Getting Started

Ignity - Vue Admin Kit is totally developer-friendly, created with an ecosystem that adapts to the most diverse scenarios and needs, composed of reusable, editable components that allow extreme gains in efficiency and development speed. Ready for small and large corporations, it naturally introduces micro services and reactivity for high performance in real-time applications. You can also display data on our components easily connected to your legacy server through web, socket, rest and much more methods.

All .vue components have been created to achieve the perfect consistency of spacing, alignment and shape on the various skins included in the package.

For color adjustment, spacing and much more, a SCSS file with clearly documented variables separated by skin allows various adjustments.

With this kit you have a multitude of customizations with changes of few centralized parameters. To change the layout (menu, orientation, size of containers, etc.) a central .json file enables many customizations.

In the "How to Customize" section, you will find more details on how to customize your project.

Premises

This project uses several modern tools to provide a productive means for you or your development team, however to be more effective in the process of building a software using this tool, we start from the following assumptions regarding your skills:

- You have some knowledge with Javascript, NodeJs, NPM and related universe.
- You have some knowledge in Vue and/or reactive technologies.
- You have some knowledge of CSS processors (like SCSS).
- You have some knowledge of pure HTML and CSS.
- You have some knowledge of Javascripts.

If you don't master any of the subjects above, don't be alarmed, most subjects have a low learning curve, you can improve your knowledge by seeing the functional examples in the project, using them as a basis for your changes according to your needs.

Installation

Install using npm

- Install latest node.js: https://nodejs.org
- Go to the directory where you extracted the project files.
- Install node modules by running terminal command: `npm install`
- Run the app using: `npm run dev`
- For build production files use `npm run build`

Install using yarn

- Install latest node.js: https://nodejs.org
- Install latest yarn package manager: https://yarnpkg.com/
- Go to the directory where you extracted the project files.
- Install node modules by running terminal command `yarn install`
- Run the app `yarn serve`
- For build production files use `yarn build`

Wait for the build to finish.

To access the theme locally, enter the address in your preferred browser:
http://localhost:3031/

For access to complete documentation:
http://localhost:3031/admin/doc/basic/getting-started

* First build can take a long time, but from the second this time is reduced considerably due to the cache generated by the webpack.

Features

We bring together current technologies used by well-established companies in the market, with this solution we tackle the problem on several fronts.

A short list of the main resources contained in this project:

- Modern and clean look
- Search engine friendly
- Hundreds of Reusable Components
- Universal code, rendered in client and/or server (much more efficient for SEO, loved by search engines)
- Maximum productivity with reactive web and reusable components (using vue)
- Easy custom to your company style (SCSS variables)
- Automatic routes based on file structure
- Enables real-time updates easily.
- Stay in the top of the search engine rankings. HTML rendered on the server
- Perfect for apps like chats, boards and more.
- Use several databases, exchanging only one line of code (thank you feathers.js) :)
- Several modules with real example working (including access to query and database editing)
- Ready and functional authentication example
- Clean, commented and easily customizable codes
- Enterprise Grade Ready
- Migrate to microservices or maintain your legacy by connecting to multiple databases.

Project Structure

This kit is designed to give you access to components separated by modules, each responsible for a business domain. The standard structure of the application (provided by Nuxt.js) gives you an excellent starting point to organize your application in an intuitive way.

Here is a summary of the main directories and their usefulness:

In the `/client` folder contained in the `project root`, we have the following folders:

/components	Main project directory, where you will make most of your changes. In this directory you will find the .vue files responsible for the design and behavior of all the components that make up the kit.
/layouts	Contains special .vue files that define layouts that are used by the pages included in the project.
/pages	A folder containing the routes of your application. Nuxt.js reads all the .vue files inside this directory and creates the application's router based on their respective directories and names. Learn more here:
/store	This project comes with a store designed to centralize the handling of information displayed throughout the project. Learn more at Vuex
/middleware	This folder contains js files with functions that are executed in specific events of the project execution lifecycle. These functions are useful for various tasks such as: authentication, validation, skins management and much more.
/plugins	The plugins directory allows you to load and register plugins to be used in every application in an organized and highly maintainable way.
/static	It contains all the static content that will be consumed by the project, such as images, fonts and icons.
/styles	Directory of all SCSS files, used to generate the global project style sheet, skins definition, colors, spacing, rounding, and more, can be made in the .scss files in this directory.
/util	This directory contains JS files that help in several small functions, such as control of menu items, fake data generation (for demonstration) and similar.

Creating new page

It is very simple to create new pages and routes in your project, for this Nuxt.js automatically generates the vue-router configuration based on your file tree of Vue files inside the pages directory.

