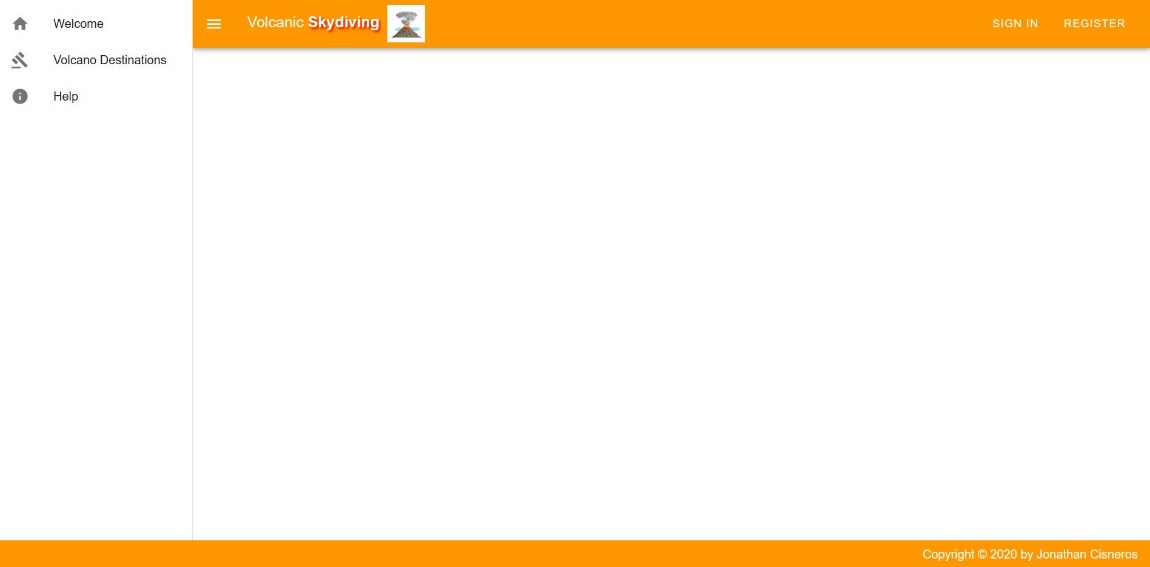
Application Technical Document

Setting up my application:

App.vue is the foundational stucture for which all my components are loaded. A brief explanation is given for each of my components in the user guide document. The App.vue component displays the navigation drawer, header and footer and stays the same throughout each view. The components are loaded in the center white space.



To download MongoDB: <https://www.mongodb.com>. Go to software, then community server and download MongoDB community server.

To install MongoDB: Click on the msi file from your download folder

To run MongoDB on a terminal: type “mongo”.

To create my database, I typed “use volcano”

To show my collections, I typed “show collections”

To insert data into my collections: “db.volcanos.insert({“name”: “property”})

Steps I took to import MongoDB into my application:

const mongoose = require("mongoose"),

  url = "mongodb://127.0.0.1:27017",

  dbName = "volcano",

  connectionString = url + "/" + dbName;

from app.js

require("./mongooseConnect");

Volcano CRUD Scripts:

---------------------RETRIEVE ALL VOLCANO SCRIPTS (GET)--------------------------

//endpoint to retrieve all volcanos

router.get("/", (req, res) => {

  Volcano.find(

    {},

    "name description image location status elevation price",

    (error, volcanos) => {

      if (error) {

        console.log(error);

      }

      res.send({

        volcanos,

      });

    }

  );

});

//used vuex to retrieve all volcanos

state: {

    volcanos: [],

  },

getters: {

    fetchVolcanos: (state) => state.volcanos,

  },

mutations: {

  GET\_VOLCANOS: (state, payload) => {

      state.volcanos = payload;

    },

  }

actions: {

    fetchVolcanos: (context, payload) => {

      axios({

        method: "get",

        url: "http://localhost:5000/api/volcanos/",

      })

        .then((response) => {

         //committing a mutation called GET\_VOLCANOS, which mutates the vaule of volcanos in the state

          context.commit("GET\_VOLCANOS", response.data.volcanos);

        })

        .catch(() => {});

    },

computed: {

    volcanos() {

      //once fetched successfully, it is mapped the volcanos variable used in the template

      return this.$store.getters.fetchVolcanos;

