication-with-jwt-part-3-fafc9d7187e8>

## Eureka Server

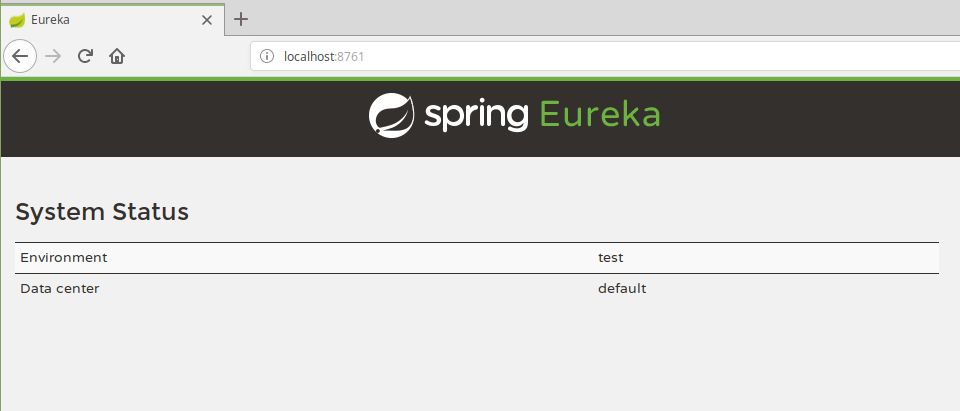
1. Create a new Maven project using Spring Initializr (<https://start.spring.io/>)
   * Spring Boot version: 2.1.2
   * Group: com.waumovil
   * Artifact: serviceregistry
   * Dependencies:
     + Web
     + Eureka Server
     + DevTools
2. Your generated *pom.xml* file should look as follows:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.1.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>com.waumovil</groupId>  <artifactId>serviceregistry</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>serviceregistry</name>  <description>Demo project for Spring Boot</description>  <properties>  <java.version>1.8</java.version>  <spring-cloud.version>**Greenwich.RC2**</spring-cloud.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>**spring-boot-starter-web**</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>**spring-cloud-starter-netflix-eureka-server**</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>**spring-boot-devtools**</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>**spring-boot-starter-test**</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>${spring-cloud.version}</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  <repositories>  <repository>  <id>spring-milestones</id>  <name>Spring Milestones</name>  <url>https://repo.spring.io/milestone</url>  </repository>  </repositories>  </project> |

1. Rename your *application.properties* to *application.yml* and include the following contents:

|  |
| --- |
| # Give a name to the eureka server  spring:  application:  name: eureka-server    # default port for eureka server  server:  port: 8761    # eureka by default will register itself as a client. So, we need to set it to false.  eureka:  client:  register-with-eureka: false  fetch-registry: false |

1. Test your application using the following command. You should be able to see your application running in [*http://localhost:8761*](http://localhost:8761/)



## Authorization Service

1. Create a new Maven project using Spring Initializr (<https://start.spring.io/>)
   * Spring Boot version: 2.1.2
   * Group: com.waumovil
   * Artifact: authservice
   * Dependencies:
     + Web
     + Eureka Client
     + Security
     + DevTools
     + gson (you will add this later)
     + jjwt (you will add this later)
     + Persistence API (you will add this later)
2. Your generated *pom.xml* file should look as follows:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.1.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>com.example</groupId>  <artifactId>jjwt-auth-1</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>jjwt-auth-1</name>  <description>Demo project for Spring Boot</description>  <properties>  <java.version>1.8</java.version>  <spring-cloud.version>**Greenwich.RC2**</spring-cloud.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>**spring-boot-starter-security**</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>**spring-boot-starter-web**</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>**spring-boot-devtools**</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>**spring-boot-starter-test**</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>**spring-security-test**</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>**spring-cloud-starter-netflix-eureka-client**</artifactId>  </dependency>  <dependency>  <groupId>io.jsonwebtoken</groupId>  <artifactId>**jjwt**</artifactId>  <version>0.9.1</version>  <type>jar</type>  </dependency>  <dependency>  <groupId>javax.persistence</groupId>  <artifactId>**javax.persistence-api**</artifactId>  <version>2.2</version>  <type>jar</type>  </dependency>  <dependency>  <groupId>com.google.code.gson</groupId>  <artifactId>**gson**</artifactId>  <version>2.8.5</version>  <type>jar</type>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>${spring-cloud.version}</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>    <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