So by creating only one .vue file, you create the route for browser access, based on the directory of that file.

Generating routes

For the correct generation of access routes to your new pages, locate the /client/pages folder inside the root of your project. In this folder create a .vue file following the structure below:

Create file `details.vue` inside `client/pages/user/` folder:

```
<template>
  <h2>User Details</h2>
</template>
```

At the end you must have this file structure:

```
pages/
--| user/
-----| details.vue
```

If you are running the project in development mode as taught in the previous session, the project will perform the automatic reload due to the change in the file structure. You should be able to access address http://localhost:3031**/user/details** and view the contents of the created vue file. Remember that within the pages you can use any vue component available in the project.

To navigate between pages, we recommend to use the `<nuxt-link>` component.

Nuxt.js automatically generates the vue-router configuration based on your file tree of Vue files inside the `pages` directory.

will automatically generate:

```
router: {
  routes: [
    {
      name: 'user',
      path: '/user/details',
      component: 'pages/user/details.vue'
    }
  ]
}
```

Read more about pages in the official documentation of Nuxt.

Dynamic Routes

To define a dynamic route with a parameter, you need to define a .vue file OR a directory **prefixed by an underscore**.

This file tree:

```
pages/
--| _slug/
-----| comments.vue
-----| index.vue
--| users/
-----| _id.vue
--| index.vue
```

will automatically generate:

```
router: {
  routes: [
    {
      name: 'index',
      path: '/',
      component: 'pages/index.vue'
    },
    {
      name: 'users-id',
      path: '/users/:id?',
      component: 'pages/users/_id.vue'
    },
    {
      name: 'slug',
      path: '/:slug',
      component: 'pages/_slug/index.vue'
    },
    {
      name: 'slug-comments',
      path: '/:slug/comments',
      component: 'pages/_slug/comments.vue'
    }
  ]
}
```

As you can see the route named `users-id` has the path `:id?` which makes it optional, if you want to make it required, create an `index.vue` file in the `users/_id` directory instead.

Validate Route Params

Nuxt.js lets you define a validator method inside your dynamic route component.

In this example: `pages/users/_id.vue`

```
export default {
  validate ({ params }) {
    // Must be a number
    return /^\d+$/.test(params.id)
  }
}
```

If the validate method does not return `true` or a `Promise` that resolve to `true`, or throws an Error, Nuxt.js will automatically load the 404 error page or 500 error page in case of an error.

More information about the validate method: API Pages validate

Nested Routes

Nuxt.js lets you create nested route by using the children routes of vue-router.

To define the parent component of a nested route, you need to create a Vue file with the **same name as the directory** which contain your children views.

Warning: don't forget to include `<nuxt-child/>` inside the parent component (`.vue` file).

This file tree:

```
pages/
--| users/
-----| _id.vue
--| users.vue
```

will automatically generate:

```
router: {
  routes: [
    {
      path: '/users',
      component: 'pages/users.vue',
      children: [
        {
          path: '',
          component: 'pages/users/index.vue',
          name: 'users'
        },
        {
          path: ':id',
          component: 'pages/users/_id.vue',
          name: 'users-id'
        }
      ]
    }
  ]
}
```

Dynamic Nested Routes

Dynamic Nested Routes

This scenario should not often happen, but it is possible with Nuxt.js: having dynamic children inside dynamic parents.

This file tree:

```
pages/
--| _category/
-----| _subCategory/
--------| _id.vue
--------| index.vue
-----| _subCategory.vue
-----| index.vue
--| _category.vue
--| index.vue
```

will automatically generate:

```
router: {
  routes: [
    {
      path: '/',
      component: 'pages/index.vue',
      name: 'index'
    },
    {
      path: '/:category',
      component: 'pages/_category.vue',
      children: [
        {
          path: '',
          component: 'pages/_category/index.vue',
          name: 'category'
        },
        {
          path: ':subCategory',
          component: 'pages/_category/_subCategory.vue',
          children: [
            {
              path: '',
              component: 'pages/_category/_subCategory/index.vue',
              name: 'category-subCategory'
            },
            {
              path: ':id',
              component: 'pages/_category/_subCategory/_id.vue',
              name: 'category-subCategory-id'
            }
          ]
        }
      ]
    }
  ]
}
```

Middleware

Middleware lets you define custom functions that can be run before rendering either a page or a group of pages.

Every middleware should be placed in the `middleware/` directory. The filename will be the name of the middleware (`middleware/auth.js` will be the `auth` middleware).

A middleware receives the context as first argument:

```
export default function (context) {
  context.userAgent = process.server ? context.req.headers['user-agent'] : navigator.userAgent
}
```

In universal mode, middlewares will be called server-side once (on the first request to the Nuxt app or when page refreshes) and client-side when navigating to further routes. In SPA mode, middlewares will be called client-side on the first request and when navigating to further routes.