    }

  },

  mounted() {

    //dispatching an action called "fetchVolcanos"

    this.$store.dispatch("fetchVolcanos");

  }

--------------------------FETCH A SINGLE VOLCANO (GET BY ID)---------------------

//endpoint to retrieve a single volcano

router.get("/:id", (req, res) => {

  Volcano.findById(

    req.params.id,

    "name description image location status elevation price",

    (error, volcano) => {

      if (error) {

        console.error(error);

      }

      res.send(volcano);

    }

  );

});

//From Api.js file

import axios from "axios";

export default () => {

  return axios.create({

    baseURL: `http://localhost:5000/api/volcanos`,

    //baseURL = `api/posts`;

  });

};

//From VolcanoService.js File

  getVolcano(params) {

    return Api().get("/" + params.id);

  }

 //method fetching single volcano from API

  methods: {

    async getVolcano() {

      const response = await VolcanoService.getVolcano({

        id: this.$route.params.id

      });

      this.name = response.data.name;

      this.description = response.data.description;

      this.image = response.data.image;

      this.location = response.data.location;

      this.status = response.data.status;

      this.elevation = response.data.elevation;

      this.price = response.data.price;

    }

--------------------------UPDATE VOLCANO SCRIPTS (PUT)--------------------------

//endpoint to update volcanos

router.put("/:id", (req, res) => {

  Volcano.findById(

    req.params.id,

    "name description image location status elevation price",

    function (error, volcano) {

      if (error) {

        console.error(error);

      }

      volcano.name = req.body.name;

      volcano.description = req.body.description;

      volcano.image = req.body.image;

      volcano.location = req.body.location;

      volcano.status = req.body.status;

      volcano.elevation = req.body.elevation;

      volcano.price = req.body.price;

      volcano.save(function (error) {

        if (error) {

          console.log(error);

        }

        res.send({

          success: true,

        });

      });

    }

  );

});

updateVolcano(params) {

    return Api().put("/" + params.id, params);

  },

//method to take in params to update volcano

async updateVolcano() {

      await VolcanoService.updateVolcano({

        id: this.$route.params.id,

        name: this.name,

        description: this.description,

        image: this.image,

        location: this.location,

        status: this.status,

        elevation: this.elevation,

        price: this.price

      });

      swal("Success!", `The Volcano has been updated!`, "success");

      this.$router.push({ name: "Volcano" });

    },

-----------------------ADD VOLCANO SCRIPTS (POST)--------------------------------

//Endpoint to add a new volcano

router.post("/", (req, res) => {

  const newVolcano = new Volcano({

    name: req.body.name,

    description: req.body.description,

    image: req.body.image,

    location: req.body.location,

    status: req.body.status,

    elevation: req.body.elevation,

    price: req.body.price,

  });

  newVolcano.save((error, volcano) => {

    if (error) {

      console.log(error);

    }

    res.send(volcano);

  });

});

ADD\_VOLCANO: (state, payload) => {

      state.volcanos.unshift(payload);

    },

//committing a mutation called ADD\_VOLCANOS, which mutates the vaule of volcanos in the state

 addVolcano: (context, payload) => {

      return axios({

        method: "post",

        data: payload,

        url: "http://localhost:5000/api/volcanos/",

        headers: {

          "Content-Type": "application/json",

        },

      })

        .then((response) => {

          context.commit("ADD\_VOLCANO", response.data);

          swal("Great!", "Volcano added successfully!", "success");

        })

        .catch((error) => {

          console.log(error);

          swal("Oh no!", "Could not add the Volcano!", "error");

        });

    },

 methods: {

    submit() {

      //if the form is filled out, the document is added to the db by updating the store

      if (this.$refs.form.validate()) {

        const volcano = {

          name: this.name,

          description: this.description,

          image: this.image,

          location: this.location,

          status: this.status,

          elevation: this.elevation,

          price: this.price

        };

        //dispatching the action "addVolcano" along with the variable"

        this.$store.dispatch("addVolcano", volcano);

        this.$refs.form.reset();

        this.$router.push({ name: "Volcano" }); //redirects the user to the homepage

      }

      return true;

    },

    clear() {

      this.$refs.form.reset();

