DOCUMENTATION
FULLSTACK SSR WEB APP

Interactive web technologies and responsive medias design

TABLE OF CONTENT

Warranty and Updates

- Security and responsiveness warranty
- Quality of Service warranty
- System updates
- Creative content updates

Certification

SSL certificate

Hosting

- IPFS (static)
- Local cloud
- Heroku

Guidelines

- Modularity
- Code guidelines
- Commented code
- Documented code

User Guide

- Style
- Generative art techniques
- Responsive design
- Client-side routing
- State management
- Code editor
- Graphical and interactive experiences
- Sass support
- Image processing
- Server-side rendering

Troubleshooting

- Common issues and solutions

ABSTRACT

The website is a dynamic site with a content management system. The site is divided into two parts: the front-end, which includes the graphical structure of the site, and the back-end, which consists of a JavaScript web application that includes data management and interface logic (interface with a database).

The front-end and the back-end both form separate web services that are able to interface and exchange data in order to create the architecture of a dynamic web application. This application is called dynamic due to its ability to manage content that users can interact with based on the logic implemented at the back-end level.

The front-end and its features have a responsive and interactive design. Responsive design refers to the ability of the website to adapt to different display environments (desktop, mobile) as well as the spontaneity of execution and rendering (display).

Interactive content is generated using various programming techniques called creative code or generative art, which translate into the generation of animations, graphic and artistic elements based on code, enhancing the user experience.

It is therefore possible to create a graphically consistent environment that creates an artistic experience during use. Based on the theme, form and colors, the design follows a graphic code that creates consistency between the different elements called components in the context of modular design. Modular web design refers to different elements that are able to adapt and combine to form the considered web architecture.

The back-end, which is responsible for the logic of the site's various functions, is developed according to the structure of an API. This is a generic element of the architecture that can serve data to multiple front-ends (graphic element of the application). It also allows for the secure management of data in terms of interactions with databases and various services on the internet that extend functionality. As a whole, this structure is considered a content management system (CMS), hosting the implemented logic such as user identification systems. Content management is an element of various internet mediums such as e-commerce that addresses the following issues:
Hosting information on the web in an accessible and secure manner.

A separate back-end, incorporating encryption methods, data validation (at multiple scales and levels of abstraction), as well as identification systems, validates this approach.
Interfaces with various data services on the internet (APIs) are integrated into the front-end or back-end depending on execution conditions (requests). On the front-end, these interfaces take the form of graphical elements that gather web functionality around the considered architecture.

On the back-end, these interfaces enable the automation of systems and requests. These APIs make it possible to integrate the Ethereum blockchain in order to retrieve ERC721 tokens or interact with decentralized web contracts (payment methods).

The Internet of Things (IoT) is presented as an expanding and innovative market; the internet of things allows for interface with electronics in order to retrieve sensor information or remotely control these elements. This is referred to as business intelligence in the case of applications that benefit from the retrospective analysis of this data in order to guarantee or optimize services. Applications developed benefit from these interfaces with the IoT, allowing for the integration of modules available on the market. The integration of these services into a web application allows for the real-time visualization of data (graphs, tables, trees).