1. Rename your *application.properties* to *application.yml* and include the following contents:

|  |
| --- |
| # Give a name to the authorization service  spring:  application:  name: auth-service    # default port for authorization service  server:  port: 9100    # authorization service by default will register itself as a client  eureka:  client:  service-url:  default-zone: <http://localhost:8761/eureka/> |

1. Basic authorization service must have at least the following components:
   * com.waumovil.authservice
     + **AuthService**.java

your main class

* + com.waumovil.authservice.controller
    - **DefaultController**.java

the controller that contains user services

* + com.waumovil.authservice.model
    - **ApplicationUser**.java

entity model for user

* + - **AuthenticationData**.java

custom data structure of Json Web Token

* + - **ResponseBody**.java

template class for response body

* + - **UnsuccessfulAuthData**.java

custom data structure for your error messages

* + - **UserCredentials**.java

entity model for basic user credentials

* + - **UserData**.java

custom data structure for your user data

* + com.waumovil.authservice.security
    - **CustomOncePerRequestFilter**.java

filter class to decode and parse the incoming Json Web Token on each request

* + - **CustomUsernamePasswordAuthenticationFilter**.java

filter class to handle events like authentication attempt, successful and unsuccessful authentication

* + - **CustomWebSecurityConfigurerAdapter**.java

filter class to configure spring security

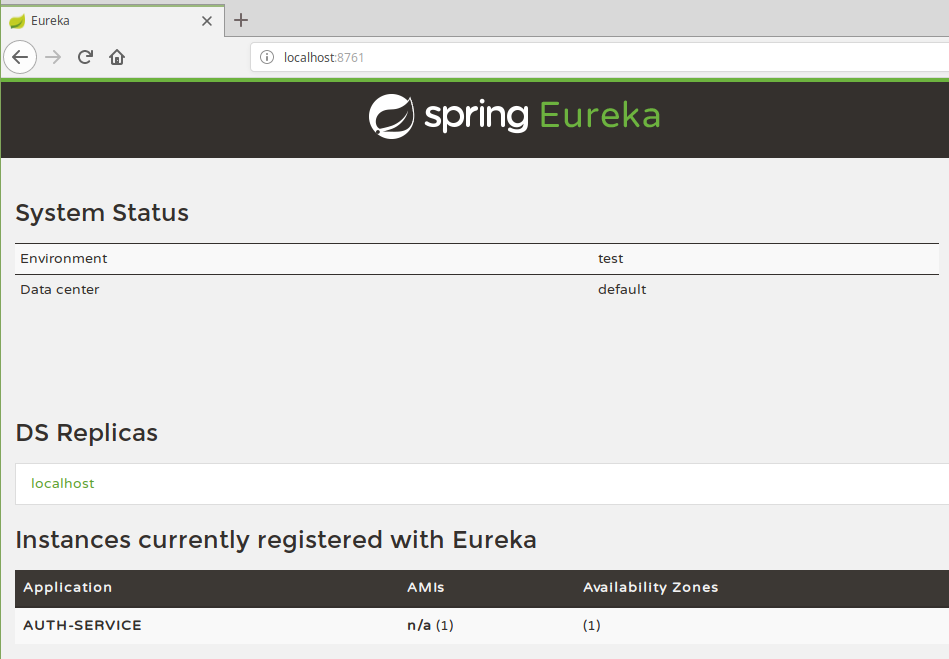
* + - **JwtConfig**.java

security properties

* + - **UserDetailsServiceImpl**.java

service implementation for user

1. You should be able to see your service registered at Eureka server in [*http://localhost:*](http://localhost:8761/)*9100*



1. Test your authorization service using a tool like Postman. You should be able to get an object containing a Json Web Token -JWT- by making a POST request to <http://localhost:9100/auth>.
   * **Request body** as Json string:

|  |
| --- |
| {  "**username**": "admin",  "**password**": "123456"  } |

* + **Response body** as Json string:

|  |
| --- |
| {  "**data**": {  "**type**": "bearer",  "**token**": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImF1dGhvcml0aWVzIjpbIlJPTEVfQURNSU4iXSwiaWF0IjoxNTQ3NzQwODM5LCJleHAiOjE1NDc4MjcyMzl9.w8EEbVNpqdzQkdXvltry7\_E7PuGS\_hy0zdhz00cfcy7lKwHmlgEafFp39PblC-Q8xnz\_ggvw4jIJf5k3EDh3Bg"  }  } |

