

Galaxy ntf marketplace

CIS 4339 - Enterprise A Development

Final Project



May 3, 2021

Shameen Rizvee Yaseer (1661391)

Professor Fardosht Amirpanahi

TABLE OF CONTENTS

[Real World Application 2](#_Toc70976723)

[Instructions on Where to Begin 2](#_Toc70976724)

[Client Application 3](#_Toc70976725)

[Database Model 5](#_Toc70976726)

[Performance Screenshot 8](#_Toc70976727)

# Real World Application

**Galaxy NTF Marketplace** is an online social platform and a crypto currently/digital asset trading platform. The goal of this website is NTF enthusiast and Cryptocurrency fanatics can Buy, Sell, Trade their Digital Assets without the struggles of Big Bank regulations and centralized big money tactics. The goal of this application is not only to generate a steady pace of income by doing business in the stocks, options, and digital asset market, but to also provider a platform for people to sell their NTF assets at a much easier way.

# Instructions on Where to Begin

In order to get started using this application, there are a couple steps that need to be taken beforehand. When you unzip the folder jsFinal.zip, you will be able to access the uncompressed file and inside you will see two folders – ‘client’ and ‘server’.

Client is where the front-end of the application is going to be, it is built under the framework of VueJS. The first thing that will need to be done is to install the node\_modules folder as this will install all the libraries needed to run this application. Import this folder ‘jsFinal’ onto VSCode.exe or your choice of IDE. Your next step will be to open up a terminal where you can run codes which help you install and run this application. Based on your current folder directory, you should be in this folder URL:

*C:\Users\(your computer name)\VSProjects\jsFinal>*

Your goal is to navigate into the ‘client’ folder, so enter ‘cd client’ into to change directory to the client folder, your new URL should be:

*C:\Users\(your computer name)\VSProjects\jsFinal\client*

Now that you are in the client folder, run ‘npm install’ in the terminal and this will install all the dependencies needed to run this application. Let’s move to the server side.

The folder named ‘server’ is the Server-side of the application, and this is where the database information, schema, and functionality is stored. We will need to either open a new terminal, which will bring you back to the original URL:

*C:\Users\(your computer name)\VSProjects\jsFinal>*

From this directory you can follow the previous steps and change directory into ‘server’ and then installing node\_module, example of the folder URL:

*C:\Users\(your computer name)\VSProjects\jsFinal\server*

or to make it easier and in one line of code, you can run ‘cd server && npm install’ which changes to the directory and installs node\_modules.

With these two important steps done, now you can navigate back to “*C:\Users\(your computer name)\VSProjects\jsFinal\client”* and enter the command “npm run serve” to start the service of the front-end application. Then navigate on your second terminal to “*C:\Users\(your computer name)\VSProjects\jsFinal\server”* and enter the command “npm run dev” to start the database.

If in any case a Library was not installed properly, you can manually install the dependencies based on the ones listed in the following sections:

Client Application & Database Model

# Client Application

**Dependencies:** Axios, Bootstrap + Bootstrap Vue, Cores-js, Cors, Vue, Vue-router, Vuex

**Component Description:**

Home: Home.vue

This is the landing page for the website, it is routed as the ${url}/ so when the customer wants to reach my website, this is the first page they will see. Here the customer has an option to either login or register.

About: About.vue

This is the Summary page for the website and also the Frequently Asked Questions page, it is routed as the ${url}/about and here the Customer can read on the purpose of this page and also see the most popular FAQ we have.

Login Page: Login.vue

This is the Login page where the Customer can use their existing login credentials and access the contents of my website. This is done by using the login() services in my personServices.js page which handles the Axio’s requesting to my Database URL. If the request comes back without any issues then the page will store the information of the user within the Vuex Store of this application and also navigate the user to the Front page of the website.

Front page of the website: HomePage.vue

This is the home page of the website where the customer can access the marketplace and view, purchase, sell a digital asset which they own. At the moment we have three digital assets being sold and the customer has the ability to see a card-format quick information description and price of the item, and the ability to click for more info or click to instantly buy.

Registration Page: Register.vue

The Registration page is responsible for allowing the customer to create an account which will catch all the information from the input fields, store that into an object and pass that to Axio’s to send to my Database as an API request. This object will be store following the schema structure of my

Navigation bar: Navbar.vue

The Navigation Bar is used to help the Customer navigator around the website. There is also a logout link which will redirect the Customer out of the website, which will cause them to lose their current login tokens, meaning they will need to re-login in order to access the website again.