    }

  }

------------------------DELETE VOLCANO SCRIPTS-------------------------------

//Endpoint to delete a Volcano

router.delete("/:id", (req, res) => {

  Volcano.remove(

    {

      \_id: req.params.id,

    },

    function (error, volcano) {

      if (error) {

        console.error(error);

      }

      res.send({ success: true });

    }

  );

});

 deleteVolcano(id) {

    return Api().delete("/" + id);

  },

//delete volcano by taking in the \_id parameter and making axios call

    async deleteVolcano(id) {

      VolcanoService.deleteVolcano(id);

      swal("Success!", `The Volcano has been Deleted!`, "success");

      this.$router.push({ name: "Volcano" });

    }

MEMBER SCRIPTS

//endpoint to enable a member to register

----------------------------ADD MEMBER (CREATE SCRIPTS)------------------------

router.post("/register", (req, res) => {

  const firstName = req.body.firstName;

  const lastName = req.body.lastName;

  const email = req.body.email;

  const password = req.body.password;

  const newMember = new Member({

    firstName,

    lastName,

    email,

    password,

  });

  Member.createMember(newMember, (error, member) => {

    if (error) {

      console.log(error);

    }

    res.send({ member });

  });

});

//hits the api endpoint, takes the parameters and responds with correct message

return axios({

          method: "post",

          data: {

            firstName: this.firstName,

            lastName: this.lastName,

            email: this.email,

            password: this.password

          },

          url: "http://localhost:5000/api/members/register",

          headers: {

            "Content-Type": "application/json"

          }

        })

          .then(() => {

            swal(

              "Success!",

              "You have been successfully registered!",

              "success"

            );

            this.$router.push({ name: "Login" });

          })

//endpoint to enable a member to login

router.post("/login", (req, res) => {

  if (req.body.email && req.body.password) {

    const email = req.body.email;

    const password = req.body.password;

    //checks to see if the member with the given email exists

    Member.fetchMemberByEmail(email, (err, member) => {

      if (!member) {

        //if member does not exist in our database

        res.status(404).json({ message: "Member does not exist" });

      } else {

        //compares the password we provided with the members password in the app

        Member.comparePW(password, member.password, (error, isMatch) => {

          if (error) throw error;

          //signs the user's payload with the jwt signature, generates a token, and responds with that token

          if (isMatch) {

            const payload = { id: member.id };

            const token = jwt.sign(payload, jwtOptions.secretOrKey);

            res.json({ message: "ok", token });

            //if passwords do not match

          } else {

            res.status(401).json({ message: "Incorrect Password" });

          }

        });

      }

    });

  }

});

//reaches the api endpoint, takes params and responds with correct message

return axios({

        method: "post",

        data: {

          email: this.email,

          password: this.password

        },

        url: "http://localhost:5000/api/members/login",

        headers: {

          "Content-Type": "application/json"

        }

      })

        .then(response => {

          window.localStorage.setItem("auth", response.data.token);

          swal("Login Success!", "You are ready to start!", "success");

          this.$router.push({ name: "Volcano" });

        })

        .catch(error => {

          const message = error.response.data.message;

          swal("Login Denied", `${message}`, "error");

          this.$router.push({ name: "Login" });

        });

**Dependencies used server side:**

**Body-parser:** Used to parse parameters of the body that I provide when making an HTTP request.

**Cors:** Handle cross-origin requests between frontend as well as backend.

**Express:** Node.js JavaScript framework used for building scalable web applications on top of Node.js.

**Mongoose:** For adding validations and typecasting fields in a document.

**Morgan:** Middleware for logging HTTP requests.

**Bcryptjs**: A package used to encrypt passwords. Prevents users from viewing passwords in plain text.

**Jsonwebtoken**: A mechanism of authenticating requests using a token-based approach.

**Passport**: A middleware provided by Node.js for authentication requests made to the server.

**Dependencies used client side:**

**Axios:** Used for making the HTTP requests from Node.js and makes Ajax calls from the frontend.

**Bootstrap, Bootstrapvue, and Vuetify:** Modules and UI framework used to build materialistic web page designs for Vue.js applications.

**Vuex, store, vuex-persistedstate:** A library used with Vue.js to manage different states in my application. Provides a centralized place to store all of the data in my application.

**Vue:** JavaScript framework for building user interfaces.

**Vue-router:** The official router for Vue.js applications. Used for making a single page application