1. Test your user service using a tool like Postman and the Json Web Token -JWT- generated in the previous step. You should be able to get an object containing user information by making a GET request to <http://localhost:9100/me>.
   * **Request headers**:
     + **Key**: Authorization
     + **Value**: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImF1dGhvcml0aWVzIjpbIlJPTEVfQURNSU4iXSwiaWF0IjoxNTQ3NzQwODM5LCJleHAiOjE1NDc4MjcyMzl9.w8EEbVNpqdzQkdXvltry7\_E7PuGS\_hy0zdhz00cfcy7lKwHmlgEafFp39PblC-Q8xnz\_ggvw4jIJf5k3EDh3Bg
2. **Response body** as Json string:

|  |
| --- |
| {  "**data**": {  "**username**": "admin",  "**authorities**": [  {  "**authority**": "ROLE\_ADMIN"  }  ],  "**permissions**": [  "dashboard.access",  "theme.colors.access",  "theme.typography.access",  "components.base.breadcrumbs.access",  ...  ]  }  } |

## Zuul Gateway

1. Create a new Maven project using Spring Initializr (<https://start.spring.io/>)
   * Spring Boot version: 2.1.2
   * Group: com.waumovil
   * Artifact: gateway
   * Dependencies:
     + Web
     + Eureka Client
     + Security
     + DevTools
     + Zuul
     + jjwt (you will add this later)
2. Your generated *pom.xml* file should look as follows:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.1.1.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>com.example</groupId>  <artifactId>zuul-2</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>zuul-2</name>  <description>Demo project for Spring Boot</description>  <properties>  <java.version>1.8</java.version>  <spring-cloud.version>**Greenwich.RC2<**/spring-cloud.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>**spring-boot-starter-web**</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>**spring-boot-devtools**</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>**spring-boot-starter-test**</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>**spring-cloud-starter-netflix-eureka-client**</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>**spring-cloud-starter-zuul**</artifactId>  <version>1.4.6.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>**spring-boot-starter-security**</artifactId>  </dependency>  <dependency>  <groupId>io.jsonwebtoken</groupId>  <artifactId>**jjwt**</artifactId>  <version>0.9.1</version>  <type>jar</type>  </dependency>  </dependencies>    <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>${spring-cloud.version}</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

1. Rename your *application.properties* to *application.yml* and include the following contents:

|  |
| --- |
| # Give a name to the zuul gateway  spring:  application:  name: gateway-service    # default port for authorization service  server:  port: 8762    # authorization service by default will register itself as a client  eureka:  client:  service-url:  default-zone: <http://localhost:8761/eureka/>  # A prefix that can added to beginning of all requests.  # zuul.prefix=/api  # Disable accessing services using service name (i.e. gallery-service).  # They should be only accessed through the path defined below.  zuul:  ignored-services: "\*"  # Map paths to services  routes:  auth-service:  path: /auth/\*\*  service-id: auth-service  strip-prefix: false  # Exclude authorization from sensitive headers  sensitive-headers: Cookie,Set-Cookie |

1. Zuul gateway must have at least the following components:
   * com.waumovil.gateway
     + **GatewayApplication**.java