The middleware will be executed in series in this order:

1. `nuxt.config.js` (in the order within the file)
2. Matched layouts
3. Matched pages

A middleware can be asynchronous. To do this, simply return a `Promise` or use the 2nd `callback` argument:

`middleware/stats.js`

```
import axios from 'axios'

export default function ({ route }) {
  return axios.post('http://my-stats-api.com', {
    url: route.fullPath
  })
}
```

Then, in your `nuxt.config.js`, use the `router.middleware` key:

`nuxt.config.js`

```
export default {
  router: {
    middleware: 'stats'
  }
}
```

Now the `stats` middleware will be called for every route change.

You can add your middleware to a specific layout or page as well:

`pages/index.vue` or `layouts/default.vue`

```
export default {
  middleware: 'stats'
}
```

Learn More

Learn More: Auto Routing w/ Nuxt

Change layout and structure opts

There are 2 groups of customizations that can be performed in the project using files with user-friendly variables. Layout variations, such as menu layout, container style, and other similar, are performed by a json file.

This file is located in:

`/[Project Dir]/config/layouts.json`

Each layout is composed of a combination of the following variables:

headerBoxed	Determines whether the header is in a fluid container or not.
headerBg	Sets the background color of the header
headerBgOnScroll	Determines the background color of the header while the scroll bar was scrolled down.
headerExtended	The extended version adds a horizontal bar to the end of the header
headerIconLighten	Enables the clear version of the header icons.
breadcrumbHide	Hides the main breadcrumb displayed in the layout.
contentBoxed	Determines whether the main content is in a fluid container or not.
menuLogoBg	Sets the background color of the logo on sidemenu
menuCompacted	Determines if the left side menu will be displayed in the compact version.
menuMinimized	Determines if the left side menu will be displayed in a minimized way.
menuHide	Hides the main side menu displayed in the layout.
menuIconIndex	The project has 2 types of icons enabled by default in the side menu. Set to `0` for `Line Awesome` or `1` for `DripIcons`
topMenuHide	Hides the top menu menu displayed in the layout.
contextBarHide	Hides the contextbar on app pages.
skin	Fill in the name of the selected skin, e.g. blue, dark-blue, green and more

skin	Fill in the name of the selected skin, e.g. blue, dark-blue, green and more

Change color schema

For changes in schema colors of the project, spacing and other style variables, use the SCSS file corresponding to the selected skin.

The name of the color variables follows the bootstrap variable pattern (`primary`, `secondary`, `warning`, `danger`, `info`, `dark`, `light`).

In addition, we have added 2 additional colors to facilitate the creation and maintenance of visual components, color `theme1` and color `theme1inv`, which help to highlight text and background color respectively.

The SCSS variables are commented in each skin file for better understanding.

This file is in:

`/[Project Dir]/client/styles/variables/skin/[Skin Name].scss`

Creating new component

As with any vue project you can create new components inside path `/client/components/`

In this project in the root folder of the components: `/[project root]/client/components`

we have 2 groups in root subfolders, specific components for modules located in the `Module` folder, and components used as the base, and reusable throughout the project, located in the `Base` folder

You can follow the organization proposed by us (in the current project) or use your preferred organization.

```
client/
--| components/
----| Module/
------| Chat/
--------| ChatDetails.vue
----| Base/
------| Card/
--------| CardTitle.vue
```

This project register globally all components of bootstrap, so just add the tag with the name of the component you want to use and enjoy all the benefits that the integration of Vue with Bootstrap can offer.

To check the complete list, and the updated documentation of the available components, you can access the library's official website by clicking here: Bootstrap Vue

This is a simple component that prints the value of a variable within an alert component, provided by Bootstrap Vue:

```

<template>
<b-alert>{ { greeting } } World!</b-alert>
</template>

<script>
module.exports = {
  data: function () {
    return {
      greeting: 'Hello'
    }
  }
}
</script>
```

Learn More

The more you know about Vue and Bootstrap, the more agile your development will be using our seed project, so we have separated 2 external links that are mandatory to read in order to make progress in your new venture.

Creating Component File   Bootstrap Vue Components

Why Feathers?

This development kit can be used with any server technology, however, to make your life easier, we have embarked on our server framework solution that can enhance your entire development cycle.

Feathers provides a lot of the things that you need for building modern web and mobile applications. Here are some of the things that you get out of the box with Feathers. All of them are optional so you can choose exactly what you need. No more, no less. We like to think of Feathers as a "batteries included but easily swappable" framework.