Updating and Deleting Page: Person.vue

The Update and Delete page are merged into one page, and this is where the Customer is able to update and/or delete their current information off the database. This is done by the help of the update() function from the personService.js and the delete() function from the personService.js file. The update function will take the input fields as an object to pass onto the database, then the database will use the email to match the customers information and then update it with the input field parameters.

The delete function works by taking the input field as an object which will be passed onto the personService.js to call on the delete() function. This function will pass a request via URL to my database, which will tell my database that I want to execute a /delete function. In my database routing it will take the id parameter that was carried with the object and locate this within the database and delete it.

**Routing and Services:**

Location for the routing of the front-end: \jsFinal\client\src\router\index.js

Location for the Services of the front-end: \jsFinal\client\src\personServices.js

**Vuex Store:**

The Vuex Store for my application acts as the storage for the Customer, which allows me to store the Customers credentials as they are accessing the website. This allows me to callback on the current user’s first and last name and reuse it on whatever page I want.

It also allows me to use getter and setter methods which are critical in providing a fluid experience while browsing the page. This gives me the ability to continuedly edit and update the stored information that is saved as the customer is browsing my pages.

# Database Model

**Dependencies:** Bcryptjs, Cors, Express, Jsonwebtoken, Mongdb, Mongoose, Nodemon Validator

**Database Schema and Functions:**

firstName: {

        type: String,

        required: true,

        trim: true,

    },

    lastName: {

        type: String,

        required: true,

        trim: true,

    },

    email: {

        type: String,

        required: true,

        trim: true,

        unique: true,

        lowercase: true,

        validate(value) {

            if (!validator.isEmail(value)) {

                throw new Error('Invalid email');

            } //This is a server-side validation to see if the email format is valid

        },

    },

    password: {

        type: String,

        required: true,

        trim: true,

        minlength: 4,

        validate(value) {

            if (value.toLowerCase().includes('password')) {

                throw new Error('Password cannot be "password"');

            } //This is a server-side validation to see if the email format is valid, also if password == "password"

        },

    },

    lastLogin: {

        type: String,

        required: false,

    },

    tokens: [

        {

            token: {

                type: String,

                required: true,

            },

        },

    ],

Location of Schema and functions: jsFinal\server\src\model\person.js

I have included **Middleware** in this application will run before any other function is called.

The first middleware is the Password Hasher:

//middleware Area- before anything happens

personSchema.pre('save', async function (next) {

    const person = this;

    if (person.isModified('password')) {

        person.password = await bcrypt.hash(person.password, 8);

    }

    next();

}); //checks the pre-saved password and changes it with a hash crypt

This takes an object named ‘person’ from whatever application entry is being called, then modifies the ‘password’ entry being caught onto the server-side and by using the library BryptJS and one of it’s method ‘hash’, I am able to take the ‘person.password’ and encrypt the password into a 8 character random hash entry. This is done to protect the User’s credentials.

The Second middleware is the Token Generator:

//generate a token based on the parameters of id, name, and email

personSchema.methods.generateAuthToken = async function () {

    const person = this;

    const token = jwt.sign(

        {

            \_id: person.\_id,

            name: person.lastName,

            email: person.email,

        },

        'secret'

    );

    person.tokens = person.tokens.concat({ token });

    await person.save();

    return token;

}; //generate a token

The Token Generator is responsible for creating a token whenever the function generateAuthToken() is called. This is done with the help of the JsonWebToken library and using its jwt.sign() function. By using the parameters of the person’s id, name, and email we are able to generate a unique token for that user. This is done to help create unique access for specific users.

**Routing for the Database:**

Location of routing: jsFinal\server\src\router\router.js

Since our database is connection on the localhost:3000 port, Axio’s (from the Client-side) will send out API request to that URL and the router. This will catch the request being sent from the client and match it with the back-end functions that we have created for our application.

We have a /register which can register a new account. We have /login which actually just checks to see if this entry is currently existing in the DB. We have a /search which is similar to the /login function except we are generating a new login validation for the user (using tokens) but rather just double checking if this user exists or not and returning that user as an object. We also have a /delete which is used to delete an entry in the DB based on the parameters of ID which we provided from the client-side.

**Starting the Database:**

Location of Connection: jsFinal\server\src\ mongooseConnection.js

Location of MongoDB properties: jsFinal\server\src\app.js

My Database was created using MongoDB’s Atlas application so I did not create my Database instance on a Locally hosted server, instead my server is resting on the cloud and waiting for my next command to execute. Because of this, all my data is always populating onto my website and I don’t run into the problem of not having any sample data to work with.

# Performance Screenshot