PROJECT TREE

Fullstack Nuxt.js SSR Application
    |
    |--> Server
    | |
    | |--> Load Balancer
    | |
    | |--> Express.js (Web application framework for Node.js)
    | | |
    | | |--> MongoDB (NoSQL database)
    | | |--> Data Validation (Validation at multiple scales and levels of abstraction)
    | | |--> Identification Systems (User identification systems)
    | | |--> Encryption Methods (Security mechanism)
    | | |--> Routing (Server-side routing)
    | | |--> Testing Framework (Unit/integration testing)
    | | |--> Deployment Tool (Continuous integration/delivery)
    | | |--> APIs (Interfaces with external data services)
    | | |--> Ethereum Blockchain (Retrieving ERC721 tokens and interacting with decentralized web contracts)
    | | |--> IoT (Interface with electronics to retrieve sensor information or remotely control elements)
    |
    |--> Nuxt.js (JavaScript framework for server-side rendering)
    | |
    | |--> Vue.js (JavaScript framework for building user interfaces)
    | | |
    | | |--> Components (Modular design elements that can adapt and combine to form the web architecture)
    | | |--> HTML (Hypertext Markup Language for structuring content on the web)
    | | |--> CSS (Cascading Style Sheets for styling web content)
    | | |--> JavaScript (Programming language for adding interactivity to web pages)
    | | |--> Creative Code (Generative art techniques for creating animations and graphical elements based on code)
    | | |--> Responsive Design (Ability to adapt to different display environments and spontaneous execution/rendering)
    | |--> Webpack (Module bundler)
    | |--> Babel (JavaScript compiler)
    | |--> Router (Client-side routing)
    | |--> PWA (Progressive Web App)
    | |--> Axios (Promise-based HTTP client)
    | |--> Bootstrap (CSS framework for building responsive, mobile-first websites)
    | |--> D3.js (JavaScript library for manipulating documents based on data)
    | |--> SSR (Server-side rendering)
    | |--> Vuex (State management pattern for Vue.js applications)
    | |--> Vue Router (Client-side routing for Vue.js applications)
    | |--> Brace (Code editor library)
    | |--> P5.js (JavaScript library for creating graphics and interactive experiences)
    | |--> Node-Sass (Library that provides binding for Node.js to libsass)
    | |--> Node-Vibrant (Library for extracting dominant colors from images)
    | |--> Vue-Masonry-Css (Library for creating masonry-style layouts with Vue.js

KEY FEATURES

Creative coding animation to enhance user experience

Data visualization for exploring data through visual analysis

Generative art design

Template theming for a visually appealing website

Responsive design to adapt to different devices and display environments

Scalability to handle increased traffic and workload

Multilingual support to target a wider audience

User sessions for clients to login and manage their accounts

Role sessions for role-based permissions

Post system to manage articles, blog posts, images, and products

Social interactions such as likes, comments, ratings, notes, and to-do lists

Collections system for sorting, tagging, hashtagging, and categorizing

Storage for multiple file uploads, including images and videos

Admin dashboard for managing users, roles, permissions, posts, and collections

Intelligent modular components and tree systems for data architecture

Dynamic database for storing and retrieving data safely

Real-time data manipulation and management using a NoSQL database

File storage for uploading, storing, and displaying single or multiple files

Dynamic management of product entries and stock calculation

Cloud notifications for instant updates on website activity

Admin backend for managing and moderating the website

API integration with services, payment systems, databases, and delivery tracking

Cloud automation to run tasks automatically from the cloud

Blockchain integration to interact with smart contracts and NFTs

AI systems such as chatbots, live assistants, automatic translators, and bug detectors

IoT integration to retrieve data from sensors and control electronic devices from the website

Server-side routing to handle different routes and pages

Cloud notifications for instant updates on website activity

Admin backend for managing and moderating the website

API integration with services, payment systems, databases, and delivery tracking

Cloud automation to run tasks automatically from the cloud

Blockchain integration to interact with smart contracts and NFTs

AI systems such as chatbots, live assistants, automatic translators, and bug detectors

IoT integration to retrieve data from sensors and control electronic devices from the website

Server-side routing to handle different routes and pages

Unit and integration testing frameworks

Continuous integration and deployment process

Accessibility guidelines to ensure the website is accessible to all users

SEO optimization to improve search engine visibility

Social sharing to make it easy to share website content on social networks

SSL security for secure payload transactions and data validation

Error handling and logging to improve security

Analytics to generate logs and reports to improve communication and user experience

Search functionality to help users find what they are looking for

Geolocation to provide location-based content to users

Decentralized hosting to prevent downtime

noSQL database (MongoDB) for personalized data management

Javascript framework (Nuxt.js) for a responsive mobile-first architecture

HTML, CSS, and JavaScript for structured and styled content

Javascript libraries (Bootstrap, p5.js, d3.js) for interactive and dynamic elements

Warranty and updates for extended security and responsiveness,

Quality of Service, and system updates

SSL certificate for secure data exchanges and requests

Hosting options including IPFS (static), local cloud, and Heroku

Modular application components and reusable code using the Vue.js framework

Commented and documented code for authenticity and ease of maintenance

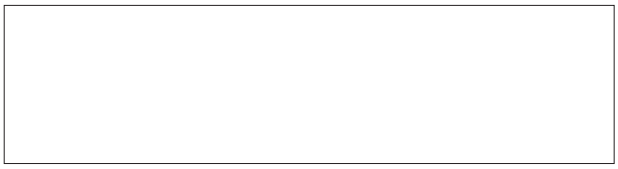
User guide with guidelines for style, generative art techniques, responsive design, client-side routing, state management, and more

Troubleshooting guide for common issues and solutions

INTERACTION DESIGN

Creative Coding Animation

Dynamic and live rendering animations in the contexte of the website. Interactive graphic elements rendering live on the browser. Creating visual and interactive experiences for users.