Feathers Features

Instant REST APIs	Feathers automatically provides REST APIs for all your services. This industry best practice makes it easy for mobile applications, a web front-end and other developers to communicate with your application.
Unparalleled Database Support	With Feathers service adapters you can connect to all of the most popular databases, and query them with a unified interface no matter which one you use. This makes it easy to swap databases and use entirely different DBs in the same app without changing your application code.
Real Time	Feathers services can notify clients when something has been created, updated or removed. To get even better performance, you can communicate with your services through websockets, by sending and receiving data directly.
Cross-Cutting Concerns	Using "hooks" you have an extremely flexible way to share common functionality or concerns . Keeping with the Unix philosophy, these hooks are small functions that do one thing and are easily tested but can be chained to create complex processes.
Universal Usage	Services and hooks are a powerful and flexible way to build full stack applications. In addition to the server, these constructs also work incredibly well on the client. That's why Feathers works the same in NodeJS, the browser and React Native.
Authentication	Almost every app needs authentication so Feathers comes with support for email/password, OAuth and Token (JWT) authentication out of the box.
Pagination	Today's applications are very data rich so most of the time you cannot load all the data for a resource all at once. Therefore, Feathers gives you pagination for every service from the start.
Error Handling	Feathers removes the pain of defining errors and handling them. Feathers services automatically return appropriate errors, including validation errors, and return them to the client in an easily consumable format.

Creating New Service

In Feathers any API endpoint is represented as a service, which we already learned about in the basics guide. To generate a new service, we can run the following command from the project root using the command line:

```
feathers generate service
```

A series of questions will be asked in order for you to define the basic data of the service.

First we have to choose what kind of service we'd like to create. You can choose amongst many databases and ORMs but for this guide we will go with the default NeDB. NeDB is a database that stores its data locally in a file and requires no additional configuration or database server.

Next, when asked for the name of the service, enter `messages`. Then keep the default path (`/messages`) by pressing enter.

The *database connection string* can also be answered with the default. (In this case of NeDB, this is the path where it should store its database files.)

The generated files

As we can see, several files were created:

- `src/services/messages/messages.service.js` - The service setup file which registers the service in a configure function
- `src/services/messages/messages.hooks.js` - A file that returns an object with all hooks that should be registered on the service.
- `src/models/messages.model.js` - The model for our messages. Since we are using NeDB, this will simply instantiate the filesystem database.
- `test/services/messages.test.js` - A Mocha test for the service. Initially, it only tests that the service exists.

Learn More

If you are interested in going deeper into our optional backend, we strongly suggest that you read the official project documentation in the following external link:

Feathers Backend

Why?

A universal application is about preloading your application on a web server and sending rendered HTML as the response to the browser for every route in your app in order to improve SEO, make loading happen faster, along with many other benefits.

What is Nuxt.js

Nuxt.js is a higher-level framework that builds *on top* of Vue. It simplifies the development of universal or single page Vue apps.

Nuxt.js abstracts away the details of server and client code distribution so you can focus on application development. The goal with Nuxt is for it to be flexible enough for you to use as a main project base. Because most of what Nuxt does happens during the development phase, you get a lot of features with only a few extra kilobytes added to your JavaScript files.

Nuxt.js helps you write universal apps more simply

Building universal applications can be tedious because you have to do a lot of configuration on both the server side and client side.

This is the problem Nuxt.js aims to solve for Vue applications. Nuxt.js makes it simple to share code between the client and the server so you can focus on your application's logic.

Nuxt.js gives you access to properties like `isServer` and `isClient` on your components so you can easily decide if you're rendering something on the client or on the server. There's also special components like the `no-ssr` component which is used to purposely prevent the component from rendering on the server side.

Lastly, Nuxt gives you access to an asyncData method inside of your components you can use to fetch data and render it on the server side.

That's the tip of the iceberg of how Nuxt helps you with creating universal applications. Click here to learn more about what Nuxt offers for rendering Universal applications.

Libraries Included

With this ecosystem, the kit provides a lot of native and added features through third-party packages.

Here is a list of the main packages imported into the project:

- Powered by Vue
- Nuxt
- Bootstrap 4.3
- Bootstrap Vue
- Feathers Optional Backend
- Vuex
- Vue Router Enhaced with Auto Routing
- Axios
- SASS Powered
- Charts
    - Apex Charts
    - eCharts
    - Morris
- Vue Kanban
- Vue Simple Calendar
- Vue Upload Multiple Image
- Vue Quill w/ vue2-editor
- JdentIcon
- Icons and Fonts from CDN
    - DripIcons
    - Font Awesome 5
    - Linear Icons
    - Material Icon
    - Vivid Icons
    - Poppins

Changelog

1.0: Initial release