your main class

* + com.waumovil.gateway.security
    - **CustomOncePerRequestFilter**.java

filter class to decode and parse the incoming Json Web Token on each request

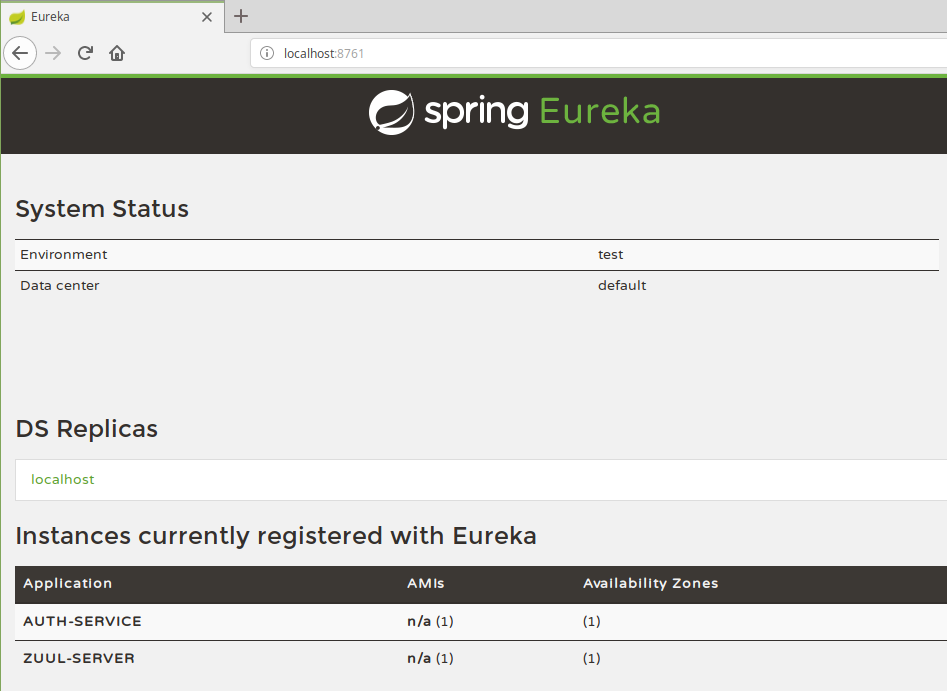
* + - **CustomWebSecurityConfigurerAdapter**.java

filter class to configure spring security

* + - **JwtConfig**.java

security properties

1. You should be able to see your service registered at Eureka server in [*http://localhost:8762*](http://localhost:8762/)



1. Test your authorization service using a tool like Postman. You should be able to get an object containing a Json Web Token -JWT- by making a POST request to [http://localhost:8762/auth](http://localhost:9100/auth).
   * **Request body** as Json string:

|  |
| --- |
| {  "**username**": "admin",  "**password**": "123456"  } |

* + **Response body** as Json string:

|  |
| --- |
| {  "**data**": {  "**type**": "bearer",  "**token**": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImF1dGhvcml0aWVzIjpbIlJPTEVfQURNSU4iXSwiaWF0IjoxNTQ3NzQwODM5LCJleHAiOjE1NDc4MjcyMzl9.w8EEbVNpqdzQkdXvltry7\_E7PuGS\_hy0zdhz00cfcy7lKwHmlgEafFp39PblC-Q8xnz\_ggvw4jIJf5k3EDh3Bg"  }  } |

1. Test your user service using a tool like Postman and the Json Web Token -JWT- generated in the previous step. You should be able to get an object containing user information by making a GET request to [http://localhost:8762/me](http://localhost:9100/me).
   * **Request headers**:
     + **Key**: Authorization
     + **Value**: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImF1dGhvcml0aWVzIjpbIlJPTEVfQURNSU4iXSwiaWF0IjoxNTQ3NzQwODM5LCJleHAiOjE1NDc4MjcyMzl9.w8EEbVNpqdzQkdXvltry7\_E7PuGS\_hy0zdhz00cfcy7lKwHmlgEafFp39PblC-Q8xnz\_ggvw4jIJf5k3EDh3Bg
2. **Response body** as Json string:

|  |
| --- |
| {  "**data**": {  "**username**": "admin",  "**authorities**": [  {  "**authority**": "ROLE\_ADMIN"  }  ],  "**permissions**": [  "dashboard.access",  "theme.colors.access",  "theme.typography.access",  "components.base.breadcrumbs.access",  ...  ]  }  } |

# Frontend Services

## System Settings

1. CoreUI Pro Vue template 2.0.3
2. NetBeans IDE 8.2
3. NPM 3.5.2

## External References

1. [https://scotch.io/tutorials/implementing-authentication-in-nuxtjs-app#toc-restricting-the-profile-page-to-only-logged-in-users](https://scotch.io/tutorials/implementing-authentication-in-nuxtjs-app" \l "toc-restricting-the-profile-page-to-only-logged-in-users)
2. <https://nuxtjs.org/examples/auth-external-jwt/>
3. <https://github.com/JiriChara/vue-kindergarten>
4. <https://codeburst.io/role-based-authorization-for-your-vue-js-and-nuxt-js-applications-using-vue-kindergarten-fd483e013ec5>