Data Visualization

Data visualization tools to display analytic informations in a meaningful way.

Generative Art

Digital art generated by code and algorithms to embelilish the artistic concept or storytelling of a website.

USER EXPERIENCE

Template theming

Personnalize the design of your website post-production. Change style, type, colors without coding. Update your front page or landing page with the latest content.

Responsivness

Modular and responsive framework mobile first.

Server Side Rendering.

Scalability

Modular component based design.

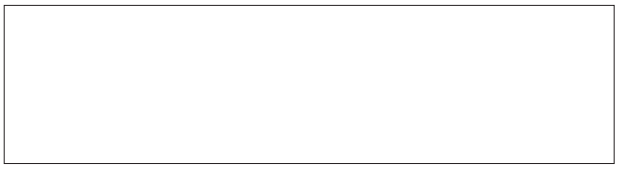
Language support

Fr/En/De/It

CONTENT SYSTEM

User session

User can login to the app and manage their accounts and content safely..



Role session

Access control based on permissions.
Restric areas, functionnalities.

Post system

Create and manage content dynamically.

Social interactions

Users can create comments, like posts, rate content.

Collections

Manage collections of items.

Storage

Host, publish and retrieve files safely.

ADMIN MANAGEMENT

Admin Dashboard

Manage style, users and content through an Admin board which access is restricted to authorized users.

User and Role management

Manage users and their different access to the website

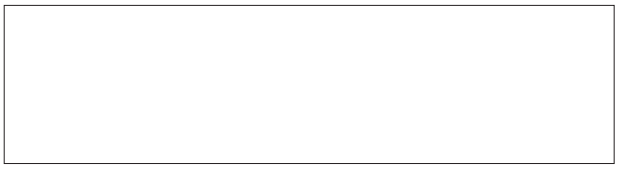
Post and Collections management

Manage the content and publications.

LOGIC

Data Architecture

System handing content



Dynamic Database

Local databse
Peer database
Database as a service

Real time data manipulation

File Storage

Users can create comments, like posts, rate content.

Database Cluster

Synchronize databse nodes accross different peers

INTERFACES

Admin Dashboard

Manage style, users and content through an Admin board which access is restricted to authorized users.

User and Role management

Manage users and their different access to the website

Post and Collections management

Manage the content and publications.

STYLE

Apply style only to the component without affecting the global CSS

```
<style scoped></style>
```

**Select all the .link descendant that
are not in elements of the root.**

```
:not(:root) >>> .link
```

Define an external CSS file to style bootstrap variable globally

```
npm install sass-loader node-sass
```

Declare the npm module for utilisation.

```
module.exports = {
...
  modules: ['bootstrap-vue/nuxt',...]
...
}
```

Define loaders to load properly the file.

```
//nuxt.config.js

module: {
    rules: [
        {
          test: /\.s[ac]ss$/i,
          use: ['style-loader','css-loader','sass-loader',],
        },
        ...
    ],
},
```

Configure Bootstrap CSS

```
//nuxt.config.js
module.exports = {
  ...
  modules: ['bootstrap-vue/nuxt',...],
  bootstrapVue: {
    bootstrapCSS: false,
    bootstrapVueCSS: false
  }
  ...
}
```

Import custom CSS file

```
//nuxt.config.js
module.exports = {
  ...
  css: [...,'@/assets/scss/custom.scss'],
  ...
}
```

assets/scss/custom.scss

Override Bootstrap CSS Variables

```
$palette11:          #f44260;
$palette12:          #3c5ba0;

$palette11: $palette11 !default;
$palette12: $palette12 !default;

$theme-colors: () !default;
$theme-colors: map-merge(
  (
    'palette11': $palette11,
    'palette12': $palette12,
  ),
  $theme-colors
);

$font-family-base: 'Helvetica Neue';

//then bootstrap & bootstrap-vue should be added after
//so that they inherit the custom SCSS variables
@import 'bootstrap/scss/bootstrap.scss';
@import 'bootstrap-vue/src/index.scss';
```