## Authentication

1. Install the auth dependency using the following command:

> npm install @nuxtjs/auth --save

1. Install the axios dependency using the followin command:

> npm install @nuxtjs/axios --save

1. Add the recently installed modules into your *nuxt.config.js* file:

|  |
| --- |
| modules: [  '**@nuxtjs/axios**',  '**@nuxtjs/auth**'  ] |

1. Configure your base URL by adding the following lines in your *nuxt.config.js* file:

|  |
| --- |
| axios: {  baseURL: '**http://localhost:8762**',  retry: {  retries: 3  },  debug: true  } |

1. Configure your authorization strategy and endpoints by adding the following lines in your *nuxt.config.j*s file:

|  |
| --- |
| auth: {  strategies: {  local: {  endpoints: {  **login**: { url: '**auth**', method: 'post', propertyName: '**data.token**' },  **user**: { url: '**me**', method: 'get', propertyName: '**data**' },  logout: false  }  }  },  plugins: [  '~/plugins/auth'  ]  } |

1. Add the following method into your Login.vue so you can perform authentication:

|  |
| --- |
| async login() {  try {  await this.**$auth.loginWith('local', {**  **data: {**  **username: this.username,**  **password: this.password**  **}**  **})**.then(function () {  console.log("Logged in!!!");  //console.log("Scope admin: " + this.$auth.hasScope("admin"));  });  this.$router.push('/Dashboard')  } catch (e) {  console.log("Error found " + JSON.stringify(e));  this.message = e.response.data.message  this.error = e.response.data.status + ":" + e.response.data.error  }  } |

1. Add the following method into your Vue file that performs logout operation:

|  |
| --- |
| async logout() {  //await this.$auth.fetchUser();  await this.**$auth.logout()**.then(function () {  console.log("Logged out!!!");  });  this.$router.push('/Login')  } |

1. Create a new file called auth.js into your /public\_html/plugins folder and add the following lines. This configuration is optional but will help getting more details on errors during authentication.

|  |
| --- |
| export default function ( { app }) {  !**app.$auth.onError**((error, name, endpoint) => {  console.log("\*\*\*\*\* AUTHORIZATION ERROR \*\*\*\*\*");  console.log("Error: " + error.response.data.status + ":" + error.response.data.error);  console.log("Message: " + error.response.data.message);  //console.error(name, error)  })  } |

1. In order to make the *auth.js* file work you must add it to your *nuxt.config.js* file within your auth-plugins section (not the regular plugins section). See the configuration below:

|  |
| --- |
| **auth**: {  strategies: {  local: {  endpoints: {  login: { url: 'auth', method: 'post', propertyName: 'data.token' },  user: { url: 'me', method: 'get', propertyName: 'data' },  logout: false  }  }  },  **plugins: [**  **'~/plugins/auth'**  **]**  } |

1. Create a new file called *index.js* into your */public\_html/store* folder and add the following lines:

|  |
| --- |
| export const getters = {  **isAuthenticated**(state) {  return state.auth.loggedIn  },  **loggedInUser**(state) {  return state.auth.user  }  } |

1. Now, in order to secure access to your pages you need to define some middleware components that will let the application know who can access or not a specific page. These middleware components will intercept each request and allow or deny access. Inside the folder */public\_html/middleware* create the following two files and add their respective contents as shown below:
   * *authenticated.js*

|  |
| --- |
| export default function ( { store, redirect }) {  // If the user is not authenticated  if (**!store.state.auth.loggedIn**) {  return redirect('/Login')  }  } |

* + *unauthenticated.js*

|  |
| --- |
| export default function ( { store, redirect }) {  // If the user is authenticated redirect to home page  if (**store.state.auth.loggedIn**) {  return redirect('/')  }  } |

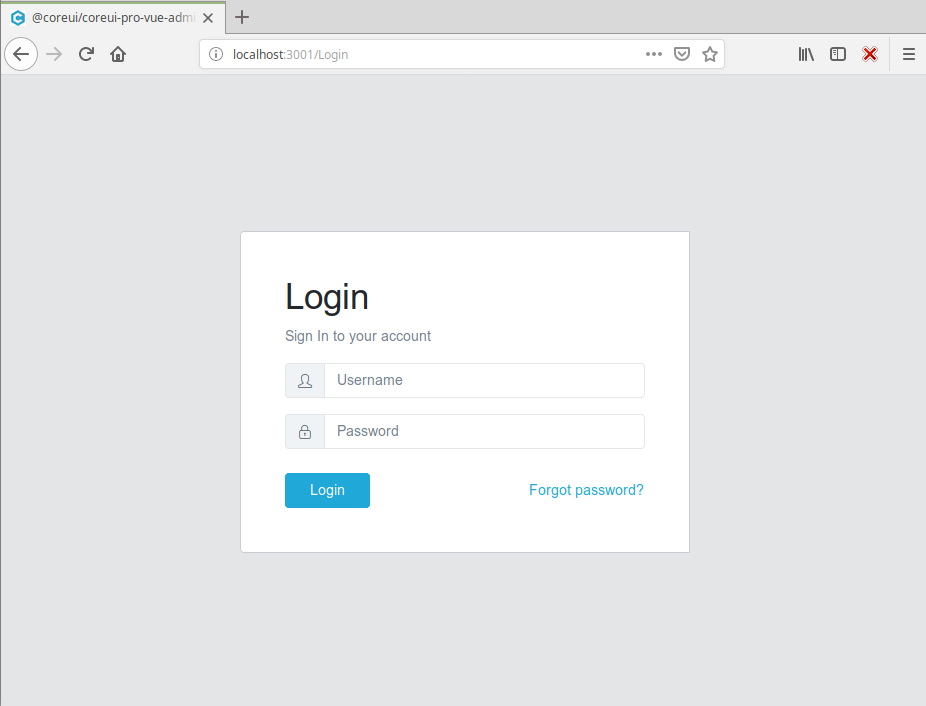
1. Once you have your middleware components ready, let’s add them to your pages to allow or deny access according to your needs. For example the *Login* page needs no authentication so you must use the *unauthenticated.js* component. On the other hand, your *Dashboard* page requires a user to be authenticated so you use the *authenticated.js* component as its middleware. See examples below:
   * *Login.vue*

|  |
| --- |
| ...  export default {  name: 'Login',  **middleware: 'unauthenticated',**  data() {  return {  ...  }  },  ... |

* + *Dashboard.js*

|  |
| --- |
| ...  export default {  name: 'dashboard',  **middleware: 'authenticated',**  ... |

1. Test your secured CoreUI Pro Vue template. You should have access to it on [http://localhost:3000](http://localhost:3000/). You must be automatically redirected to <http://localhost:3000/Login> since it is supposed that you have your *index.vue* secured using the *authenticated.js* middleware as explained in previous step.



## Authorization

Now that you have a fully authorized user logged into your application his/her user information, including permissions, is available through the store component of Vue. To include authorization features to your application you’ll make use of some of the authentication features you added in the Authorization section.

1. Install the Vue Kindergarten dependency using the following command:

> npm install vue-kindergarten --save

1. You need to define a component to refer to the store. Create a file named *child.js* inside */public\_html* folder ant put the contents shown below:

|  |
| --- |
| export default (store) => store; |

1. Make the following changes to your *nuxt.config.js* file:
   * Register the vue-kindergarten plugin:

|  |
| --- |
| plugins: [  '**~/plugins/vue-kindergarten**'  ] |

* + Add a middleware for router:

|  |
| --- |
| router: {  middleware: 'vue-kindergarten'  } |

1. Create a new file called vue-kindergarten.js inside the /public\_html/plugins folder and add the following contents. This will make your child available to all your pages.

|  |
| --- |
| import Vue from 'vue';  import VueKindergarten from 'vue-kindergarten';  import child from '@/child';  Vue.use(VueKindergarten, {  child: (store) => {  return store;  }  }); |

1. Create a new file called RouteGoverness.js inside the /public\_html/governess folder and add the following contents. This governess will guard all access attempts to all your resources. In case a user doesn’t have permission to perform a specific action it will be rejected and sent to homepage.

|  |
| --- |
| import { HeadGoverness } from 'vue-kindergarten';  export default class RouteGoverness extends HeadGoverness {  guard(action, { redirect }) {  if (this.isNotAllowed(action)) {  redirect('/');  }  }  } |

1. Once you have your governess component, you need to define the limits of your secured areas. We will call these perimeters, so let’s create some basic perimeters:

* BasePerimeter.js

This will serve as a base class for the rest of perimeters which will inherit the log function. Here you can add functions that you will be using in the rest of perimeters.