Define plugins to export nuxt.js context into external files and create modules.

```
// plugins/main.js

import Vue from 'vue'
import VueMasonry from 'vue-masonry-css'

import DateFilter from './date.js'

Vue.use(VueMasonry)
Vue.filter('date', DateFilter)
```

```
// nux.config.js

plugins: [
    '~/plugins/main',
    {
      src: '~/plugins/gtag.js',
      mode: 'client'
    }
  ],
```

Use plugins only on client

```
// nuxt-client-init.client.js
```

Define plugins only on client by file name.

export the context

```
export default async context => {
    await context.store.dispatch('nuxtClientInit', context)
}
```

STYLE

Apply auth navigation guard with middleware

```
// middleware/auth.js

import tokenService from "../services/token.service"
import TokenService from "../services/token.service"

export default ({ app, store, route }) => {

  const loggedIn = store.getters['auth.module/loadedStatus'].loggedIn
  // check if the user is logged by retrieving user object in cookie
  const user = tokenService.getUser()
  const isAuthenticated = user && user.accessToken


  const authRequired = route.meta.map((meta) => {
    if (meta.auth && typeof meta.auth !== 'undefined')
      return meta.auth
    return false
  })

  app.router.beforeResolve((to, from, next) => {
    if (authRequired && !isAuthenticated) {
        next('/user/login')
    } else {
        next();
    }
});


}
```

Then in the component to secure

```
export default {
  middleware: 'auth',
  meta: {
    auth: true,
  }
}
```

Nuxt.js page and router structure

```
/pages
--| /products
----| /edit
------| _product_id.vue
```

Vuex store example

```
export const state = () => ({
  })
  export const mutations = {
  }
  export const actions = {
  }
  export const getters = {
  }
  export const modules = {

  }
```

Initialize store with data in SSR

```
export const actions = {
 nuxtServerInit ({ dispatch }) {
   // localStorage should work, right?
   const token = localStorage.getItem('token')
   if (token) return dispatch('user/load', token)
 }
}
```

Initialize client

```
nuxtClientInit({ dispatch }, { req, res, app }) {
      const user = app.$cookies.get('user')
      if (user) {
        return dispatch('auth.module/load', user)
      }

    },
```

STYLE

Pass axios through the context to exploit the component in an external module

// plugins/axio.singleton.js

Configure a plugin

Create a unique instance of axios and pass it to external module

```
import AxiosSingleton from '~/services/axios.singleton.
service';
import { setClient } from '~/services/api'

export default ({ app }, inject) => {
   const axiosInstance = app.$axios;
   const axiosSingleton = AxiosSingleton.initAxios(axio-
sInstance);
   setClient(axiosSingleton)
   inject('axiosSingleton', axiosSingleton);
}
```

configure the plugin in nuxt.config.js

```
plugins: [
    '~/plugins/axios.singleton.plugin',
]
```

Use d3.js library

// nux.config.js

```
build: {
   standalone: true,

 }
```

Create a method in the component and call this methods from mounted

```
import * as d3 from 'd3';
```

```
html,body{
    overflow-x: hidden;
}
```

Prevent horizontal scrolling, applied to default.vue css

Check that no element has an offset or a width more than 100%

COMMON

Upload multiple image to express backend

On the client, posting form with a controller that puts the data in form to send it by http request.

vue.js component method

```
 <b-button variant="primary" @click="onPickFile">U-
pload images </b-button>
        <input type="file" multiple style="display: none"
ref="fileInput" accept="image/*" @change="onFile-
Picked">


onPickFile() {
     this.$refs.fileInput.click()
   },
   onFilePicked(event) { // upload multiple images
     const files = event.target.files

     if (files.length > 0) {

       for (let i = 0; i < files.length; i++) {
         const fileReader = new FileReader()
         fileReader.addEventListener('load', () => {
           this.imageUrls.push(fileReader.result)
         })
         fileReader.readAsDataURL(files[i])
         this.images.push(files[i])

       }
```

```
html,body{
    overflow-x: hidden;
}
```

Prevent horizontal scrolling, applied to default.vue css