|  |
| --- |
| import { Perimeter } from 'vue-kindergarten';  export default class BasePerimeter extends Perimeter {    log(action, isAllowed) {  console.log("Store = " + JSON.stringify(this.child.state));  console.log(action + ' ' + this.name + ': ' + (isAllowed ? 'Allowed' : 'Denied'));  }    } |

* DashboardPerimeter.js

This will contain all specific abilities required to access resources in Dashboard page.

|  |
| --- |
| import BasePerimeter from './BasePerimeter';  export default new BasePerimeter({  name: 'Dashboard',    purpose: 'dashboardPerimeter',    debug: true,    can: {  access() {  if (this.debug && this.child.state.auth.loggedIn) this.log('Access', this.child.state.auth.user.permissions.includes('dashboard.access'))    return this.child.state.auth.loggedIn && this.child.state.auth.user.permissions.includes('dashboard.access');  }  }  }); |

1. Create a new middleware called vue-kindergarten.js and put the following contents:

|  |
| --- |
| import { createSandbox } from 'vue-kindergarten';  import RouteGoverness from '@/governess/RouteGoverness';  import \* as perimeters from '@/perimeter';  import child from '@/child';  export default (context) => {  const {route, error, redirect, store, isServer} = context;  route.matched.some((routeRecord) => {  var routeParts = routeRecord.name.split("-");    var perimeterName = routeParts[routeParts.length-1][0].toUpperCase() + routeParts[routeParts.length-1].substr(1);    const perimeter = perimeters[perimeterName + 'Perimeter'];  const Governess = RouteGoverness;  if (perimeter) {  const sandbox = createSandbox(child(store), {  governess: new Governess(context),  perimeters: [  perimeter  ]  });  return sandbox.guard('access', {redirect});  }  });  } |

1. Register all your perimeters in a /public\_html/perimeter/index.js file as shown below. You must register all your perimeters here so they can be found by your vue-kindergarten.js middleware.

|  |
| --- |
| import **DashboardPerimeter** from './**DashboardPerimeter**';  ...  export {  **DashboardPerimeter**  ….  }; |

1. To make use of your perimeters you must add a reference to them in your pages. For example the Dashboard page will make use of DashboardPerimeter so add it as shown below in your Dashboard script section as shown below:

|  |
| --- |
| …  import DashboardPerimeter from '@/perimeter/DashboardPerimeter';  …  export default {  name: 'dashboard',  middleware: 'authenticated',  **perimeters: [**  **DashboardPerimeter**  **]**,  … |

1. To test your authorization implementation you must manually set the exact path to a specific resource, for example <http://localhost:3000/Dashboard>, <http://localhost:3000/theme/Colors>, etc. To configure the required permissions to a specific page you must edit your perimeters. In the following section you will complete the authorization settings in order to show/hide items in your navigation bar.

## Navigation

Your authorization feature is not complete if you don’t configure your navigation bar items to show/hide based on the logged in user.

1. Create a global perimeter to configure the required abilities to access your navigation bar resources. Create the file /public\_html/perimeter/DefaultPerimeter.js and add the following contents:

|  |
| --- |
| import { Perimeter } from 'vue-kindergarten';  export default new Perimeter({  name: 'Default',  purpose: 'defaultPerimeter',  debug: true,  can: {  **accessDashboard**() {  return this.accessDashboard();  },  },  **accessAllowed**(permission) {  // user must be authenticated and have required authorization to access a resource  return this.child.state.auth.loggedIn && this.child.state.auth.user.permissions.includes(permission);  },  **accessDashboard**() {  return this.accessAllowed('**dashboard.access**');  }  }); |

1. Modify your /public\_html/components/sidebar/SidebarNav.vue file including the following directive in your menu elements. This

|  |
| --- |
| v-if="item.permission && $isAllowed(item.permission)" |

1. Modify your /public\_html/layouts/\_nav.js file. This is your menu definition. Add the required ability to each item as shown below. The permission you set in there correspond to one of the abilities you configured in your global perimeter.

|  |
| --- |
| export default {  items: [  {  name: 'Dashboard',  url: '/Dashboard',  icon: 'icon-speedometer',  badge: {  variant: 'primary',  text: 'NEW'  },  **permission: 'accessDashboard'**  } |

1. Now your authorization feature is complete so you can fully test your application. The user can only see and access the resources according to his permissions provided by the authorization service in the backend.